Non-Negative Residual Matrix Factorization with Application to Graph Anomaly Detection[*]

Hanghang Tong
IBM T.J. Watson Research Center
htong@us.ibm.com

Ching-Yung Lin
IBM T.J. Watson Research Center
chingyung@us.ibm.com

Abstract

Given an IP source-destination traffic network, how do we spot mis-behavioral IP sources (e.g., port-scanner)? How do we find strange users in a user-movie rating graph? Moreover, how can we present the results intuitively so that it is relatively easier for data analysts to interpret?

We propose *NrMF*, a non-negative residual matrix factorization framework, to address such challenges. We present an optimization formulation as well as an effective algorithm to solve it. Our method can naturally capture abnormal behaviors on graphs. In addition, the proposed algorithm is linear wrt the size of the graph therefore it is suitable for large graphs. The experimental results on several data sets validate its effectiveness as well as efficiency.

1 Introduction

Graphs appear in a wide range of settings, e.g., social networks, computer networks, user-movie rating graphs in collaborative filtering, the world wide web, biological networks, and many more. How can we find patterns, e.g. communities and anomalies, in a large sparse graph?

Naturally, low-rank approximations on the adjacency matrices of the graph provide powerful tools to answer the above questions. Formally, let $\mathbf{A}$ be the adjacency matrix of the graph, a rank-r approximation of matrix $\mathbf{A}$ is a matrix $\tilde{\mathbf{A}}$ where $\tilde{\mathbf{A}}$ is of rank r and the residual matrix $(\mathbf{A} - \tilde{\mathbf{A}})$ has small norm. The low-rank approximation is usually presented in a factorized form e.g., $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{R} = \mathbf{FG} + \mathbf{R}$ where $\mathbf{F}$, $\mathbf{G}$ are the factorized matrices of rank-r, and $\mathbf{R}$ is the residual matrix. The factorized matrices $\mathbf{F}$ and $\mathbf{G}$ can naturally reveal the community structure in the graph. The residual matrix $\mathbf{R}$, on the other hand, is often a strong indicator for anomalies on graphs (e.g., a large norm of the residual matrix $\mathbf{R}$ suggests a significant deviation from low-rank structure in the graph).

From *algorithmic* aspect, a recent trend in matrix factorization is to improve the interpretation of such graph mining results. To name a few, non-negative matrix factorization methods [31] restrict the entries in $\mathbf{F}$ and $\mathbf{G}$ to be non-negative; example-based methods [15] generate sparse decomposition by requiring the columns of the matrix $\mathbf{F}$ to be the actual columns of the original matrix $\mathbf{A}$; etc. By imposing such non-negativity and/or sparseness constrains on the *factorized matrices*, it is relatively easier to interpret the community detection results. Actually, it is now widely realized that non-negativity is a highly desirable property for interpretation since negative values are usually hard to interpret. However, most, if not all, of these constrains (i.e., non-negativity, sparseness, etc) are imposed on the *factorized matrices*. Consequently, these existing methods are tailored for the task of community detection. It is not clear how to improve the interpretation for the task of anomaly detection from the algorithmic aspect. Can we impose similar constraints (e.g., non-negativity) on the *residual matrix* $\mathbf{R}$ to improve the interpretation for graph anomaly detection?

From *application* side, it is often the case that anomalies on graphs correspond to some actual behaviors/activities of certain nodes. For instance, we might flag an IP source as a suspicious port-scanner if it *sends packages* to a lot of destinations in an IP traffic network [37]; an IP address might be under the DDoS (distributed denial-of-service) attack if it *receives packages* from many different sources [37]; a person is flagged as 'extremely multi-desciplinary' if s/he *publishes papers* in many remotely related fields in an author-conference network [2]; in certain collusion-type of fraud in financial transaction network, a group of users always *give good ratings* to another group of users in order to artificially boost the reputation of the target group [9], etc. If we map such behaviors/activities (e.g., 'sends/receives packages', 'publishes papers', 'gives good ratings', etc) to the language of matrix factorization, it also suggests that the corresponding entries in the residual matrix $\mathbf{R}$ should be non-negative.

In response to such challenges, in this paper, we propose

[*]Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

a new matrix factorization (*NrMF*) for the task of graph anomaly detection. To the best of our knowledge, we are the first to address the interpretation issue of matrix factorization for the purpose of graph anomaly detection. The major contributions of the paper can be summarized as follows:

1. Problem formulation, presenting a new formulation for matrix factorization (*NrMF*) tailored for graph anomaly detection;
2. An effective algorithm (*AltQP-Inc*) to solve the above optimization problem, linear wrt the size of the graph;
3. Proofs and analysis, showing the effectiveness as well as the efficiency of the proposed method;
4. Experimental evaluations, demonstrating both the effectiveness and efficiency of the proposed method.

The rest of the paper is organized as follows: we introduce notation and formally define the problem (*NrMF*) in Section 2. We present and analyze the proposed solution for *NrMF* in Section 3 and Section 4, respectively. We provide experimental evaluations in Section 5. The related work is reviewed in Section 6. Finally, we conclude in Section 7.

2 Problem Definitions

Table 1: Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper case)
$\mathbf{A}(i,j)$	the element at the i^{th} row and j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}(i,:)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:,j)$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{a}, \mathbf{b}, \ldots$	column vectors (bold lower case)
$\mathbf{F}$,$\mathbf{G}$	factorized matrices of $\mathbf{A}$
$\mathbf{R}$	residual matrix of $\mathbf{A}$
n	number of type 1 objects in $\mathbf{A}$
l	number of type 2 objects in $\mathbf{A}$
m	number of edges in $\mathbf{A}$
r	rank size

Table 2 lists the main symbols we use throughout the paper. In this paper, we consider the most general case of bipartite graphs. We represent a general bipartite graph by its adjacency matrix[1]. Following the standard notation, we use capital bold letters for matrices (e.g. $\mathbf{A}$), lower case bold letters for vectors (e.g. $\mathbf{a}$). We denote the transpose with a prime (i.e., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). We use subscripts to denote the size of matrices/vectors (e.g. $\mathbf{A}_{n\times l}$ means a matrix of size $n \times l$). When the size of a matrix or a vector is clear from the context, we ignore such subscripts for brevity. Also, we represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{A}(i,j)$ is the element at the i^{th} row and j^{th} column of the matrix $\mathbf{A}$, and $\mathbf{A}(:,j)$ is the j^{th} column of $\mathbf{A}$, etc.

[1]In practice, we store these matrices using an adjacency list representation, since real graphs are often very sparse.

With the above notations, a general matrix factorization problem can be formally defined as follows:

PROBLEM 1. *Matrix Factorization*

Given: *A graph $\mathbf{A}_{n\times l}$, and the rank size r;*

Find: *Its low-rank approximation structure. That is, find (1) two factorized matrices $\mathbf{F}_{n\times r}$ and $\mathbf{G}_{r\times l}$, and the residual matrix $\mathbf{R}_{n\times l}$; such that (1) $\mathbf{A}_{n\times l} \approx \mathbf{F}_{n\times r}\mathbf{G}_{r\times l}$, and (2) $\mathbf{R}_{n\times l} = \mathbf{A}_{n\times l} - \mathbf{F}_{n\times r}\mathbf{G}_{r\times l}$.*

Existing matrix factorization techniques can be viewed as different instantiations of Problem 1. They differ from each other, mainly from the following two aspects: (1) by using the different metrics to measure the approximation accuracy (some norms on the residual matrix $\mathbf{R}$); and (2) by imposing the different constraints on the *factorized matrices* $\mathbf{F}$ and $\mathbf{G}$. For example, non-negative matrix factorization requires the factorized matrices to be non-negative (see Section 6 for a review).

In this paper, we present another instantiation of Problem 1 by imposing the non-negativity constrains on the *residual matrix* $\mathbf{R}$. Our problem, Non-Negative Residual Matrix Factorization (*NrMF*), is formally defined as follows:

PROBLEM 2. *Non-Negative Residual Matrix Factorization (NrMF)*

Given: *A graph $\mathbf{A}_{n\times l}$, and the rank size r;*

Find: *Its low-rank approximation structure. That is, find two factorized matrices $\mathbf{F}_{n\times r}$ and $\mathbf{G}_{r\times l}$, and the residual matrix $\mathbf{R}_{n\times l}$; such that (1) $\mathbf{A}_{n\times l} \approx \mathbf{F}_{n\times r}\mathbf{G}_{r\times l}$; (2) $\mathbf{R}_{n\times l} = \mathbf{A}_{n\times l} - \mathbf{F}_{n\times r}\mathbf{G}_{r\times l}$; and (3) for all $\mathbf{A}(i,j) > 0, \mathbf{R}(i,j) \geq 0$.*

Problem 2 is tailored for the task of graph anomaly detection, where we explicitly require the corresponding elements $\mathbf{R}(i,j)$ in the *residual matrix* $\mathbf{R}$ to be non-negative if there exists an edge between node i and node j in the original graph (i.e., $\mathbf{A}(i,j) > 0$). As explained earlier in Section 1, the residual matrix $\mathbf{R}$ is often a good indicator for anomalies on graphs. Moreover, many abnormal behaviors/activities (e.g., port-scanner, DDoS, etc) can be mapped to some non-negative entries in the residual matrix $\mathbf{R}$. For instance, a large entry in $\mathbf{R}$ might indicate a strange interaction between two objects; a heavy row/column of $\mathbf{R}$ might indicate a suspicious object (e.g., port-scanner, or an IP address that is under DDoS attack, etc). In *NrMF*, we aim to capture such abnormal behaviors/activities by explicitly imposing non-negativity constrains on the residual matrix $\mathbf{R}$. Moreover, *NrMF* directly brings the non-negativity, an interpretation-friendly property, to the task of graph anomaly detection since negative values are usually hard to interpret. For example, by existing matrix factorization methods, the data analyst has to look at (somewhat abstract) residual matrix, which contains both positive and negative entries; and calculate the re-construction errors to spot anomalies. In contrast, thanks to the non-negativity constraints in *NrMF*, we can present the residual matrix itself as a residual graph, which might be more intuitive for the data analyst to interpret.

3 The Proposed Solutions for *NrMF*

In this section, we formally represent our solutions for non-negative residual matrix factorization (*NrMF*). We first formulate Problem 2 as an optimization problem; and then we present effective algorithms to solve it.

3.1 Optimization Formulations

General Formulation of Problem 2. Formally, Problem 2 can be formulated as the following optimization problem:

$$\begin{aligned} \text{argmin}_{\mathbf{F},\mathbf{G}} \quad &= \quad \|\mathbf{R}_{n\times l} \otimes \mathbf{W}_{n\times l}\|_F^2 \\ &= \quad \sum_{i=1}^{n}\sum_{j=1}^{l}(\mathbf{A}(i,j) - \mathbf{F}(i,:)\mathbf{G}(:,j))^2\mathbf{W}(i,j)^2 \\ \text{s.t.} \quad & \quad \text{for all } \mathbf{A}(i,j) > 0 : \\ & \quad \mathbf{F}(i,:)\mathbf{G}(:,j) \le \mathbf{A}(i,j) \end{aligned} \tag{3.1}$$

In eq. (3.1), '$\otimes$' means element-wise multiplication. In other words, here we use a weighted squared Frobenius norm of the residual matrix $\mathbf{R}$ to measure the approximation accuracy, through a weight matrix $\mathbf{W}_{n\times l}$. For every edge in the graph (i.e., $\mathbf{A}(i,j) > 0$), we require that $\mathbf{F}(i,:)\mathbf{G}(:,j) \le \mathbf{A}(i,j)$, which means that the corresponding residual entry $\mathbf{R}(i,j)$ should satisfy that $\mathbf{R}(i,j) = \mathbf{A}(i,j) - \mathbf{F}(i,:)\mathbf{G}(:,j) \ge 0$.

0/1 Weight Matrix for e.q. (3.1). In eq. (3.1), the weight matrix $\mathbf{W}$ reflects the user's preference among all $n \times l$ reconstructed entries. In this paper, we focus on a special case of weight matrix $\mathbf{W}$: $\mathbf{W}(i,j) = 1$ for $\mathbf{A}(i,j) > 0$; and $\mathbf{W}(i,j) = 0$ otherwise. This means that we only measure the element-wise loss on the observed edges; and among all these edges, we treat the element-wise loss equally (referred to as '0/1 Weight Matix'). This type of weight matrix in widely used in the literature, especially in the context of collaborative filtering [5, 35].

With such 0/1 weight matrix, e.q. (3.1) can be simplified as:

$$\begin{aligned} \text{argmin}_{\mathbf{F},\mathbf{G}} \quad & \sum_{i,j,\ \mathbf{A}(i,j)>0} (\mathbf{A}(i,j) - \mathbf{F}(i,:)\mathbf{G}(:,j))^2 \\ \text{s.t.} \quad & \text{for all } \mathbf{A}(i,j) > 0 : \\ & \mathbf{F}(i,:)\mathbf{G}(:,j) \le \mathbf{A}(i,j) \end{aligned} \tag{3.2}$$

In the rest of this paper, we will focus on eq. (3.2) for clarity. However, we would like to point out that the upcoming proposed techniques can be naturally applied to a general, arbitrary weight matrix $\mathbf{W}$. We will present such algorithms in the appendix for completeness.

Rank-1 Approximation for e.q. (3.2). In e.q. (3.2), if we restrict the rank of the factorized matrices $\mathbf{F}$ and $\mathbf{G}$ to be 1, we have the following rank-1 approximation of e.q. (3.2), where $\mathbf{f}$ is an $n \times 1$ column vector and $\mathbf{g}$ is a $1 \times l$ row vector.

$$\begin{aligned} \text{argmin}_{\mathbf{f},\mathbf{g}} \quad & \sum_{i,j,\ \mathbf{A}(i,j)>0} (\mathbf{A}(i,j) - \mathbf{f}(i)\mathbf{g}(j))^2 \\ \text{s.t.} \quad & \text{for all } \mathbf{A}(i,j) > 0 : \\ & \mathbf{f}(i)\mathbf{g}(j) \le \mathbf{A}(i,j) \end{aligned} \tag{3.3}$$

3.2 The Proposed Optimization Algorithms

Next, we present our algorithms to solve e.q. (3.2). We first analyze the challenges of optimizing e.q. (3.2) directly, and then present an incremental alternative optimization strategy.

3.2.1 Challenges

Unfortunately, the optimization problem formulated in e.q. (3.2) is *not convex* wrt $\mathbf{F}$ and $\mathbf{G}$ jointly due to the coupling between $\mathbf{F}$ and $\mathbf{G}$ in both the objective function and the inequality constraints. Therefore, it might be unrealistic to seek for a global optimal solution. A natural way to handle this issue is to find $\mathbf{F}$ and $\mathbf{G}$ *alternatively*. Actually, we can show that if we fix either $\mathbf{G}$ or $\mathbf{F}$ in (3.2), the resulting optimization problem is a convex quadratic programming problem wrt the remaining matrix ($\mathbf{F}$ or $\mathbf{G}$). This suggests the following greedy optimization strategy (referred to as *AltQP-Batch*, see the appendix for the formal description): after some initialization, we alternatively update $\mathbf{F}$ and $\mathbf{G}$ using convex quadratic programming until convergence. With *AltQP-Batch*, we can find a local minimal solution for e.q. (3.2), which is acceptable in terms of optimization quality for a non-convex problem. However, most, if not all, of existing convex quadratic programming methods are *polynomial* wrt the number of variables. This makes the overall complexity of *AltQP-Batch* to be polynomial, which might not scale very well for large graphs.

To address these challenges, in this paper, we propose an effective and efficient algorithm *AltQP-Inc*. The basic idea of *AltQP-Inc* is to find the resulting $\mathbf{F}$ and $\mathbf{G}$ *incrementally*: at each iteration, we try to find a rank-1 approximation on the current residual matrix by solving e.q. (3.3). As we will show soon, this strategy bears the similar greedy nature as *AltQP-Batch*. Therefore it also leads to a local minimal solution for e.q. (3.2). Yet its time complexity is *linear* wrt the size of the graph, which makes the algorithm more suitable for large graphs. Next, we first present our algorithm (*AltQP-Inc*-1) for solving e.q. (3.3), and then present our algorithm (*AltQP-Inc*) for solving e.q. (3.2).

3.2.2 *AltQP-Inc*-1: Proposed Algorithm for e.q. (3.3)

Again, e.q. (3.3) is not convex wrt $\mathbf{f}$ and $\mathbf{g}$ jointly due to the coupling between $\mathbf{f}$ and $\mathbf{g}$. Therefore, we seek for an alternative strategy: the algorithm alternatively iterates as follows until convergence: (1) updating $\mathbf{f}$ while keeping $\mathbf{g}$ fixed; and (2) updating $\mathbf{g}$ while keeping $\mathbf{f}$ fixed.

Formally, let us consider how to update $\mathbf{g}$ while keeping $\mathbf{f}$ fixed (updating $\mathbf{f}$ is similar as updating $\mathbf{g}$). In this case, e.q. (3.3) can be further simplified as:

$$\begin{aligned} \text{argmin}_{\mathbf{g}} \quad & \sum_{i,j,\ \mathbf{A}(i,j)>0} (\mathbf{A}(i,j) - \mathbf{f}(i)\mathbf{g}(j))^2 \\ \text{s.t.} \quad & \text{for all } \mathbf{A}(i,j) > 0 : \\ & \mathbf{f}(i)\mathbf{g}(j) \leq \mathbf{A}(i,j) \end{aligned} \tag{3.4}$$

It is easy to show that e.q. (3.4) is convex wrt $\mathbf{g}$. The proposed algorithm (*Update-g*) for solving e.q. (3.4) is summarized in Alg. 1. At each outer loop of Alg. 1, we update a single entry $\mathbf{g}(j)(j = 1, ..., l)$, which is in turn done by some closed formula (steps 19-25). The main difference between *Update-g* and *AltQP-Batch* is as follows: in *Update-g*, $\mathbf{g}$ is a row vector and we can have computationally cheap closed formula to solve e.q. (3.4). In contrast, we have to call some expensive convex quadratic programming packages in *AltQP-Batch* to find the optimal solution.

Based on Alg. 1, we present Alg. 2 (*Rank-1-Approximation*) to solve e.q. (3.3): after some initializations (step 1), Alg. 2 alternates between the following two steps until convergence: (1) update $\mathbf{g}$ while keeping $\mathbf{f}$ fixed by calling *Update-g* (step 3); and (2) update $\mathbf{f}$ while keeping $\mathbf{g}$ fixed by calling *Update-g* (step 4).

Algorithm 1 *Update-g* (For Solving e.q. 3.4)

```
Input: The original matrix $\mathbf{A}_{n\times l}$; and a column vector $\mathbf{f}_{n\times 1}$
Output: A row vector $\mathbf{g}_{1\times l}$
1: for $j = 1 : l$ do
2:     Initialize the lower bound low $= -$inf, upper bound up $=$ inf, $t = 0$ and $q = 0$;
3:     for each $i$, s.t., $\mathbf{A}(i,j) > 0$ do
4:         Update: $q \leftarrow q + \mathbf{f}(i)\mathbf{A}(i,j)$
5:         Update: $t \leftarrow t + \mathbf{f}(i)^2$
6:         if $\mathbf{f}(i) > 0$ then
7:             Update: up $=$ min(up, $\mathbf{A}(i,j)/\mathbf{f}(i)$)
8:         else if $\mathbf{f}(i) < 0$ then
9:             Update: low $=$ max(low, $\mathbf{A}(i,j)/\mathbf{f}(i)$)
10:        else
11:            Continue;
12:        end if
13:    end for
14:    if $t == 0$ then
15:        Set: $\mathbf{g}(j) = 0$;
16:        Continue;
17:    end if
18:    Set: $q \leftarrow q/t$
19:    if $q <=$ up and $q >=$ low then
20:        Output: $\mathbf{g}(j) = q$;
21:    else if $q >$ up then
22:        Output: $\mathbf{g}(j) =$ up;
23:    else
24:        Output: $\mathbf{g}(j) =$ low;
25:    end if
26: end for
```

3.2.3 *AltQP-Inc*: Proposed Algorithm for e.q. (3.2)

Based on Alg. 2, our algorithm (*AltQP-Inc*) for solving the original e.q. (3.2) is summarized in Alg. 3. It is an incremental algorithm: at each iteration, it calls Alg. 2 to find a rank-1 approximation for the current residual matrix $\mathbf{R}$ (steps 3-4). Notice that since e.q. (3.2) is an instantiation of e.q. (3.1) by using the 0/1 weight matrix, we only need to update the residual entries where there exists an edge in the original graph (i.e., $\mathbf{A}(i,j) > 0$) in steps 5-7.

4 Analysis of the Proposed Algorithms

In this section, we analyze the effectiveness as well as the efficiency of the proposed algorithms. Our main results are (1) the proposed algorithms find (at least) a local optimal solution for the corresponding optimization problems; and (2) the complexity of the proposed algorithms is linear in both time and space.

4.1 Effectiveness of the Proposed Algorithms

The effectiveness of the proposed algorithms is summarized in Lemma 4.1, which basically says that the proposed *AltQP-Inc* finds a local minima of e.q. (3.2). Given that the optimization problem in e.q. (3.2) is not convex wrt $\mathbf{F}$ and $\mathbf{G}$ jointly, such a local minima is acceptable in terms of the optimization quality.

LEMMA 4.1. **Effectiveness.** *(P1) Update-g in Alg. 1 gives the global optimal solution for the optimization problem in e.q.* (3.4)*; (P2) Rank-1-Approximation in Alg. 2 finds a local minima of the optimization problem in e.q.* (3.3)*; and (P3) AltQP-Inc in Alg. 3 finds a local minima for the optimization problem in e.q.* (3.2)*.*

Sketch of Proof: For brevity, we only give the proof for (P1); since (P2) and (P3) are relatively straight-forward based on (P1).

Here, the key point is that e.q. (3.4) can be decomposed into the following l independent optimization problems, each of which only involves a single variable $\mathbf{g}(j)$ $(j = 1, ..., l)$:

$$\begin{aligned} \text{For} \quad & j = 1, ..., l : \\ \text{argmin}_{\mathbf{g}_j} \quad & \sum_{i,\ \mathbf{A}(i,j)>0} (\mathbf{A}(i,j) - \mathbf{f}(i)\mathbf{g}(j))^2 \\ \text{s.t.} \quad & \text{for all } \mathbf{A}(i,j) > 0 : \\ & \mathbf{f}(i)\mathbf{g}(j) \leq \mathbf{A}(i,j) \end{aligned} \tag{4.5}$$

Algorithm 2 *Rank-1-Approximation* (For Solving e.q. 3.3)

```
Input: The original matrix $\mathbf{A}_{n\times l}$
Output: A column vector $\mathbf{f}_{n\times 1}$; and a row vector $\mathbf{g}_{1\times l}$;
1: Initialize $\mathbf{f}_{n\times 1}$ and $\mathbf{g}_{1\times l}$;
2: while Not convergent do
3:     Update: $\mathbf{g} \leftarrow$ Update-g$(\mathbf{A}, \mathbf{f})$
4:     Set: $\tilde{\mathbf{f}} \leftarrow$ Update-g$(\mathbf{A}', \mathbf{g}')$
5:     Update: $\mathbf{f} = \tilde{\mathbf{f}}'$
6: end while
```

Algorithm 3 *AltQP-Inc* (For Solving e.q. 3.2)

```
Input: The original matrix $\mathbf{A}_{n\times l}$, and rank size $r$
Output: An $n \times 1$ matrix $\mathbf{F}$; a $r \times l$ matrix $\mathbf{G}$; and an $n \times l$
    matrix $\mathbf{R}$
1: Initialize $\mathbf{F} = \mathbf{0}_{n\times r}$, $\mathbf{G} = \mathbf{0}_{r\times l}$, and $\mathbf{R} = \mathbf{A}$
2: for $k = 1 : r$ do
3:     $(\mathbf{f}, \mathbf{g}) \leftarrow$ Rank-1-Approximation$(\mathbf{R})$
4:     Set $\mathbf{F}(:, k) = \mathbf{f}$, and $\mathbf{G}(k, :) = \mathbf{g}$
5:     for every $(i, j)$, s.t., $\mathbf{A}(i, j) > 0$ do
6:         Update $\mathbf{R}(i, j) \leftarrow \mathbf{R}(i, j) - \mathbf{f}(i)\mathbf{g}(j)$
7:     end for
8: end for
```

For a given j, e.q. (4.5) is equivalent to[2] :

$$\begin{aligned}
\text{argmin}_{\mathbf{g}(j)} \quad & \mathbf{g}(j)^2 - 2q\mathbf{g}(j) \\
\text{s.t.} \quad & \text{low} \le \mathbf{g}(j) \le \text{up} \\
\text{where:} \quad & q = (\sum_{i,\mathbf{A}(i,j)>0} \mathbf{f}(i)\mathbf{A}(i,j))/(\sum_{i,\mathbf{A}(i,j)>0} \mathbf{f}(i)^2) \\
& \text{low} = \max_{\mathbf{f}(i)<0,\mathbf{A}(i,j)>0}\{\mathbf{A}(i,j)/\mathbf{f}(i)\} \\
(4.6) \quad & \text{up} = \min_{\mathbf{f}(i)>0,\mathbf{A}(i,j)>0}\{\mathbf{A}(i,j)/\mathbf{f}(i)\}
\end{aligned}$$

In e.q. (4.6), we have a quadratic objective function wrt a single variable $\mathbf{g}(j)$, where $\mathbf{g}(j)$ has a boundary constraint (low $\le \mathbf{g}(j) \le$ up). It is easy to verify that each outer loop of Alg. 1 gives the global optimal solution for (4.6). Therefore, the whole Alg. 1 gives the global optimal solution for e.q. (3.4), which completes the proof. □

4.2 Time Efficiency of the Proposed Algorithms

The time complexity of the proposed algorithms is summarized in Lemma 4.2, which basically says that for all the three algorithms we proposed, they are linear wrt the size of graph m, n and l. Therefore, they are scalable for large graphs.

LEMMA 4.2. **Time Complexity.** *(P1) Update-g in Alg. 1 requires $O(m+l)$ time; (P2) Rank-1-Approximation in Alg. 2 requires $O(mt+nt+lt)$ time; and (P3) AltQP-Inc in Alg. 3 requires $O(nrt+mrt+lrt)$ time, where t is the maximum iteration number in Alg. 2.*

[2]We have dropped a constant term from the objective function since it does not affect the optimal solution.

Proof of P1: The time cost for step 2 of Alg 1 is $O(1)$. Let m_j be the total number of non-zero elements in the j^{th} column of matrix $\mathbf{A}$, we have $\sum_{j=1}^{l} m_j = m$. The time cost for step 3 and 13 is $O(m_j)$ since we need $O(1)$ operations for each non-zero element in $\mathbf{A}(:, j)$. The cost for steps 14-17 is $O(1)$. We need another $O(1)$ time for step 18. Finally, for steps 19-25, we need $O(1)$ time. Therefore, the total cost for Alg. 1 is $\sum_{j=1}^{l}(O(1) + O(m_j)) = O(l) + O(\sum_{j=1}^{l} m_j) = O(m + l)$, which completes the proof. □

Proof of P2: Step 1 in Alg. 2 takes $O(l + n)$ time. Based on (P1), we need $O(m + l)$ and $O(m + n)$ for step 3 and 4, respectively. We need another $O(n)$ for step 5. Therefore, the overall time complexity of Alg. 2 is $O(l+n) + (O(m+l)+O(m+n)+O(n))t = O(mt+nt+lt)$, which completes the proof. □

Proof of P3: Step 1 in Alg. 3 takes $O(nr + lr + m)$ time. Let $\tilde{m}_k$ be the number of non-zeros elements in $\mathbf{R}$ in the k^{th} iteration of Alg. 3, we have that $\tilde{m}_1 = m$ and $\tilde{m}_k \le m$ $(k = 2, ..., r)$. Based on (P2), we need $O(\tilde{m}_k t + nt + lt)$ for step 3. For step 4, we need $O(n + l)$ time. We need additional $O(m)$ time for updating $\mathbf{R}$ (steps 5-7). Putting these together, the overall time complexity of Alg. 3 is $O(nr + lr + m) + \sum_{k=1}^{r} O(\tilde{m}_k t + nt + lt + m + n + l) = O(mrt + nrt + lrt)$, which completes the proof. □

4.3 Space Efficiency of the Proposed Algorithms

The space complexity of the proposed algorithms is summarized in Lemma 4.3, which basically says that for all the three algorithms we proposed, the space complexity is linear wrt the size of graph m, n and l. Therefore, they are scalable for large graphs.

LEMMA 4.3. **Space Complexity.** *(P1) Update-g in Alg. 1 requires $O(m + n + l)$ space; (P2) Rank-1-Approximation in Alg. 2 requires $O(m + n + l)$ space; and (P3) AltQP-Inc in Alg. 3 requires $O(m + nr + lr)$ space.*

Proof of P1: In Alg. 1, we need $O(m)$, $O(n)$, and $O(l)$ space to keep the original matrix $\mathbf{A}$, the column vector $\mathbf{f}$, and the row vector $\mathbf{g}$, respectively. For all the remaining steps in Alg. 1, they requires $O(1)$ space respectively. Among the different iterations of Alg. 1, we can re-use the space from the previous iteration. Therefore, the overall space complexity of Alg. 1 is $O(m + n + l)$, which completes the proof. □

Proof of P2: In Alg. 2, we need $O(m)$ space for the original matrix $\mathbf{A}$. The initialization in step 1 needs $O(n + l)$ space. By (P1), we need $O(m + n + l)$ space for steps 3-4, respectively. Step 5 tasks another $O(n)$ space. Among the different iterations of Alg. 1, we can re-use the space from the previous iteration. Therefore, the overall space complexity of Alg. 2 is $O(m) + O(n + l) + O(2m + 2n + 2l) + O(n) = O(m + n + l)$, which completes the proof.□

Table 2: Data sets used in evaluations

Name	$n \times l$	m
MIT-DP	103×97	5,449
NIPS-PW	2,037× 13,649	1,624,335
CIKM-PA	1,895×952	2,664
MovieLens	6,040×3,952	575,281

Proof of P3: In Alg. 3, we need $O(m)$ space for the original matrix $\mathbf{A}$. The initialization in step 1 needs $O(nr + lr + m)$ space. Let $\tilde{m}_k$ be the number of non-zeros elements in $\mathbf{R}$ in the k^{th} iteration, we have that $\tilde{m}_1 = m$ and $\tilde{m}_k \leq m$ $(k = 2, ..., r)$. Based on (P2), we need $O(m_k + n + l)$ for step 3. For steps 4-7, they do not require extra space. Finally, among different iterations, we can reuse the space from the first iteration since $\tilde{m}_k < \tilde{m}_1 = m(k = 1, ..., r)$. Therefore, the overall space complexity of Alg. 3 is $O(m) + O(nr + lr + m) + O(max(m_1) + n + l) = O(m + nr + lr)$, which completes the proof. □

5 Experimental Results

In this section, we present experimental evaluations, after we introduce the data sets. All the experiments are designed to answer the following two questions:

- *Effectiveness:* What kinds of anomalies can the proposed *AltQP-Inc* detect?
- *Efficiency:* How fast is the proposed *AltQP-Inc*? How does it scale?

5.1 Data Sets

We use four different data sets in our experiments, summarized in Table 2.

The first data set (**MIT-DP**) is from MIT Reality Mining project[3]. Rows represent the blue tooth devices and columns represent the persons. The un-weighted edges represent the scanning activities between the devices and persons. In total, there are 103 devices, 97 persons and 5,449 scanning activities.

NIPS-PW is from the NIPS proceedings[4]. Rows represent papers and columns represent words. Weighted edges represent the count of the words that appear in the corresponding papers. In total, there are 2,037 authors, 13,649 words, and 1,624,335 edges.

CIKM-PA is an author-paper graph constructed from CIKM proceedings[5]. Rows represent the authors and columns represent the papers. We connect a given paper with all of its co-authors by edges. In total, we have 1,895 authors, 952 papers and 2,664 edges.

MovieLens is a user-movie rating graph[6]. Rows represent users and columns represent movies. If a user has given a positive rating (4 or 5) to a particular movie, we connect them with an edge. Here, the edge weight is the actual rating (4 or 5). In total, we have 6,040 users, 3,952 movies, and 575,281 edges.

5.2 Effectiveness Results

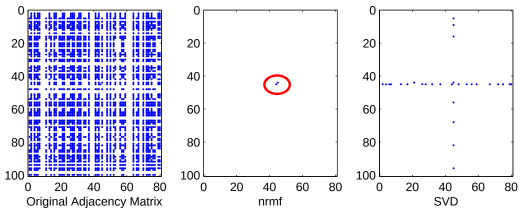
In this paper, we focus on the following four types of anomalies on bipartite graphs:

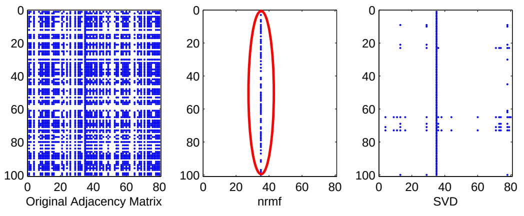
1. *Strange connection* (referred to as 'strange connection'). It is a connection between two nodes which belong to two remotely connected communities, respectively. For example, in author-conference graph, this could be the case that an author publishes a paper in a conference which is remotely related to his/her major research interest (e.g., a system guy publishes a paper in a theory conference, etc) [36].
2. *Port-scanning like behavior* (referred to as 'port-scan'). It is a type-1 node that is connected to many different type-2 nodes in the bipartite graph. For example, in an IP traffic network, this could be an IP source which sends packages to many different IP destinations (therefore it might be a suspicious port scanner) [37].
3. *DDoS like behavior* (referred to as 'ddos'). It is a type-2 node that is connected to many different type-1 nodes in the bipartite graph. For example, in an IP traffic network, this could be an IP destination which receives packages from many different IP sources (therefore it might be under DDoS, distributed denial-of-service, attack) [37].
4. Collusion type of fraud (referred to as 'bipartite core'). It is a group of type-1 nodes and a group of type-2 nodes which are tightly connected with each other. For example, in financial transaction network, this could be a group of users who always give good ratings to another group of users in order to artificially boost the reputation of the target group [9].

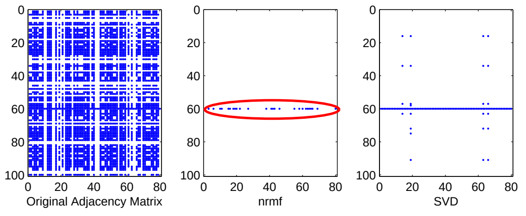
Since we do not have the ground-truth for the anomalies, we use the following methodology for evaluation: we randomly inject one of the above anomalies into the original (normal) graph, and see if the proposed algorithm can spot it from the top-k edges of the residual matrix $\mathbf{R}$.

Qualitative Results. Since the residual elements in $\mathbf{R}$ by the proposed *AltQP-Inc* are non-negative, we can plot the residual $\mathbf{R}$ itself as a residual graph as follows. The residual graph has the same node sets as the original graph $\mathbf{A}$. For each edge (i, j) in $\mathbf{A}$ (i.e., $\mathbf{A}(i, j) > 0$), we put an edge between node i and node j in the residual graph if $\mathbf{R}(i, j) > 0$ with the weight $\mathbf{R}(i, j)$. Compared with the traditional matrix factorization methods (where one has to calculate and look at the abstract re-construction error for anomalies), the residual graph might provide a more intuitive way to spot anomalies on graphs. Figure 1 presents an illustrative example on synthetic graphs. For each sub-figure, we inject

[3]http://reality.media.mit.edu/
[4]http://www.cs.toronto.edu/~roweis/data.html
[5]http://www.informatik.uni-trier.de/~ley/db/conf/cikm/
[6]http://www.grouplens.org/


(a) strange connection


(b) port scanning


(c) ddos

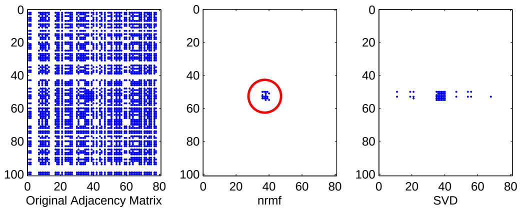

(d) bipartite core

Figure 1: Anomaly detection on synthetic graphs. Each blue dot in the figures represents an edge (or non-zero elements) in the graph or in the residual matrices. The anomalies detected by the proposed *AltQP-Inc* are marked by red circles. (best viewed in color)

one of the four anomalies into the normal graphs, and plot the original matrix (left), the top-k edges in the residual matrix by *AltQP-Inc* (middle) and the residual matrix by singular value decomposition (SVD) (right). It can be seen that in all cases, the corresponding anomalies clearly stand out in the corresponding residual matrix by the proposed *AltQP-Inc* (middle figures). On the other hand, (1) SVD does not always capture the corresponding anomalies (e.g., (a)), and/or (2) there might be some noise in the residual matrix by SVD (e.g., (b-d)). In addition, since the residual entries in SVD can be both positive and negative, we cannot plot the residual matrix by SVD as an intuitive residual graph.

Quantitative Results. We also present the quantitative results on the four real data sets. For each data set, we inject one of the four anomalies into the data set randomly. We then run the proposed *AltQP-Inc* to find the residual matrix and output its top-k edges as anomalies. We repeat each of such experiments 20 times and report the mean accuracy and variance in figure 2. It can be seen that *AltQP-Inc* achieves high detection accuracy for all the four types of anomalies, across all the four data sets.

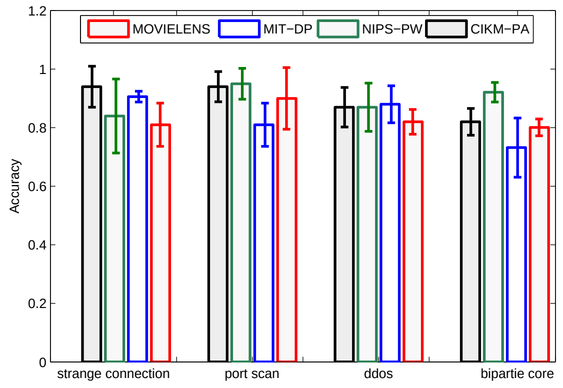


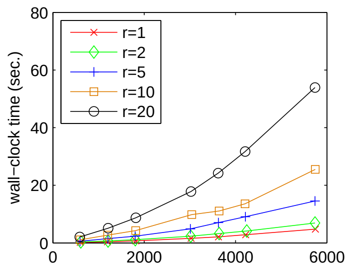
Figure 2: Anomaly detection on real graphs by the proposed *AltQP-Inc*. The proposed *AltQP-Inc* achieves high accuracy to detect all the four types of anomalies.

5.3 Efficiency Results

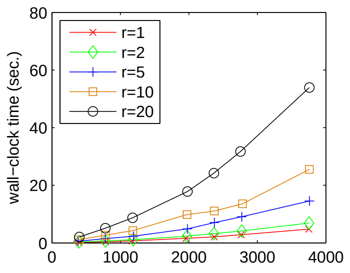
Here, we evaluate the efficiency of the proposed *AltQP-Inc*. For the results we reported in this subsection, they are tested on the same machine with four 3.0GHz Intel (R) Xeon (R) CPUs and 16GB memory, running Linux (2.6 kernel). We repeat the experiments 10 times and report the mean wall-clock time.

First, we compare the wall-clock time between the proposed *AltQP-Inc* and *AltQP-Batch* (see the appendix for the description of *AltQP-Batch*). The result is presented in figure 3. In figure 3, the number inside the parenthesis beside the name of the data sets is the ratio between the re-construction error by *AltQP-Inc* and that by *AltQP-Batch*. It can be seen that the proposed *AltQP-Inc* is much faster than *AltQP-Batch*. For example, *AltQP-Inc* is 51x faster (3.6sec. vs. 186sec.) than *AltQP-Batch* on **MovieLens** data set. Note that the ratio between the re-construction error by *AltQP-Inc* and that by *AltQP-Batch* is always less than or equal to 1, indicating that the optimization solution by *AltQP-Inc* is better than (**MIT-DP** and **MovieLens**) or similar to (**NIPS-PW** and **CIKM-PA**) that by *AltQP-Batch*.

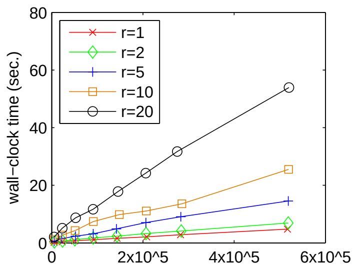
Next, we test the scalability of *AltQP-Inc* using the subsets of the **MovieLens** data set with the different rank size r. The result is presented in figure 4. It can be seen that the proposed *AltQP-Inc* scales linearly wrt the graph size (n,



(a) wall-clock time vs. n



(b) wall-clock time vs. l



(c) wall-clock time vs. m

Figure 4: Scalability of the proposed *AltQP-Inc* with different rank size r. *AltQP-Inc* scales linearly wrt the size of the graph.

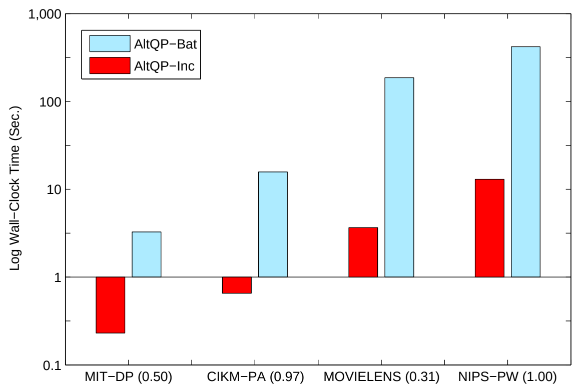


Figure 3: Comparison of wall-clock time. The wall-clock time is in logarithmic scale. The number inside the parenthesis beside the name of the data sets is the ratio between the re-construction error by *AltQP-Inc* and that by *AltQP-Batch*. The proposed *AltQP-Inc* is much faster than *AltQP-Batch*, with better or similar re-construction error.

l and m).

6 Related Work

In this section, we review the related work, which can be categorized into three parts: matrix factorization, anomaly detection and general graph mining.

Matrix Factorization. Matrix factorization [21, 15, 1] plays a very important role in graph mining. The most popular choices include SVD/PCA [21, 26] and random projection [25]. However, these methods often ignore the sparseness and nonnegativity of many real graphs and lead to dense and negative results, which make the results hard to interpret. A recent trend in matrix factorization has been devoted to improving the interpretation of the mining results. For example, to address the sparseness issue, the example-based factorization methods have been proposed [15, 37, 38]. By requiring the columns of the factorized matrix $\mathbf{F}$ be to actual columns from the original matrix $\mathbf{A}$, the factorization is naturally sparse and therefore good for interpretation. To address the non-negativity issue, non-negative matrix factorization has been studied in the past few years. Pioneering work in this thread can be traced back to [31] and there are a lot of follow-up work in this direction [13, 29, 28, 12]. There are also efforts to address both the sparseness and non-negativity issues [23, 24]. It is worth pointing out that most, if not all, of these modifications (i.e., sparseness and non-negativity constrains) are imposed on the factorized matrices. As a result, they mainly improve the interpretation for the task of community detection. It is unclear how these efforts can also help to improve the interpretation for the task of anomaly detection. This is exactly one major motivation of this work. By imposing the non-negativity constrains on the *residual matrix*, instead of the factorized matrices, we bring this interpretation-friendly property (i.e., non-negativity) to graph anomaly detection.

Anomaly Detection. Noble et al was among the first to detect abnormal sub-graphs using MDL (minimum description length) criteria [34]. Follow-up work along this criteria includes [7, 16]. In [2], the authors proposed using ego-net to detect abnormal nodes on weighted graphs. In [36], the authors proposed using proximity to detect abnormal nodes and edges. The work in [37, 38] is most related to our work. In [37, 38], the authors use matrix factorization to detect port scanning like behavior by looking at the reconstruction error (certain norms of the residual matrix). One limitation of [37, 38] is that its residual matrix can be arbitrary numbers (either positive or negative). Therefore, the result might be too abstract and not intuitive for data analysts to interpret. We restrict the residual matrix to be non-negative so that we can plot it as an intuitive residual graph. From the application side, many graph anomalies correspond to some actual behaviors/activities of certain nodes (e.g., for port-scanner in an IP traffic network, it has connections to many different IP destinations). Such abnormal behaviors can be naturally captured by the corresponding non-negative entries in the residual matrix $\mathbf{R}$. For anomaly detection for other types of data, please refer to a recent comprehensive survey [8].

General Graph Mining. There is a lot of research work on static graph mining, including pattern and law mining [3, 14, 17, 6, 33], frequent substructure discovery [41], influence propagation [27], social networks compression [11] and community mining [18][19][20], etc. More recently, there is an increasing interest in mining time-evolving graphs, such as densification laws and shrinking diameters [32], community evolution [4], proximity tracking [39], conversation dynamics [30] and dynamic communities [10], etc.

7 Conclusion

In this paper, we present a novel matrix factorization (*NrMF*) paradigm, which aims to detect abnormal behaviors/activities on graphs in a more interpretable way. Our main contributions are:

1. Problem formulation, presenting a new formulation for matrix factorization tailored for graph anomaly detection;
2. An effective algorithm (*AltQP-Inc*) to solve the above optimization problem, linear wrt the size of the graph;
3. Proofs and analysis, showing the effectiveness as well as the efficiency of the proposed method (e.g., Lemma 4.1, Lemma 4.2, etc);
4. Experimental evaluations, demonstrating both the effectiveness and efficiency of the proposed method.

Future research directions include (1) extending *AltQP-Inc* to time-evolving graphs, and (2) parallelizing *AltQP-Inc* using Hadoop[7].

References

[1] D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2), 2007.

[2] L. Akoglu, M. McGlohon, and C. Faloutsos. oddball: Spotting anomalies in weighted graphs. In *PAKDD (2)*, pages 410–421, 2010.

[3] R. Albert, H. Jeong, and A.-L. Barabasi. Diameter of the world wide web. *Nature*, (401):130–131, 1999.

[4] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

[5] R. M. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD*, pages 95–104, 2007.

[6] A. Broder, R. Kumar, F. Maghoul1, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *WWW Conf.*, 2000.

[7] D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004.

[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 2009.

[7] http://hadoop.apache.org/

[9] D. H. Chau, S. Pandit, and C. Faloutsos. Detecting fraudulent personalities in networks of online auctioneers. In *PKDD*, pages 103–114, 2006.

[10] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD*, pages 153–162, 2007.

[11] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.

[12] C. H. Q. Ding, T. Li, and M. I. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):45–55, 2010.

[13] D. L. Donoho and V. Stodden. When does non-negative matrix factorization give a correct decomposition into parts? In *NIPS*, 2003.

[14] S. Dorogovtsev and J. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.

[15] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal of Computing*, 2005.

[16] W. Eberle and L. B. Holder. Mining for structural anomalies in graph-based data. In *DMIN*, pages 376–389, 2007.

[17] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, Aug-Sept. 1999.

[18] G. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3), Mar. 2002.

[19] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *9th ACM Conf. on Hypertext and Hypermedia*, pages 225–234, New York, 1998.

[20] M. Girvan and M. E. J. Newman. Community structure is social and biological networks.

[21] G. H. Golub and C. F. Van-Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[22] S.-P. Hong and S. Verma. A note on the strong polynomiality of convex quadratic programming. *Mathematical Programming*, 68:131–139, 1995.

[23] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[24] S. Hyvönen, P. Miettinen, and E. Terzi. Interpretable nonnegative matrix decompositions. In *KDD*, pages 345–353, 2008.

[25] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.

[26] K. V. R. Kanth, D. Agrawal, and A. K. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD Conference*, pages 166–176, 1998.

[27] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. *KDD*, 2003.

[28] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *ICDM*, pages 353–362, 2008.

[29] R. Kompass. A generalized divergence measure for nonnegative matrix factorization. *Neural Computation*, 19(3):780–791, 2007.

[30] R. Kumar, M. Mahdian, and M. McGlohon. Dynamics of conversations. In *KDD*, pages 553–562, 2010.

[31] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.

[32] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[33] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[34] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *KDD*, pages 631–636, 2003.

[35] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.

[36] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

[37] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *SDM*, 2007.

[38] H. Tong, S. Papadimitriou, J. Sun, P. S. Yu, and C. Faloutsos. Colibri: fast mining of large static and dynamic graphs. In *KDD*, pages 686–694, 2008.

[39] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.

[40] P. Tseng. Simple polynomial-time algorithm for convex quadratic programming. *Laboratory for Information and Decision Systems, Massachusetts Institute of Technology*, 1988.

[41] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB*, pages 709–720, 2005.

A Appendix

Throughout this paper, we have focused on the optimization problem in e.q. (3.2) by restricting ourselves to the 0/1 weight matrix. In this section, we give our algorithms for solving the optimization problem in e.q. (3.1) with a general weight matrix $\mathbf{W}$ for the purpose of completeness. We first generalize the proposed *AltQP-Inc* to handle the general weight matrix $\mathbf{W}$ (*AltQP-Inc-General*), and then give the alternative optimization algorithm (*AltQP-Batch*) for solving e.q. (3.1), using convex quadratic programming.

A.1 Generalized *AltQP-Inc* for e.q. (3.1)

In order to generalize the proposed *AltQP-Inc* to solve e.q. (3.1) with a general weight matrix $\mathbf{W}$, we first give the algorithm (*Update-General-g*) to solve the sub-problem expressed in e.q. (1.7). *Update-General-g* is for an arbitrary weight matrix $\mathbf{W}$ and is a natural generalization of *Update-g*. In *Update-General-g*, $\text{diag}(\mathbf{W}(:,j))$ is a diagonal matrix with diagonal elements being $\mathbf{W}(i,j)(i = 1, ..., n)$. Similar as *Update-g*, in *Update-General-g*, we update $\mathbf{g}(j)$ one by one in each outer loop. For each $\mathbf{g}(j)$, it can be solved in a closed formula (steps 20-26). This is due to the fact that the optimization problem in (1.7) can be decomposed into l independent optimization problems, each of which only involves a single variable $\mathbf{g}(j)(j = 1, ..., l)$.

$$\begin{aligned}
&\text{argmin}_{\mathbf{g}} && \sum_{i,j}((\mathbf{A}(i,j) - \mathbf{f}(i)\mathbf{g}(j)) \cdot \mathbf{W}(i,j))^2 \\
&\quad\text{s.t.} && \text{for all } \mathbf{A}(i,j) > 0 : \\
&(1.7) && \mathbf{f}(i)\mathbf{g}(j) \leq \mathbf{A}(i,j)
\end{aligned}$$

Algorithm 4 *Update-General-g* (For Solving e.q. 1.7)

```
Input: The original matrix $\mathbf{A}_{n\times l}$, the weight matrix $\mathbf{W}_{n\times l}$,
    and a column vector $\mathbf{f}_{n\times 1}$
Output: A row vector $\mathbf{g}_{1\times l}$
 1: for $j = 1 : l$ do
 2:    Initialize the lower bound low $= -$inf and upper
       bound up $=$ inf;
 3:    Compute: $\mathbf{a} = \text{diag}(\mathbf{W}(:,j)) \cdot \mathbf{A}(:,j)$
 4:    Compute: $\mathbf{b} = \text{diag}(\mathbf{W}(:,j)) \cdot \mathbf{f}$
 5:    Compute: $t = \mathbf{b}'\mathbf{b}$
 6:    if $t == 0$ then
 7:       Set: $\mathbf{g}(j) = 0$;
 8:       Continue;
 9:    end if
10:    Compute: $q = \mathbf{a}'\mathbf{b}/t$
11:    for each $i$ s.t. $\mathbf{A}(i,j) > 0$ do
12:       if $\mathbf{f}(i) > 0$ then
13:          Update: up $= \min(\text{up}, \mathbf{A}(i,j)/\mathbf{f}(i))$
14:       else if $\mathbf{f}(i) < 0$ then
15:          Update: low $= \max(\text{low}, \mathbf{A}(i,j)/\mathbf{f}(i))$
16:       else
17:          Continue;
18:       end if
19:    end for
20:    if $q <=$ up and $q >=$ low then
21:       Output: $\mathbf{g}(j) = q$;
22:    else if $q >$ up then
23:       Output: $\mathbf{g}(j) =$ up;
24:    else
25:       Output: $\mathbf{g}(j) =$ low;
26:    end if
27: end for
```

Based on Alg. 4, we have Alg. 5 (*AltQP-Inc-General*) to solve e.q. (3.1). *AltQP-Inc-General* is a natural generalization of *AltQP-Inc*. Similar as *AltQP-Inc*, *AltQP-Inc-General* tries to find the factorized matrices $\mathbf{F}$ and $\mathbf{G}$ in an incremental way. At each outer loop of Alg. 5, it finds a Rank-1 approximation on the current residual matrix $\mathbf{R}$ (steps 2-11). At the inner loop of Alg. 5 (steps 4-8), it calls *Update-General-g* to alternatively update $\mathbf{f}$ and $\mathbf{g}$, respectively. This alternative process will be iterated until convergence. After it finds a rank-1 approximation, we update the current residual matrix in step 10.

Algorithm 5 *AltQP-Inc-General* (For Solving e.q. 3.1)

```
Input: The original matrix $\mathbf{A}_{n\times l}$, the weight matrix $\mathbf{W}$, and
    rank size $r$
Output: An $n \times r$ matrix $\mathbf{F}$; a $r \times l$ matrix $\mathbf{G}$; and an $n \times l$
    matrix $\mathbf{R}$;
1: Initialize $\mathbf{F} = \mathbf{0}_{n\times r}$, $\mathbf{G} = \mathbf{0}_{r\times l}$, and $\mathbf{R} = \mathbf{A}$
2: for $k = 1 : r$ do
3:   Initialize $\mathbf{f}$ and $\mathbf{g}$
4:   while Not convergent do
5:     Update: $\mathbf{g} \leftarrow$ Update-General-g$(\mathbf{R}, \mathbf{W}, \mathbf{f})$
6:     Set: $\tilde{\mathbf{f}} \leftarrow$ Update-General-g$(\mathbf{R}', \mathbf{W}', \mathbf{g}')$
7:     Update: $\mathbf{f} = \tilde{\mathbf{f}}'$
8:   end while
9:   Set $\mathbf{F}(:,k) = \mathbf{f}$, and $\mathbf{G}(k,:) = \mathbf{g}$
10:  Update $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{f}\cdot\mathbf{g}$
11: end for
```

A.2 *AltQP-Batch* for e.q. (3.1)

We can also solve the optimization problem in e.q. (3.1) by convex quadratic programming. To this end, let us assume that we have a package ($\mathbf{x} = \text{QpProg}(\mathbf{T}, \mathbf{S}, \mathbf{u}, \mathbf{v})$) to solve the following quadratic programming problem in e.q. (1.8).

$$\begin{aligned} \text{argmin}_{\mathbf{x}_{d\times 1}} \quad &= \quad \mathbf{x}'\mathbf{T}_{d\times d}\mathbf{x} + \mathbf{u}'_{d\times 1}\mathbf{x} \\ \text{s.t.} \quad & \quad \mathbf{S}_{s\times d}\mathbf{x} \le \mathbf{v}_{s\times 1} \end{aligned} \tag{1.8}$$

In e.q. (1.8), $\mathbf{x}$ is a $d \times 1$ vector that we want to solve and the inequality holds element-wisely. If $\mathbf{T}$ is semi-positive definite, QpProg() requires *at least* $O(d^k)$ time[8], where $k > 1$ and it depends on the actual methods to solve quadratic programming (e.g., $k = 3.5$ for the method in [40], $k = 3$ for the method in [22], etc).

To solve e.q. (3.1) by convex quadratic programming, we first give the algorithm (*Batch-Update-G*) to solve the following optimization problem in e.q. (1.9), which is a sub-problem of the optimization problem in e.q. (3.1).

$$\begin{aligned} \text{argmin}_{\mathbf{G}} \quad &= \quad \sum_{i=1}^{n}\sum_{j=1}^{l}(\mathbf{A}(i,j) - \mathbf{F}(i,:)\mathbf{G}(:,j))^2\mathbf{W}(i,j)^2 \\ \text{s.t.} \quad & \quad \text{for all } \mathbf{A}(i,j) > 0 : \\ & \quad \mathbf{F}(i,:)\mathbf{G}(:,j) \le \mathbf{A}(i,j) \end{aligned} \tag{1.9}$$

Batch-Update-G is similar as *Update-g* except that: in each outer loop of *Batch-Update-G*, we find a single $r \times 1$ column vector $\mathbf{G}(:,j)(j = 1, ..., l)$. Whereas in each outer loop of *Update-g*, we find a single variable $\mathbf{g}(j)(j = 1, ..., l)$. This subtle point leads to a big difference in terms of the time complexity. In *Batch-Update-G*, we have to use expensive convex quadratic programming to find $\mathbf{G}(:,j)$; whereas in *Update-g*, we can use computationally cheap closed formula to find $\mathbf{g}(j)$. It can be shown that the quadratic programming problem in step 14 is semi-positive definite which takes at least $O(r^k)$ time, and the overall *Batch-Update-G* requires at least $O(m + nlr^2 + lr^k)$ time.

[8]Besides the polynomial term, there is usually an additional term in the time complexity which relates to the encoding length of the quadratic programming problem.

Algorithm 6 *Batch-Update-G* (For Solving e.q. 1.9)

```
Input: The original matrix $\mathbf{A}_{n\times l}$, the weight matrix $\mathbf{W}_{n\times l}$,
    and left matrix $\mathbf{F}_{n\times r}$
Output: The right matrix $\mathbf{G}_{r\times l}$
1: for $j = 1 : l$ do
2:   Compute: $\mathbf{a}_{n\times 1} = \text{diag}(\mathbf{W}(:,j)) \cdot \mathbf{A}(:,j)$
3:   Compute: $\mathbf{B}_{n\times r} = \text{diag}(\mathbf{W}(:,j)) \cdot \mathbf{F}$
4:   for $i = 1 : n$ do
5:     if $\mathbf{A}(i,j) > 0$ then
6:       Set: $\mathbf{v}(i) = \mathbf{A}(i,j)$
7:     else
8:       Set: $\mathbf{v}(i) = \text{inf}$
9:     end if
10:  end for
11:  Compute: $\mathbf{X} = \mathbf{B}'\mathbf{B}$
12:  Compute: $\mathbf{u} = -2\mathbf{B}'\mathbf{a}$
13:  Set: $\mathbf{S} = \mathbf{F}$
14:  Solve $\mathbf{G}(:,j) \leftarrow \text{QpProg}(\mathbf{T}, \mathbf{S}, \mathbf{u}, \mathbf{v})$
15: end for
```

Algorithm 7 *AltQP-Batch* (For Solving e.q. 3.1)

```
Input: The original matrix $\mathbf{A}_{n\times l}$, the weight matrix $\mathbf{W}$, and
    rank size $r$
Output: An $n \times r$ matrix $\mathbf{F}$; a $r \times l$ matrix $\mathbf{G}$; and an $n \times l$
    matrix $\mathbf{R}$;
1: Initialize $\mathbf{F}_{n\times r}$ and $\mathbf{G}_{r\times l}$
2: while Not convergent do
3:   Update: $\mathbf{G} \leftarrow$ Batch-Update-G$(\mathbf{A}, \mathbf{W}, \mathbf{F})$
4:   Set: $\tilde{\mathbf{F}} \leftarrow$ Batch-Update-G$(\mathbf{A}', \mathbf{W}', \mathbf{G}')$
5:   Update: $\mathbf{F} = \tilde{\mathbf{F}}'$
6: end while
7: Output: $\mathbf{R} = \mathbf{A} - \mathbf{FG}$
```

Based on *Batch-Update-G*, we have Alg. 7 (*AltQP-Batch*) to solve the problem in e.q. (3.1). In *AltQP-Batch*, after some initialization (step 1), we alternatively call *Batch-Update-G* to update $\mathbf{F}$ and $\mathbf{G}$ by fixing one of them. This alternative process will be iterated until convergence. It can be shown that the time complexity of *AltQP-Batch* is at least $O(mt + nlr^2t + lr^kt + nr^kt)$, where t is the maximum iteration number in *AltQP-Batch* and $k > 1$ relates to the actual methods to solve the convex quadratic programming. Compared with the complexity of the proposed *AltQP-Inc* ($O(mrt + nrt + lrt)$), *AltQP-Batch* is much more time consuming.