# Logistic-Tropical Decompositions and Nested Subgraphs

Sanjar Karaev
Max-Planck-Institut für Informatik
Saarland Informatics Campus
Saarbrücken, Germany
skaraev@mpi-inf.mpg.de

Saskia Metzler
Max-Planck-Institut für Informatik
Saarland Informatics Campus
Saarbrücken, Germany
smetzler@mpi-inf.mpg.de

Pauli Miettinen
Max-Planck-Institut für Informatik
Saarland Informatics Campus
Saarbrücken, Germany
pmiettin@mpi-inf.mpg.de

## ABSTRACT

Communities in graphs are usually modelled as (quasi-) cliques, but this is not the only – or even necessarily the best – model. Other models, such as stars, hyperbolic shapes, or core-periphery communities have been proposed as well. These can be generalized to nested subgraphs, i.e. graphs whose adjacency matrix is nested. In this paper, we study the problem of summarizing a graph as a union of nested subgraphs. We approach the problem by applying a recent characterization of nested graphs using rounding rank. We extend this characterization to sets of overlapping nested matrices using tropical algebra. This allows us to model the problem as a thresholded tropical matrix factorization, and to design an algorithm for a maximum-likelihood version of the problem. Our experiments show that our algorithm is very scalable and can find good summarizations using structures that cannot be concisely expressed in terms of normal matrix factorizations.

## CCS CONCEPTS

**•Information systems →Data mining; •Computing methodologies →Factorization methods;**

## KEYWORDS

tropical algebra, nested matrix, rounding rank

## 1 INTRODUCTION

Mining a concise set of dense subgraphs (quasi-cliques) that jointly explain most of the edges in a graph is a fundamental problem in graph mining. Quasi-cliques are usually identified as the *communities* of the graph, and the problem of finding them is called *community detection*. Many variants of the problem exist, depending on whether the communities are allowed to overlap or not (e.g. [5, 20]), whether the graph has labels [5], and so forth. Quasi-cliques, however, are not the only possible model for communities. The *core-periphery* model [4] in social sciences models communities having an L-shaped structure: there are densely connected people in the core of a social network with loose ties to the periphery. Recently there has been an increasing interest in finding non-clique communities [2, 10, 11, 13] and evidence that real-world communities assume shapes that range from star graphs to hyperbolic shapes and cliques [2, 11, 13].

In this paper we study the problem of identifying *nested* subgraphs from given undirected graphs. Nested subgraphs generalize many of the proposed shapes of communities, including stars, hyperbolic structures, core-periphery communities, and cliques. In addition, nested subgraphs are important in their own right, and are studied extensively, especially in ecology since the 1980's [16].

While nested subgraphs offer a generalized way of summarizing a graph with a collection of submatrices of its adjacency matrix, the existing work has required or assumed these submatrices to be non-overlapping. This is, however, unrealistic in many applications. For instance, in ecology, it would mean that no species can be part of different ecosystems, and widely-spread species would have to be artificially assigned into one ecosystem only; in social networks, similarly, no person could be a member of multiple communities. To overcome this problem, we allow the nested submatrices to overlap.

Overlapping community detection is often formalized as a matrix factorization problem [5, 20]. The key observation in these approaches is that rank-1 submatrices of the adjacency matrix correspond to cliques. Hence, representing the adjacency matrix as a union (or sum) of rank-1 matrices identifies the cliques. Re-writing the union (sum) of rank-1 matrices as a matrix product gives the standard matrix factorization formulation.[1]

Unfortunately, this approach does not work as such for nested subgraphs, as they are not rank-1 matrices in the conventional sense. Instead, we base our approach on the recent work on *rounding rank* [15] that gives us a convenient characterization of nested matrices (see also [3]). Another problem lies in the combination of nested graphs, as the characterization from rounding rank falls apart in higher-rank decompositions. The crux of our approach is to replace the standard algebra with *tropical* algebra [8, 9]: we will show that the characterization of nested matrices under rounding rank extends naturally to higher-rank tropical decompositions. Furthermore, as nested matrices do not allow any way of assessing the confidence of the algorithm, we relax the problem so that we can obtain the likelihood of the data under the model.

Our main contributions are as follows: (*i*) We *characterize the task of summarizing a graph with overlapping nested subgraphs* as a thresholded matrix factorization problem over the tropical algebra. This characterization provides a new view to the task of finding non-clique-looking subgraphs, and will facilitate the design of algorithms for the task. (*ii*) We present a *probabilistic formulation* of the problem, allowing the assessment of the likelihood of the data under the model. (*iii*) We present the *first algorithm for the problem.* The algorithm, `SLTF`, uses our matrix factorization framework, is *very scalable*, and *can identify nested submatrices* that would be impossible to find using conventional methods.

[1] The factorization is Boolean if we take the union [5], and standard or nonnegative if we take the sum of the rank-1 matrices [20].

## 2 BACKGROUND

For an undirected graph $G = (V, E)$, we use $N(v)$ to denote the neighbourhood of the vertex $v \in V$: $N(v) = \{u \in V : \{v, u\} \in E\}$. Throughout this paper, we will denote a matrix by upper-case boldface letters ($\boldsymbol{A}$), and vectors by lower-case boldface letters ($\boldsymbol{a}$). The $i$th row of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_i$ and the $j$th column by $\boldsymbol{A}^j$.

The union of two graphs over the same set of vertices, $G = (V, E)$ and $H = (V, E')$, is the union of their edges: $G \cup H = (V, E \cup E')$. Likewise, for two adjacency matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, which are binary $n$-by-$m$ matrices, we say their *union* $\boldsymbol{A} \cup \boldsymbol{B}$ is the element-wise logical OR of the entries of $\boldsymbol{A}$ and $\boldsymbol{B}$. That is, $(\boldsymbol{A} \cup \boldsymbol{B})_{ij} = \boldsymbol{A}_{ij} \vee \boldsymbol{B}_{ij}$.

The *threshold function* is defined as follows.

*Definition 2.1.* Let $\tau_\alpha : \mathbb{R} \to \{0, 1\}$. For a value $\alpha \in \mathbb{R}$, we define

$$\tau_\alpha(x) = \begin{cases} 1 & \text{if} \quad x \geq \alpha \\ 0 & \text{otherwise} \end{cases} . \tag{1}$$

Typical thresholds are $\alpha = 1/2$ in the binary domain, and $\alpha = 0$ when working with tropical algebra. If implicitly clear, we will omit the subscript $\alpha$ subsequently.

**Tropical algebra.** The tropical algebra (see e.g. [1]) will be used for combining nested submatrices.

*Definition 2.2.* The *max-plus* (or *tropical*) algebra is defined over the set of extended real numbers $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$ with operations $a \boxplus b = \max\{a, b\}$ (addition) and $a \boxdot b = a + b$ (multiplication). The identity elements for addition and multiplication are $-\infty$ and 0, respectively.

The tropical matrix product as well as the rank for tropical matrices are defined analogously to the classical counterparts:

*Definition 2.3.* For two matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n \times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k \times m}$ their *tropical matrix product* is defined as

$$(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} = \max_{s=1}^{k}\{\boldsymbol{B}_{is} + \boldsymbol{C}_{sj}\} . \tag{2}$$

*Definition 2.4.* We say that a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$ has tropical *rank-1* if it can be represented as an outer sum of two vectors, that is $\boldsymbol{A}_{ij} = \boldsymbol{b}_i + \boldsymbol{c}_j$ for some vectors $\boldsymbol{b} \in \overline{\mathbb{R}}^{n \times 1}$ and $\boldsymbol{c} \in \overline{\mathbb{R}}^{1 \times m}$.

If a matrix has tropical rank 1, the matrix obtained by its elementwise exponentiation is rank-1 in the classical sense. The (Schein/Barvinok) *rank* of a matrix $\boldsymbol{A}$ over the tropical algebra is defined analogously to the standard matrix rank as the least number $k$ of rank-1 matrices $\boldsymbol{S}_i$ whose (tropical) sum $\boldsymbol{S}_1 \boxplus \boldsymbol{S}_2 \boxplus \cdots \boxplus \boldsymbol{S}_k = \boldsymbol{A}$.

**Nested subgraphs.** Let $G = (V, E)$ be an undirected graph. We say $G$ is *nested* if we can order the vertices $v \in V$ in a sequence $(v_1, v_2, \ldots, v_n)$ such that $N(v_{i+1}) \subseteq N(v_i)$ for all $i = 1, \ldots, n - 1$. For the equivalent definition in the adjacency matrix, we use the concept of a step function. Let $[n] = \{1, 2, \ldots, n\}$ and $[m] = \{1, 2, \ldots, m\}$. We say that function $s \colon [n] \to [m]$ is a *step function* if $s(i) \geq s(j)$ for all $i$ and all $j < i$. A binary matrix $\boldsymbol{A} \in \{0, 1\}^{n \times m}$ is *directly nested* if there exists a step function $s$ such that on each row $i \in [n]$ of $\boldsymbol{A}$, $a_{ij} = 1$ if $j \leq s(i)$ and $a_{ij} = 0$ if $j > s(i)$. $\boldsymbol{A}$ is *nested* if there exists a way to permute its rows and columns such that the permuted matrix $\boldsymbol{A}'$ is directly nested. Notice that this definition applies also to asymmetric and even non-square matrices, making it more general than the graph-based definition.

An equivalent way to define a nested matrix is by using the concept of rounding rank [15]. A binary matrix $\boldsymbol{A} \in \{0, 1\}^{n \times m}$ has *nonnegative rounding rank* of 1 if and only if there exist nonnegative vectors $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ and $\boldsymbol{y} \in \mathbb{R}^m_{\geq 0}$ such that $A = \tau(\boldsymbol{x}\boldsymbol{y}^T)$. Notice that we apply the threshold function independently to each element of the matrix $\boldsymbol{x}\boldsymbol{y}^T$. The nonnegative rounding rank is connected to nestedness as the following proposition shows:

PROPOSITION 2.5 ([15]). *Let $\boldsymbol{A}$ be an arbitrary binary matrix. $\boldsymbol{A}$ is nested if and only if it has nonnegative rounding rank of 1.*

## 3 RELATED WORK

Community detection is an important problem in data analysis; for the purpose of this paper, the methods based on matrix factorizations are the most relevant ones (e.g. [3, 5, 20]). Often, the communities are assumed to be quasi-cliques, and NMF [20] and Boolean matrix factorization [5] have been proposed to find overlapping quasi-cliques.

On the other hand, the 'beyond blocks' movement has argued that the communities come in more varied shapes than just blocks [2, 4, 10, 11, 13]; stars, biclique cores, and chains are some examples of the types of communities considered. Studies on real-world graphs support this argument [2, 13].

Many of these community models are a special case of nested subgraphs [12, 16]. Although for example [7] used SVD to solve the segmented nestedness, the connection between nestedness and continuous matrix factorizations was unclear until the recent characterization of nested matrices via rounding rank [15]. Independently, [3] proposed an algorithm called `FastStep` for doing thresholded nonnegative matrix factorization. Each rank-1 component of `FastStep` is nested after thresholding, but the full factorization is *not* a union of nested matrices (see Section 4.2). Logistic PCA (`LPCA`) [17] is an earlier approach for finding thresholded factorizations, although it uses also negative values (and hence has no nested structure) and the algorithm is not scalable.

Subtropical matrix factorizations have recently been proposed for finding 'winner-takes-it-all' type decompositions [8, 9] instead of NMF's 'parts-of-whole', although earlier examples of using the maximum operator instead of the sum exist as well (e.g. [19]).

## 4 PROBLEM DEFINITIONS AND THEORY

The main problem studied in this paper is to (approximately) express a given unweighted graph $G = (V, E)$ as a union of nested subgraphs $N_i = (V_i \subseteq V, E_i)$. Formally, for $\boldsymbol{A}$ as the adjacency matrix of $G$:

*Problem 4.1 (Covering by nested submatrices).* Given a binary matrix $\boldsymbol{A}$ and an integer $k$, find $k$ nested binary matrices $\boldsymbol{N}_1, \boldsymbol{N}_2, \ldots, \boldsymbol{N}_k$ such that their union $\tilde{\boldsymbol{A}} = \bigcup_{\ell=1}^{k} \boldsymbol{N}_\ell$ minimizes

$$\sum_i \sum_{j : j \neq i} \left| \boldsymbol{A}_{ij} - \tilde{\boldsymbol{A}}_{ij} \right| . \tag{3}$$

Notice that in (3), we do not consider the diagonal, ignoring the potential self-loops. If the graph $G$ is undirected, $\boldsymbol{A}$ is symmetric and we assume also the nested matrices $\boldsymbol{N}$ to be symmetric.

### 4.1 Computational Complexity

To analyse the computational complexity of Problem 4.1, we will start by analysing its parts. In the first problem we are given a binary matrix $\boldsymbol{A}$, and our task is to find the nested binary matrix $\boldsymbol{N}$ that is as close to $\boldsymbol{A}$ as possible (i.e. Problem 4.1 with $k = 1$). No polynomial-time algorithm for this problem is known, but the problem is also not known to be NP-hard [6, Ch. 4.4]. On the other hand, the problem of finding the largest nested submatrix of $\boldsymbol{A}$ is NP-hard [6, Thm. 4.13], where the size of the submatrix is counted as the total number of its rows and columns.

While generating the optimal matrices $\boldsymbol{N}_i$ is hard, we can generate *some* nested matrices, and choose from them. Unfortunately, a nested matrix is an example of *generalized rank-1 matrix* [14], and hence, given a matrix $\boldsymbol{A}$ and a collection $\mathcal{N} = \{\boldsymbol{N}_i\}_{i=1}^{n}$, it is NP-hard to choose the smallest subcollection $\mathcal{S} \subseteq \mathcal{N}$ such that $\boldsymbol{A} = \bigcup_{\boldsymbol{N} \in \mathcal{S}} \boldsymbol{N}$ (assuming such collection exists). It is also NP-hard to choose the $k$ matrices from $\mathcal{N}$ that minimize the distance between $\boldsymbol{A}$ and $\bigcup_{i=1}^{k} \boldsymbol{N}_i$ [14]. Even approximating the error to within a superpolylogarithmic factor is NP-hard [14].

Conclusively establishing the computational complexity of Problem 4.1 remains intriguing future work. As many of its subproblems are NP-hard, it seems reasonable to expect it to be NP-hard.

## 4.2 Matrix Factorization Formulation

Given Proposition 2.5, it is tempting to think that Problem 4.1 can be solved by finding a symmetric nonnegative rounding rank-$k$ decomposition of $\boldsymbol{A}$, i.e. an $n$-by-$k$ nonnegative matrix $\boldsymbol{B}$ such that $\tau(\boldsymbol{B}\boldsymbol{B}^T) \approx \boldsymbol{A}$. Unfortunately, this *does not* solve the problem. The nonnegative rounding rank-$k$ decomposition is not equal to the union of $k$ nested matrices, as the following example illustrates.

*Example 4.2.* Consider the rank-1 matrices $\boldsymbol{A}$ and $\boldsymbol{B}$:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1/\sqrt{3} & 1/3 \\ 1/\sqrt{3} & 1/3 & 1/\sqrt{27} \\ 1/3 & 1/\sqrt{27} & 1/9 \end{pmatrix} \quad \boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1/9 & 1/3 \\ 0 & 1/3 & 1 \end{pmatrix} .$$

$\tau(\boldsymbol{A}) = \left(\begin{smallmatrix} 1&1&0 \\ 1&0&0 \\ 1&0&0 \end{smallmatrix}\right)$ and $\tau(\boldsymbol{B}) = \left(\begin{smallmatrix} 0&0&0 \\ 0&0&0 \\ 0&0&1 \end{smallmatrix}\right)$ are nested. But for $\boldsymbol{A} + \boldsymbol{B}$ the resulting matrix is not the union $\tau(\boldsymbol{A}) \cup \tau(\boldsymbol{B})$ as $\tau(\boldsymbol{A} + \boldsymbol{B})$ has 1s in the lower-right corner that are not present in either $\tau(\boldsymbol{A})$ or $\tau(\boldsymbol{B})$:

$$\boldsymbol{A} + \boldsymbol{B} = \begin{pmatrix} 1 & 1/\sqrt{3} & 1/3 \\ 1/\sqrt{3} & 4/9 & \frac{3+\sqrt{3}}{9} \\ 1/3 & \frac{3+\sqrt{3}}{9} & 10/9 \end{pmatrix}, \tau(\boldsymbol{A} + \boldsymbol{B}) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} .$$

With the tropical algebra we can preserve the original nested matrices. To see that, we will make use of the following lemma.

Lemma 4.3. *Let $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a monotonically increasing function. Then $f$ and $\boxplus$* distribute, *that is*

$$f(a \boxplus b) = f(a) \boxplus f(b) \quad \text{for all } a, b \in \mathbb{R}_{\geq 0}. \tag{4}$$

Proof. Without loss of generality, let $a > b$. Then $f(a \boxplus b) = f(a) = f(a) \boxplus f(b)$, following from the monotonicity of $f$. □

Elementwise exponentiation of a rank-1 tropical matrix produces a classic rank-1 matrix. In fact, the exponential map establishes an isomorphic relation between the set of rank-1 tropical matrices and the nonnegative subset of classical rank-1 matrices: for any nested matrix $\boldsymbol{N}$, such that $\boldsymbol{N}_{ij} = \tau_{1/2}(\boldsymbol{a}_i \boldsymbol{b}_j)$ for some vectors $\boldsymbol{a} \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{b} \in \mathbb{R}^{1 \times m}$, there exists a tropical rank-1 matrix $\boldsymbol{T}$ such that $\boldsymbol{N}_{ij} = \tau_0(\exp(\boldsymbol{T}_{ij}))$. This can be verified by setting $\boldsymbol{T}_{ij} = \boldsymbol{x}_i + \boldsymbol{y}_j$, where $\boldsymbol{x}_i = \log(\boldsymbol{b}_i)$ and $\boldsymbol{y}_j = \log(\boldsymbol{c}_j)$. The expression of nested matrices via the tropical algebra can be simplified even further:

Proposition 4.4. *Let $\boldsymbol{A} \in \{0, 1\}^{n \times m}$ be an arbitrary binary matrix. $\boldsymbol{A}$ is nested if and only if there exist vectors $\boldsymbol{x} \in \overline{\mathbb{R}}^{n \times 1}$ and $\boldsymbol{y} \in \overline{\mathbb{R}}^{1 \times m}$ such that $\boldsymbol{A}_{ij} = \tau(\boldsymbol{x}_i + \boldsymbol{y}_j)$.*

Proof. From Proposition 2.5 we know that $\boldsymbol{A}_{ij} = \tau(\boldsymbol{a}_i \boldsymbol{b}_j)$ for some vectors $\boldsymbol{a} \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{b} \in \mathbb{R}^{1 \times m}$. Define $\boldsymbol{x}_i = \log(\boldsymbol{a}_i) + \log(2)/2$ and $\boldsymbol{y}_j = \log(\boldsymbol{b}_j) + \log(2)/2$. We have

$$\begin{aligned} \tau(\boldsymbol{x}_i + \boldsymbol{y}_j) &= \tau(\log(\boldsymbol{a}_i) + \log(\boldsymbol{b}_j) + \log(2)) \\ &= \tau(\log(\boldsymbol{a}_i \boldsymbol{b}_j) - \log(2)) \end{aligned} \tag{5}$$

By exponentiating (5) and monotonicity of the exponential function, we conclude $\tau(\boldsymbol{x}_i + \boldsymbol{y}_j) = 1$ if and only if $2\exp(\boldsymbol{x}_i)\exp(\boldsymbol{y}_j) \geq 1$. □

The nested matrix preservation property follows from Lemma 4.3.

Corollary 4.5. *Let $\boldsymbol{N}_i = \tau(\boldsymbol{a}_i + \boldsymbol{b}_i)$, $i = 1, \ldots, k$, be a set of nested binary matrices, where $\boldsymbol{a}_i \in \overline{\mathbb{R}}^{n \times 1}$ and $\boldsymbol{b}_i \in \overline{\mathbb{R}}^{1 \times m}$ for all $i$. Let $\boldsymbol{A}$ be the $n$-by-$k$ matrix with the vectors $\boldsymbol{a}_i$ as its columns, and let $\boldsymbol{B}$ be the $m$-by-$k$ matrix with $\boldsymbol{b}_i$ as its columns. Then the rounded tropical matrix product is the union of the nested matrices:*

$$\tau(\boldsymbol{A} \boxtimes \boldsymbol{B}^T) = \bigcup_{i=1}^{k} \boldsymbol{N}_i . \tag{6}$$

Proof. We have

$$\begin{aligned} \tau(\boldsymbol{A} \boxtimes \boldsymbol{B}^T) &= \tau((\boldsymbol{a}_1 + \boldsymbol{b}_1) \boxplus (\boldsymbol{a}_2 + \boldsymbol{b}_2) \boxplus \cdots \boxplus (\boldsymbol{a}_k + \boldsymbol{b}_k)) \\ &= \tau(\boldsymbol{a}_1 + \boldsymbol{b}_1) \boxplus \tau(\boldsymbol{a}_2 + \boldsymbol{b}_2) \boxplus \cdots \boxplus \tau(\boldsymbol{a}_k + \boldsymbol{b}_k) \\ &= \tau(\boldsymbol{a}_1 + \boldsymbol{b}_1) \cup \tau(\boldsymbol{a}_2 + \boldsymbol{b}_2) \cup \cdots \cup \tau(\boldsymbol{a}_k + \boldsymbol{b}_k) , \end{aligned}$$

where we used (4) in the second equality, and the fact that if $a, b \in \{0, 1\}$ then $a \boxplus b = a \vee b$ in the last equality. □

Hence we can re-write Problem 4.1 as follows.

*Problem 4.6 (Rounded tropical factorization).* Given an $n$-by-$m$ binary matrix $\boldsymbol{A}$ and an integer $k$, find matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n \times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k \times m}$ that minimize

$$\sum_{i} \sum_{j : j \neq i} \left| \boldsymbol{A}_{ij} - \tau(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} \right| . \tag{7}$$

For a symmetric decomposition we have $\boldsymbol{A} \approx \tau(\boldsymbol{B} \boxtimes \boldsymbol{B}^T)$.

## 4.3 Maximizing the Likelihood of the Data

Problem 4.6 has two issues: it is NP-hard to optimize, and, as (7) measures the binary reconstruction error, it yields no *confidence* how certain a particular entry should be 1 or 0. Therefore, we replace the threshold function $\tau$ with the *logistic* (or *sigmoid*) *function*

$$\sigma_t(x) = \left(1 + \exp(-tx)\right)^{-1} . \tag{8}$$

We will omit the subscripts when they are obvious and we will write $\sigma_t(\boldsymbol{A}) = \boldsymbol{B}$ for the matrix $\boldsymbol{B}$ that has $\boldsymbol{B}_{ij} = \sigma_t(\boldsymbol{A}_{ij})$.

Using the sigmoid function, we model the input matrix as a multivariate Bernoulli random variable with its odds given by $\sigma_t(\boldsymbol{B} \boxtimes \boldsymbol{C})$ (cf. [17]). As the sigmoid function is monotonically increasing, the distributivity lemma (Lemma 4.3) holds. Thus, our goal becomes to maximize the likelihood of observing the data. This is equivalent to minimizing the negative log-likelihood of $\boldsymbol{A}$,

$$\begin{aligned} E(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},t) = &-\sum_i \sum_{j\neq i} \boldsymbol{A}_{ij} \log(\sigma_t(\boldsymbol{B}\boxtimes\boldsymbol{C})_{ij}) \\ &-\sum_i \sum_{j\neq i} (1-\boldsymbol{A}_{ij}) \log(1-\sigma_t(\boldsymbol{B}\boxtimes\boldsymbol{C})_{ij})\ , \end{aligned} \tag{9}$$

We can now formulate the maximum likelihood problem:

*Problem 4.7 (Logistic-tropical factorization).* Given an $n$-by-$m$ binary matrix $\boldsymbol{A}$, an integer $k$, and $t > 0$, find matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$ that minimize (9).

For a symmetric decomposition, the objective (9) changes to

$$\begin{aligned} E(\boldsymbol{A},\boldsymbol{B},t) = &-2\sum_i \sum_{j> i} \boldsymbol{A}_{ij} \log(\sigma_t(\boldsymbol{B}\boxtimes\boldsymbol{B}^T)_{ij}) \\ &-2\sum_i \sum_{j> i} (1-\boldsymbol{A}_{ij}) \log(1-\sigma_t(\boldsymbol{B}\boxtimes\boldsymbol{B}^T)_{ij})\ , \end{aligned} \tag{10}$$

which yields the problem we use in the remainder of this work:

*Problem 4.8 (Symmetric logistic-tropical factorization).* Given an $n$-by-$n$ binary matrix $\boldsymbol{A}$, an integer $k$, and $t > 0$, find a matrix $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ minimizing (10).

## 5 ALGORITHM

Real-world networks tend to be very sparse, and hence we need an algorithm that can solve Problem 4.8 without having to all $O(n^2)$ potential edges. To that end, we use stochastic gradient descent (SGD) with subsampling of the zeros. SGD is a common approach for decomposing sparse matrices, but usually – as in collaborative filtering – the zero entries are assumed to be unobserved. This is not the case here, as the zeros also carry information. Hence, we use subsampling.

The algorithm, called SLTF (Symmetric Logistic-Tropical Factorization) and presented in Algorithm 1, runs multiple epochs. On every epoch, we sample $O(|\boldsymbol{A}|)$ elements, where $|\boldsymbol{A}|$ is the number of non-zero elements in the matrix, and update the corresponding rows of the factor matrix. It takes three main parameters: $k$, the rank of the decomposition; $t$, the steepness of the sigmoid function; and $\mu$ that controls the behaviour of the soft max function (see Section 5.1).

The factor matrix $\boldsymbol{B}$ is initialized with random numbers from $[-0.1, 0]$ (Line 2). This guarantees that the initial solution is sparse. We use tiered sampling to sample the elements we update (Lines 6 and 7) so that we can ensure that we sample enough 1s even from the sparsest matrices. We sample the locations $\boldsymbol{A}_{ij} = 1$ uniformly, but to sample the locations of zeros, we use weighted sampling. We sample a location $\boldsymbol{A}_{ij} = 0$ with a probability that is proportional to the number of 1s in rows $i$ and $j$ of $\boldsymbol{A}$.

We use separate step sizes for the elements that are 1 and that are 0 (*s1* and *s0*, respectively). These are updated (Line 11) using a bold driver heuristic: we increase them if the new error is smaller than the previous one, otherwise we decrease them. In addition, if the current error is greater on 1s than on 0s, then we increase the step size for 1s relative to that for 0s, and vice versa. The actual gradient updates are explained in detail below.

**Algorithm 1** SLTF

```
Input: $A \in \{0,1\}^{n\times m}$, $k \in \mathbb{N}$, $t \in \mathbb{R}_{>0}$, $\mu \in \mathbb{R}_{>0}$
Output: $B \in \overline{\mathbb{R}}^{n\times k}$
1:  function SLTF($A$, $k$, $t$, $\mu$)
2:      Initialize $B$
3:      $LL \leftarrow E(A, B, t)$
4:      Initialize $s1$, $s0$
5:      while not converged do
6:          $idx_1 \leftarrow$ sample $O(|A|)$ 1s uniformly at random
7:          $idx_0 \leftarrow$ sample $O(|A|)$ 0s weighted by the degree
8:          $B \leftarrow$ UpdateFactors($A$, $B$, $k$, $t$, $s1$, $s0$, $\mu$, $idx_1$, $idx_0$)
9:          $LL_{\text{old}} \leftarrow LL$
10:         $LL \leftarrow E(A, B, t)$
11:         [$s1$, $s0$] $\leftarrow$ UpdateStepSizes($s1$, $s0$, $LL$, $LL_{\text{old}}$)
12:     return $B$ that had the smallest negative log-likelihood
```

Computing the objective function (Line 10) has complexity $O(n^2k)$ for $n$-by-$k$ matrix $\boldsymbol{B}$. Instead of computing it completely, we approximate it on a sample of size $O(|\boldsymbol{A}|)$, again using tiered sampling.

### 5.1 UpdateFactors

The function UpdateFactors (Line 8) follows the SGD approach and updates the factor matrix $\boldsymbol{B}$ given a sequence of sampled data points by optimizing the objective locally. We will explain UpdateFactors using an asymmetric notation $\boldsymbol{A} \approx \boldsymbol{BC}$ as this simplifies the discussion. The symmetric variant will be explained at the end of this section.

In the standard matrix factorization setting the objective of SGD, $\|\boldsymbol{A}-\boldsymbol{BC}\|_F^2$, is represented as a sum of functions that each depend only on one row of $\boldsymbol{B}$ and one column of $\boldsymbol{C}$:

$$\|\boldsymbol{A}-\boldsymbol{BC}\|_F^2 = \sum_{ij}(\boldsymbol{A}_{ij} - (\boldsymbol{BC})_{ij})^2 = \sum_{ij}\varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j)\ . \tag{11}$$

Then, given a sequence of index pairs $\{i(\alpha), j(\alpha)\}_{\alpha=1}^{l}$ that correspond to the elements of $\boldsymbol{A}$, SGD updates the individual rows of $\boldsymbol{B}$ and columns of $\boldsymbol{C}$ by each time taking a single step in the direction of the steepest descent of $\varphi_{i(\alpha)j(\alpha)}(\boldsymbol{B}_{i(\alpha)}, \boldsymbol{C}^{j(\alpha)})$.

We adapt the SGD approach to the tropical-logistic factorization problem. First observe that (9) is additive and can be represented in a form identical to (11) by setting

$$\begin{aligned} \varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t) = &-\big(\boldsymbol{A}_{ij}\log(\sigma_t(\boldsymbol{B}_i \boxtimes \boldsymbol{C}^j)) \\ &+ (1-\boldsymbol{A}_{ij})\log(1-\sigma_t(\boldsymbol{B}_i \boxtimes \boldsymbol{C}^j))\big)\ . \end{aligned} \tag{12}$$

Unfortunately, $\varphi_{ij}$ are not differentiable, as they contain the max operator. In order to differentiate $\varphi_{ij}$, we replace the maximum with the *soft max* function,

$$\max_{s=1..k}\{x_s\} \approx \sum_{s=1}^{k} \frac{\mathrm{e}^{\mu x_s}}{\sum_{j=1}^{k}\mathrm{e}^{\mu x_j}} x_s = \sum_{s=1}^{k} f(x_s, x, \mu) x_s\ , \tag{13}$$

where $\mu$ is a relaxation parameter.

Using the soft max in our original objective, we obtain

$$\begin{aligned} \varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t) \approx &-\big(\boldsymbol{A}_{ij}\log(\sigma_t(\boldsymbol{B}_i \boxtimes_\mu \boldsymbol{C}^j)) \\ &+ (1-\boldsymbol{A}_{ij})\log(1-\sigma_t(\boldsymbol{B}_i \boxtimes_\mu \boldsymbol{C}^j))\big) \\ = &\ \tilde{\varphi}_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t, \mu)\ , \end{aligned} \tag{14}$$

where $\boxtimes_\mu$ denotes the tropical matrix product with the max operation relaxed using parameter $\mu$. For any matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$, we have $\boldsymbol{B} \boxtimes_\mu \boldsymbol{C} \to \boldsymbol{B} \boxtimes \boldsymbol{C}$ when $\mu \to \infty$. We can now use the gradient of the right side of (14) as a "relaxation" of the gradient of $\varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t)$. Note that since the functions $\tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu)$ depend only on elementwise sums of the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$, we have $\nabla_{\boldsymbol{b}} \tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu) = \nabla_{\boldsymbol{c}} \tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu)$. If we denote $\boldsymbol{x} = \boldsymbol{b} + \boldsymbol{c}$, $a = \boldsymbol{A}_{ij}$, and $\max_\mu(x) = \sum_{s=1}^{k} f(x_s, x, \mu) x_s$, then the relaxed value of $\frac{\partial}{\partial \boldsymbol{b}_l} \tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu)$ is given by

$$-t\left[a(1-\sigma_t(\boldsymbol{b}\boxtimes_\mu \boldsymbol{c})) + (1-a)(1-\sigma_t(-\boldsymbol{b}\boxtimes_\mu \boldsymbol{c}))\right] \frac{\partial}{\partial \boldsymbol{b}_l} \boldsymbol{b} \boxtimes_\mu \boldsymbol{c}\ ,$$

$$\text{where} \quad \frac{\partial}{\partial \boldsymbol{b}_l} \boldsymbol{b} \boxtimes_\mu \boldsymbol{c} = f(x_l, x, \mu)\left[\mu(x_l - \max_\mu(x)) + 1\right] . \qquad (15)$$

Finally, to adapt (15) to our symmetric objective, we need to set $\boldsymbol{b} = \boldsymbol{B}_i$ and $\boldsymbol{c} = \boldsymbol{B}_j^T$ and recall that the gradients with respect to $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$ are the same.

## 5.2 Time Complexity of SLTF

The major contributors to the complexity of SLTF are the computation within UpdateFactors and computing the likelihood of the data given the current factor matrix. On every iteration of SLTF, UpdateFactors performs updates to $\boldsymbol{B}$ for each of the $O(|\boldsymbol{A}|)$ sampled data points. Each time it has to compute the gradient of $\tilde{\varphi}_{ij}(\boldsymbol{B}_i, \boldsymbol{B}_j^T, t, \mu)$ and then update $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$. Both of these procedures take time $O(k)$, and hence the complexity of UpdateFactors is $O(|\boldsymbol{A}|\, k)$. Since we use a fixed number of samplings for estimating the likelihood, and an evaluation of the likelihood at a single point takes $O(k)$ time, the complexity of approximating the objective is $O(|\boldsymbol{A}|\, k)$. Other parts of SLTF are not as computationally expensive – initializing $\boldsymbol{B}$ takes time $O(nk)$ and the complexity of SampleData is $O(|\boldsymbol{A}|)$. If $M$ is the total number of cycles performed by SLTF, then its total complexity is $O(Mk(n + |\boldsymbol{A}|)$. Due to the complex objective, proofs of speed of convergence seem hard to obtain.

One of the benefits of SGD is that it allows us to parallelize the algorithm: we can use a partitioned approach, an asynchronous distributed approach, or other parallelization methods for SGD (see e.g. [18] and references therein). For our implementation, we use shared-memory parallelization over the different elements we update, and SIMD vectorization over the individual gradient computations.

## 6 EXPERIMENTAL EVALUATION

In this section we experimentally test SLTF and compare it to other methods. We first describe the competing algorithms, then report the performance for synthetically generated data, and finish with the results on various real-world data.

SLTF is implemented in Matlab and C and uses OpenMP for parallel processing. The source code for SLTF, the scripts to generate synthetic data and to execute the experiments, together with the parameters used in all experiments, are freely available.[2] For all of these experiments, we ran SLTF for 600 iterations without any early stopping criteria.

[2]http://people.mpi-inf.mpg.de/~pmiettin/tropical/logistic/

## 6.1 Methods and metrics

To evaluate SLTF, we compare it against existing approaches. These can be divided into three classes: Basic matrix factorization methods – NMF and Asso– aim for finding a good decomposition of the input using real values, nonnegative real values, or Boolean algebra, respectively. As NMF and Asso both are asymmetric decompositions, we also experimented with a symmetric version of NMF, $\text{NMF}_{\text{sym}}$, that has format $\boldsymbol{W}\boldsymbol{W}^T$ or $\boldsymbol{H}^T\boldsymbol{H}$, depending on which one gives less error. We also tried to use the factors found by NMF in a logistic tropical factorization, i.e. $\sigma(\boldsymbol{W}^1\boldsymbol{H}_1) \boxplus \sigma(\boldsymbol{W}^2\boldsymbol{H}_2) \boxplus \cdots \boxplus \sigma(\boldsymbol{W}^k\boldsymbol{H}_k)$. We call this version $\text{NMF}_{\boxplus}$. It solves Problem 4.7, but since NMF does not try to optimize for this, it is not expected to perform as well as SLTF.

The second group involves methods that directly find thresholded decompositions. The main methods here are LPCA and FastStep (see Section 3). We use the approximate version of FastStep with at least 10 internal iterations, as suggested.

The third group contains just one method, HyCoM-FIT [2]. It fits power-law models to non-overlapping communities. Since HyCoM-FIT does not decompose the data in the matrix factorization sense, we obtain its reconstruction matrix by joining the top $k$ largest communities predicted by its model.

We measure the quality using two metrics: negative log-likelihood (10) and relative binary reconstruction error. Assuming $\tilde{\boldsymbol{A}}$ is the approximation a method gave for input matrix $\boldsymbol{A}$, the relative binary error is $\|\boldsymbol{A} - \tau_\alpha(\tilde{\boldsymbol{A}})\|_F^2 / \|\boldsymbol{A}\|_F^2$. Here, $\alpha$ is selected to be correct with the method (usually $\alpha = 1/2$) and we set the diagonal of $\boldsymbol{A}$ and $\tilde{\boldsymbol{A}}$ to all-zeros to ignore self-loops. Notice that $\|\boldsymbol{A}\|_F^2$ simply calculates the number of non-zeros in $\boldsymbol{A}$, as $\boldsymbol{A}$ is binary. Also note that every column $\boldsymbol{b}$ of the factor matrix $\boldsymbol{B}$ yields a nested matrix, which can be seen by applying the thresholding $\tau(\sigma(\boldsymbol{b} + \boldsymbol{b}^T))$.

Most methods do not optimize for our likelihood model, and hence the binary error is more fair measure for them. All methods are run 10 times on the synthetic data and 3 times on the real data to account to the random variance and select a good result. After the required number of runs, we return the best result.

## 6.2 Synthetic data

In the synthetic experiments we test whether SLTF can find the logistic tropical structure when it is present in the data. To generate the data, we first create the factor matrix $\boldsymbol{B} \in \mathbb{R}^{n\times k}$, and compute the thresholded product $\boldsymbol{A} = \tau(\sigma_t(\boldsymbol{B} \boxtimes \boldsymbol{B}^T))$.

Unless specified otherwise, we generated matrices of size $1000 \times 1000$ with matrix $\boldsymbol{B}$ having dimensions $1000 \times 10$ (i.e. rank 10). By default, the input matrix density is set to 3 %, and the levels of both additive and destructive noise to are 5 %. In every experimental setup we vary one of the above parameters, while keeping the rest fixed. The number of nonzeros in the synthetic data sets ranges from less than 1000 to roughly 120 000.

All algorithms were run 5 times on each matrix, with the best result being selected. We do not report the results for $\text{LPCA}_{\text{sym}}$ or $\text{NMF}_{\boxplus}$, as their original versions rely heavily on the ability to use asymmetric factors, and hence their errors were orders of magnitude worse than those of other methods. For the negative log-likelihood, we also show the likelihood the original factors would give.

We will first present the experimental setups before discussing the results in detail.

**Varying additive noise.** The level of additive noise (i.e. replacing 0s with 1s) is defined with respect to the number of 1s in the data. We varied the noise from 0 % to 55 % with increments of 5 %. The results are in Figures 1a and 1g.

**Varying additive noise with 12 % density.** This setup only differs from the one above in that the factor matrix density is 12 %. The results are in Figures 1c and 1i.

**Varying destructive noise.** The level of destructive noise (turning 1s to 0s) rises from 0 % to 55 % with increments of 5 %. The results are shown in Figures 1b and 1h, and the results of the dense version in Figures 1d and 1j.

**Varying density.** We varied the density of the input matrix $\boldsymbol{A}$ from 1 % to 12 % with increments of 1 % and report the errors in Figures 1e and 1k.

**Varying rank.** In this setup we investigate how the algorithms respond to varying the (tropical logistic) rank of the data. We varied the number of columns of the factor matrix $\boldsymbol{B}$ from 2 to 12 with increments of 2. The results are shown in Figures 1f and 1l.

**Scalability.** Here we test how well SLTF scales with respect to the number of non-zeros (edges), rows (nodes), rank, and computing cores. The results are in Figure 2. These experiments were run on an Intel Xeon E5-2667 CPU with 16 cores at 3.2 GHz. When not varied, these matrices had $2^{13}$ rows and columns and $2^{13}$ non-zeros. We used rank 50 and sixteen cores.

**Discussion.** SLTF and LPCA give the best overall results. For the most part they are quite close, with SLTF producing somewhat better results on the varying density test and LPCA being slightly better with additive noise. The close results of SLTF and LPCA are not surprising because we tested them on SLTF's native data, while LPCA has more degrees of freedom. It is worth noting that both SLTF and LPCA produce likelihoods that are better than those of original factors that were used to generate the data. While this might seem strange, it is easily explained by the fact that the original factors were merely used for generating binary matrices and not optimized for the likelihood. In contrast to SLTF and LPCA, the NMF-based methods do not show good results with either likelihood or binary error. FastStep gives quite good binary reconstruction errors, although it is on average inferior to SLTF and LPCA. Asso, on the other hand, has consistently high reconstruction errors, indicating that the structure of the data is very different from what it expects.

SLTF is a scalable algorithm – its running time grows moderately relative to the number of non-zeros; indeed, we can increase the number of non-zeros by 32-fold (from $2^{16}$ to $2^{20}$) while only doubling the running time (from 4 to 8 seconds). The behaviour with respect to the number of rows and factorization rank is also good, as expected by the runtime analysis. The algorithm also shows good speedups with increasing numbers of cores.

The synthetic experiments confirm that SLTF behaves as expected: high noise levels do have an effect, but otherwise it is quite robust against the characteristics of the data.

## 6.3 Real-world data

The real-world experiments are conducted to validate that our findings on the synthetic experiments correlate with the real world.

**Table 1: Properties of small real-world data sets. All matrices are symmetric and $k$ denotes the rank used for reconstruction.**

| | $\text{Mam}_N$ | $\text{Mam}_E$ | Jazz | Paleo | 4News | Christ. |
|---|---|---|---|---|---|---|
| # rows | 3203 | 194 | 198 | 139 | 800 | 1736 |
| # nonzeros | 864083 | 21844 | 5484 | 8995 | 263400 | 15010 |
| density | 0.084 | 0.580 | 0.139 | 0.465 | 0.411 | 0.005 |
| $k$ | 40 | 10 | 5 | 6 | 10 | 30 |

We will also use the real-world data to study what kind of rank-1 matrices SLTF finds. It is important to note that, as with many other matrix factorization methods, there is no general way of determining the best rank for SLTF. The user might apply their prior knowledge about the nature of the data, but in most cases some empirical search for the best rank will be needed.

**Data sets.** We used two sets of real-world data sets: smaller sets where we could compare all different methods, and larger sets, where only SLTF was able to run.

The smaller data sets are $\text{Mam}_N$, $\text{Mam}_E$, Jazz, Paleo, 4News, and Christ.. Their properties are listed in Table 1, and further information is given in our online appendix.[3]

The larger data sets come from the SNAP dataset collection.[4] We preprocessed the datasets by removing nodes that were not part of communities of size at least 100. That left us with: YouTube, a graph of 20 329 nodes, 133 communities, and a density of 0.061 %; DBLP, a graph of 212 637 nodes, 954 communities, and a density of 0.004 %; Amazon, a graph of 299 902 nodes, 1675 communities, and a density of 0.002 %; and LiveJournal, a graph of 775 003 nodes, 9314 communities, and a density of 0.003 %.

**Numerical results.** We report the binary errors for all methods in Table 2; log-likelihoods are available in the online appendix.

With the small real-world data sets, LPCA is the best, obtaining often perfect reconstructions (behaviour also observed in [15]). One should note, however, that LPCA is an asymmetric decomposition and clearly the slowest of the methods presented here. The symmetric version of LPCA, $\text{LPCA}_{\text{sym}}$, is among the worst methods. FastStep finds decompositions with reconstruction error comparable to or better than SLTF. It does this with a significantly slower running time, though. Hence, we also tested a version with the minimum number of iterations set to 5 instead of 10, called $\text{FastStep}_5$. This improved the running time to be comparable to SLTF, but with a significant cost in quality. Finally, while some NMF results are of comparable quality, one should remember that it is not symmetric.

As only LPCA and FastStep were really comparable to our results, we tried to use them with the larger real-world data sets. For LPCA, this was clearly undoable, while $\text{FastStep}_5$ managed to run YouTube in a bit over 9 h, compared to the 22 min SLTF took, and it could not finish DBLP in a week. Hence, we will only report results from SLTF: Table 3 gives the negative log-likelihoods *relative* to the data size for the different data sets, together with the rank and the time SLTF took on a 16-core server.

As can be seen from Table 3, YouTube gives clearly the best result. This is probably a combination of the data being more amenable to

[3]http://people.mpi-inf.mpg.de/~pmiettin/tropical/logistic/
[4]http://snap.stanford.edu/data/

**Table 2: Binary error for small real-world data sets**

| | $\mathsf{Mam}_N$ | $\mathsf{Mam}_E$ | Jazz | Paleo | 4News | Christ. |
|---|---|---|---|---|---|---|
| SLTF | 0.184 | 0.033 | 0.508 | 0.110 | 0.264 | 0.567 |
| LPCA | 0.000 | 0.000 | 0.305 | 0.000 | 0.222 | 0.000 |
| $\mathtt{LPCA}_{\mathtt{sym}}$ | 8.682 | 0.624 | 3.891 | 0.394 | 0.850 | 203.900 |
| NMF | 0.189 | 0.158 | 0.481 | 0.169 | 0.325 | 0.503 |
| $\mathtt{NMF}_{\mathtt{sym}}$ | 10.910 | 0.729 | 5.531 | 1.136 | 1.434 | 201.600 |
| $\mathtt{NMF}_{\boxplus}$ | 0.477 | 0.544 | 0.577 | 0.320 | 0.525 | 0.539 |
| Asso | 0.336 | 0.202 | 0.562 | 0.228 | 0.423 | 0.542 |
| HyCoM-FIT | 0.869 | 0.842 | 0.829 | 0.968 | 0.955 | 2.128 |
| FastStep | 0.024 | 0.021 | 0.405 | 0.120 | 0.236 | 0.236 |
| $\mathtt{FastStep}_5$ | 0.024 | 0.444 | 1.750 | 0.120 | 1.109 | 1.109 |

**Table 3: Rank, relative negative log-likelihood, and time for large real-world data sets and SLTF**

| | YouTube | DBLP | Amazon | LiveJournal |
|---|---|---|---|---|
| rank | 133 | 954 | 1675 | 5000 |
| $\log L/n^2$ | 0.0091 | 0.0283 | 0.0277 | 0.0440 |
| time | 22 min | 263 min | 441 min | 36 h |

the model and SLTF having a better initial solution or parameter configuration for the data (all results are best-of-3 restarts, but we tuned the parameters for the YouTube data). Nonetheless, SLTF was relatively fast with all data sets, including the large ones; for example, on LiveJournal, storing the $775003 \times 5000$ factor matrix in 64-bit floating point numbers takes approximately 31 GB.

**Example results.** To understand the kind of decompositions SLTF finds, and to evaluate our assumption that the datasets have nested communities, we looked at the factors in the real-world datasets. Example rank-1 matrices are shown in Figure 3. We see that the communities are nested, and that the area under the blue curve, which is where our factorization gives likelihoods above $1/2$, is also dense, while the area above it is much sparser. The shape of these communities also varies. The leftmost community (from Christ.) has near star-like structure, in the next (from Jazz) the line is almost diagonal. The third community (from $\mathsf{Mam}_N$) shows a concave line, while the last one (also from $\mathsf{Mam}_N$) has a more convex structure. Notice that all these submatrices have essentially full rank, and are hence very hard to describe using standard algebra.

These different communities provide an empirical verification for three of our hypotheses: First, the communities in the different real-world graphs are indeed nested. Second, the nested structure is not only 'hyperbolical' (cf. [2, 13]). Third, SLTF can find these non-clique-like structures from different data sets.

## 7 CONCLUSIONS

Covering a graph with nested subgraphs seems like an inherently combinatorial problem. But using the rounding rank characterization of nested matrices together with the tropical algebra, we could express it as a continuous problem. This allows us to use optimization methods, such as stochastic gradient descent, that would not be applicable to the combinatorial domain. It seems plausible that also other types of graph patterns could be modelled in a similar vein, allowing novel algorithms to be developed also for them.

The use of SGD also allows us to utilize the vast literature on parallel and distributed implementations. For this work, we have only studied a shared-memory parallel implementation, but a distributed approach is equally viable. It can also solve the biggest efficiency bottleneck of our approach, namely that the factor matrix $\boldsymbol{B}$ is dense and storing it is memory-intensive for larger matrices.

Sometimes, the goal of identifying the communities is to *compress* the matrix (see, e.g. [10, 11]). Here nested submatrices can also be used – the main problem is then how the nestedness structure should be expressed: using the continuous-valued vectors (as done here), or the step function. Different expressions will lead to different compression, and potentially also to different-looking results. We leave further studies on this for future research.

Being mainly interested in undirected graphs, we designed SLTF for symmetric matrices. An asymmetric version would not be a significant change, though. An asymmetric algorithm could be used on directed or bipartite graphs (e.g. locations-by-species matrices).

## REFERENCES

[1] Marianne Akian, Ravindra Bapat, and Stéphane Gaubert. 2007. Max-Plus Algebra. In *Handbook of Linear Algebra*, Leslie Hogben (Ed.). Chapman & Hall/CRC, Boca Raton.

[2] Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos. 2014. Beyond Blocks: Hyperbolic Community Detection. In *ECML PKDD '14*. 50–65.

[3] Miguel Araujo, Pedro Ribeiro, and Christos Faloutsos. 2016. Faststep: Scalable boolean matrix decomposition. In *PAKDD '16*. 461–473.

[4] Stephen P Borgatti and Martin G Everett. 1999. Models of core/periphery structures. *Soc. Networks* 21 (1999), 375–395.

[5] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2014. Overlapping community detection in labeled graphs. *Data Min. Knowl. Discov.* 28, 5-6 (Sept. 2014), 1586–1610.

[6] Esa Junttila. 2011. *Patterns in permuted binary matrices*. Ph.D. Dissertation. Helsinki University Press, Helsinki.

[7] Esa Junttila and Petteri Kaski. 2013. Segmented nestedness in binary data. In *SDM '11*. 235–246.

[8] Sanjar Karaev and Pauli Miettinen. 2016. Cancer: Another Algorithm for Subtropical Matrix Factorization. In *ECMLPKDD '16*. 576–592.

[9] Sanjar Karaev and Pauli Miettinen. 2016. Capricorn: An Algorithm for Subtropical Matrix Factorization. In *SDM '16*. 702–710.

[10] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. 2014. VoG: Summarizing and Understanding Large Graphs. In *SDM '14*. 91–99.

[11] Yongsub Lim, U Kang, and Christos Faloutsos. 2014. SlashBurn: Graph Compression and Mining beyond Caveman Communities. *IEEE Trans. Knowl. Data Eng.* 26, 12 (2014), 3077–3089.

[12] Heikki Mannila and Evimaria Terzi. 2007. Nestedness and segmented nestedness. In *KDD '07*. 480–489.

[13] Saskia Metzler, Stephan Günnemann, and Pauli Miettinen. 2016. Hyperbolae Are No Hyperbole: Modelling Communities That Are Not Cliques. In *ICDM '16*. 330–339.

[14] Pauli Miettinen. 2015. Generalized Matrix Factorizations as a Unifying Framework for Pattern Set Mining: Complexity Beyond Blocks. In *ECMLPKDD '15*. 36–52.

[15] Stefan Neumann, Rainer Gemulla, and Pauli Miettinen. 2016. What You Will Gain By Rounding: Theory and Algorithms for Rounding Rank. In *ICDM '16*. 380–389.

[16] Bruce D. Patterson and Wirt Atmar. 1986. Nested subsets and the structure of insular mammalian faunas and archipelagos. *Biol. J. Linnean Soc.* 28, 1-2 (May 1986), 65–82.

[17] Andrew I. Schein, Lawrence K. Saul, and Lyle H. Ungar. 2003. A Generalized Linear Model for Principal Component Analysis of Binary Data. In *AISTATS '03*.

[18] Christina Teflioudi, Faraz Makari, and Rainer Gemulla. 2012. Distributed Matrix Completion. In *ICDM '12*. 655–664.

[19] Jason Weston, Ron J Weiss, and Hector Yee. 2013. Nonlinear latent factorization by embedding multiple user interests. In *RecSys '13*. 65–68.

[20] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM '13*. 587–596.

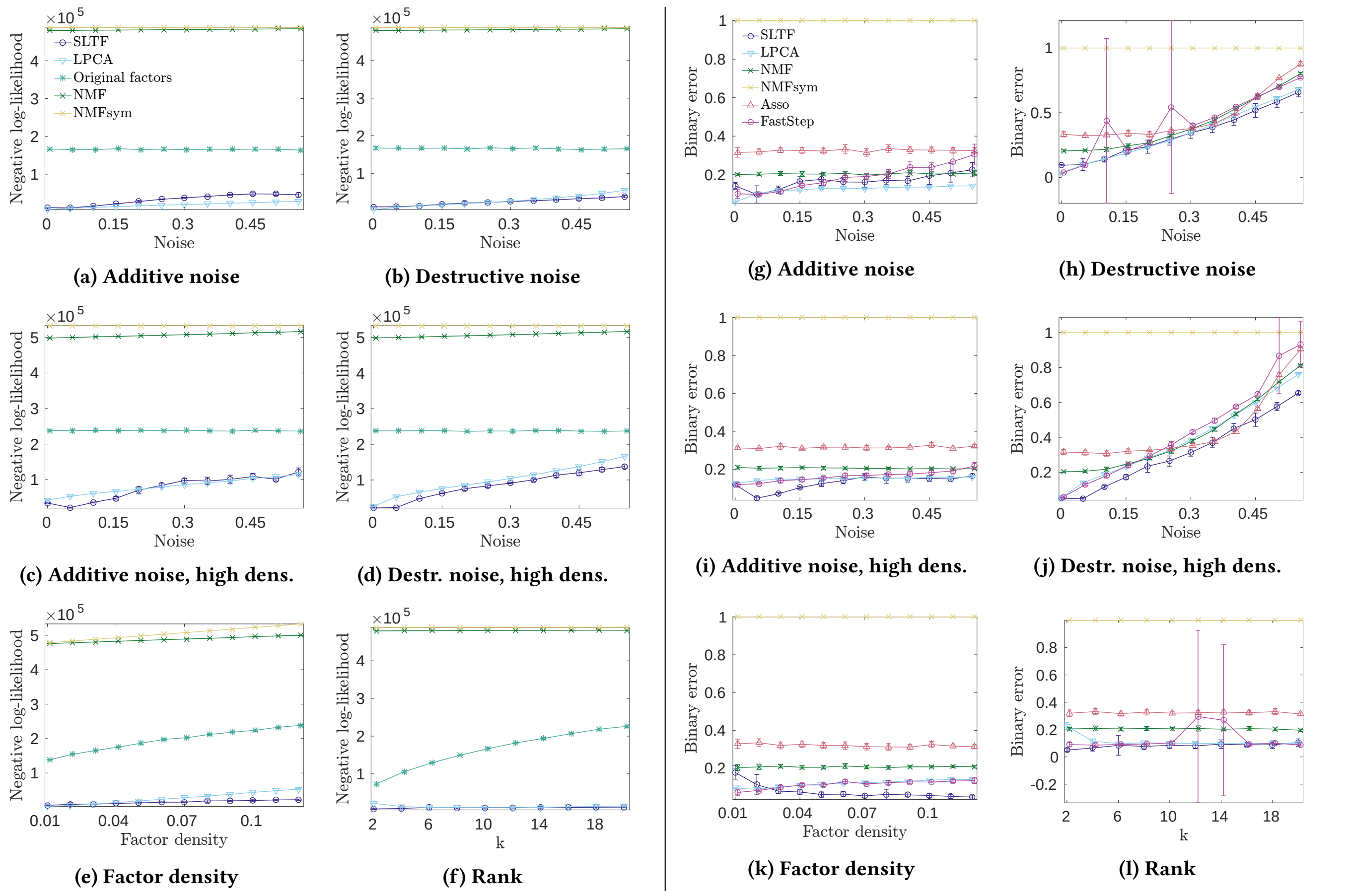

**Figure 1: Negative log-likelihoods (a–f) and binary reconstruction errors (g–l) on synthetic data. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.**

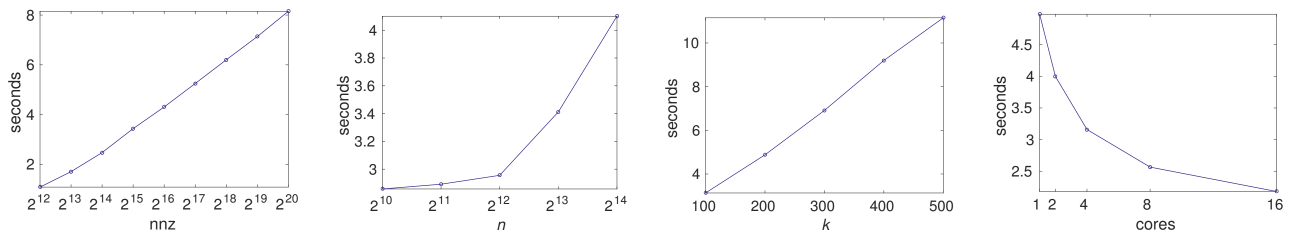

**Figure 2: Scalability of SLTF with respect to (from left to right) number of non-zeros; dimensionality; rank; number of cores. All values are means over five restarts. Notice that the first two plots have logarithmic $x$-scale.**

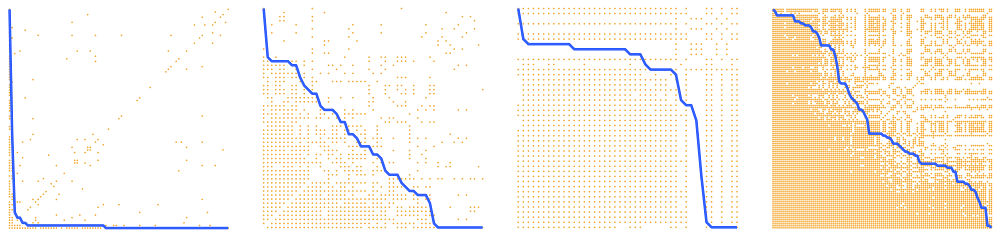

**Figure 3: Example nested factors. Datasets from left: Christ., Jazz, and Mam$_N$ (twice).The orange dots are the 1s in the matrix and the found community is the area left and down from the blue line.**