Capricorn: An Algorithm for Subtropical Matrix Factorization

Sanjar Karaev* Pauli Miettinen*

Abstract Max-times algebra, sometimes known as subtropical algebra, is a semi-ring over the nonnegative real numbers where the addition operation is the max function and the multiplication is the standard one. Factorizing a nonnegative matrix over the max-times algebra, instead of the standard (nonnegative) one, allows us to find structures and regularities that cannot be easily expressed in the standard algebra. In this paper we study the problem of decomposing matrices over the max-times algebra; in particular, we present our algorithm, `Capricorn`, to find such decompositions. Experimental evaluation shows that `Capricorn` finds the max-times structure from data reliably, and furthermore, also identifies areas where it cannot find any structure.

Keywords: tropical algebra; matrix factorization

1 Introduction

Summarising data using a set of patterns, or regularities, found from it is among the primary tasks of data mining. The type of patterns used and how they are combined dictates the kind of insights a data analyst can get from the summarisation. Matrix factorizations are one of the most common techniques for doing the summarisation, expressing the data matrix as a sum of rank-1 matrices, and revealing linear dependencies between the data values; the venerable Singular Value Decomposition (SVD) being perhaps the best-known factorization. But if we want to find another type of structure, say, where every rank-1 matrix contributes something positive to the sum, we should use another kind of decomposition – in this case, the non-negative matrix factorization (NMF). Clearly, the kind of structure NMF finds is not, in general, similar to what SVD finds.

Another, and arguably less common, way of changing the structure we are after is to alter the summation, i.e. how the individual rank-1 factors are combined. In this paper we study an alternative way of combining the rank-1 matrices, namely, by replacing the sum with the max operation. Used over the nonnegative real numbers together with the standard multiplication to generate the rank-1 matrices, this gives us the so-called *max-times* (or *subtropical*) algebra.

While NMF is often interpreted to give the 'parts-of-whole' view of the data, max-times algebra rather gives us 'winner-takes-it-all' view, where every rank-1 matrix can be interpreted to give the *minimum* values for entries in the matrix. Furthermore, using different algebra allows us potentially to achieve better reconstruction error (or lower rank) than NMF (or even SVD), as we show in Section 3.3.

The main contribution of this paper, however, is our algorithm, `Capricorn`, aimed to find low-error fixed-rank max-times decomposition of the matrix (Section 4). `Capricorn` aims to minimize the L_1 error and is tuned to work on discrete-valued matrices. As we show in our experiments (Section 5), `Capricorn` is capable of accurately reconstructing a matrix and estimating unobserved values. Not every matrix has structure that can be well represented using the max-times algebra, though, and `Capricorn` can generally identify these areas and try not to fit anything there.

2 Related Work

Matrix factorization methods are part of the standard toolbox for data analysis, and defining them over non-standard algebras has resulted in a number of important techniques. Arguably the best-known one is the nonnegative matrix factorization (NMF) (see, e.g. [4]), where the factorization is restricted to the nonnegative real numbers.

Another successful example is the Boolean matrix factorization (see, e.g. [11] and references therein), where the factorization is restricted to binary matrices and the algebra is the Boolean one (i.e. the summation is the logical *or*). The max-times algebra is in fact a relaxation of Boolean algebra, since the latter can be seen as its restriction to the set $\{0,1\}$. It should be stressed, however, that our emphasis is on non-binary data sets.

The max-plus algebra [8] has received much more research interest than max-times. Due to their isomorphic nature (see Section 3), many results obtained for max-plus automatically hold for max-times, although this is not directly true in the case of approximate matrix factorizations (Theorem 3.3). Despite the theory of max-plus algebra being relatively young it has been thoroughly studied in recent years. The reason for this is an explosion of interest in so called discrete event systems (DES) [3], where max-plus algebra has become ubiquitously used for modeling (see e.g. [1] and [5]).

Yet another approach of computing the matrix factorization over non-standard algebras involves using the Łukasiewicz algebra. They have been recently applied to decompose matrices with grade values [2]. On the other hand, the max-times algebra has also been used as a part of a recommender system [12].

*Max Planck Institute for Informatics, Saarbrücken, Germany
{skaraev, pmiettin}@mpi-inf.mpg.de

3 Theory

In this section we will introduce the theory of max-times algebra. We will start with the general notation we use throughout the paper, followed by the basic definitions of max-times algebra and decompositions. In Section 3.3 we study the computational complexity of finding good approximate max-times decompositions, what can be said about the sparsity of such decompositions with respect to the sparsity of the original matrix, and the relations between max-times and max-plus algebras in approximate decompositions.

3.1 Notation. Throughout this paper, we will denote a matrix by upper-case boldface letters ($\mathbf{A}$), and vectors by lower-case boldface letters ($\mathbf{a}$). The ith row of matrix $\mathbf{A}$ is denoted by $\mathbf{A}_{i,:}$ and the jth column by $\mathbf{A}_{:,j}$. Most matrices and vectors in this paper are restricted to the nonnegative real numbers $\mathbb{R}_+$. We often endow $\mathbb{R}_+$ with *not-a-number* symbol *NaN*: $\mathbb{R}_+ = [0,\infty) \cup \{NaN\}$. The *NaN* symbol is used to encode values that are either unknown or should be ignored for other reasons; its presence is usually clear from the context, and for the sake of clarity, we will present our theoretical analysis assuming no *NaN*s. Any arithmetic operation involving *NaN* always returns *NaN*, but they are ignored in comparison operations. Empty matrices are denoted by [ ] while for integer n, $[n] = \{1,\dots,n\}$.

Given matrices $\mathbf{A} \in \mathbb{R}_+^{n\times m}$, $\mathbf{B} \in \mathbb{R}_+^{s\times m}$, and $\mathbf{C} \in \mathbb{R}_+^{n\times p}$, we denote by $\left[\begin{smallmatrix}\mathbf{A}\\ \mathbf{B}\end{smallmatrix}\right]$ the $(n+s)$-by-m matrix obtained by vertical concatenation of $\mathbf{A}$ and $\mathbf{B}$, and by $[\mathbf{A},\mathbf{C}]$ the n-by-$(m+p)$ matrix, which is a horizontal concatenation of $\mathbf{A}$ and $\mathbf{C}$.

For any two vectors $\mathbf{v} \in \mathbb{R}_+^k$ and $\mathbf{w} \in \mathbb{R}_+^k$ of equal length, we denote the elementwise division as ./, that is $\mathbf{v}\,./\,\mathbf{w} = (v_1/w_1,\dots,v_k/w_k) \in \mathbb{R}_+^k$.

3.2 Basic definitions. In this paper we consider matrix factorization over so called max-times algebra. It differs from the standard algebra of real numbers in that addition is replaced with the operation of taking the maximum. Also the domain is restricted to the set of nonnegative real numbers. More formally we have

DEFINITION 3.1. *The* max-times *(or* subtropical*) algebra is a set $\mathbb{R}_+$ of nonnegative real numbers together with operations $a \boxplus b = \max\{a,b\}$ (addition) and $a \boxdot b = ab$ (multiplication) defined for any $a,b \in \mathbb{R}_+$. The identity element for addition is 0 and for multiplication it is 1.*

In the future we will use the notation $a \boxplus b$ and $\max\{a,b\}$ and the names *max-times* and *subtropical* interchangeably. It is straightforward to see that the max-times algebra is a *dioid*, that is, a semiring with idempotent addition ($a \boxplus a = a$). It is important to note that subtropical algebra is anti-negative, that is, there is no subtraction operation.

A very closely related algebraic structure is the *max-plus* (*tropical*) algebra. It is defined over the set of extended real numbers $\mathbb{R} \cup \{-\infty\}$ and has operations $a \oplus b = \max\{a,b\}$ (addition) and $a \odot b = a+b$ (multiplication) with identity elements $-\infty$ (addition) and 0 (multiplication). The tropical and subtropical algebras are isomorphic, which can be seen by taking the logarithm of the subtropical algebra (with the convention that $\log 0 = -\infty$).

The subtropical matrix algebra follows naturally:

DEFINITION 3.2. *The* max-times matrix product *of two matrices $\mathbf{B} \in \mathbb{R}_+^{n\times k}$ and $\mathbf{C} \in \mathbb{R}_+^{k\times m}$ is defined as*

$$(\mathbf{B} \boxtimes \mathbf{C})_{ij} = \max_{s=1}^{k} \mathbf{B}_{is}\mathbf{C}_{sj}\ . \tag{3.1}$$

The definition of a *rank-1* matrix over the max-times algebra is the same as over the standard algebra, i.e. a matrix that can be expressed as an outer product of two vectors. We will use the term *block* to mean a rank-1 matrix. The general *rank* of a matrix over the max-times algebra is defined analogously to the standard rank:

DEFINITION 3.3. *The* max-times rank *of a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ is the least integer k such that $\mathbf{A}$ can be expressed as a (max) sum of k rank-1 matrices, $\mathbf{A} = \mathbf{F}_1 \boxplus \mathbf{F}_2 \boxplus \cdots \boxplus \mathbf{F}_k$, where all $\mathbf{F}_i$ are rank-1.*

The final concept we need is that of *dominating* matrix:

DEFINITION 3.4. *Let $\mathbf{A}$ and $\mathbf{X}$ be matrices of the same size, and let Γ be a subset of their indices. Then if for all indices $(i,j) \in \Gamma$, $\mathbf{X}_{ij} \geq \mathbf{A}_{ij}$, we say that $\mathbf{X}$* dominates $\mathbf{A}$ within Γ. *If Γ spans the entire size of $\mathbf{A}$ and $\mathbf{X}$, we simply say that $\mathbf{X}$* dominates $\mathbf{A}$. *Correspondingly, $\mathbf{A}$ is said to be* dominated by $\mathbf{X}$.

Now that we have sufficient notation, we can formally introduce the main problem considered in the paper.

PROBLEM 3.1. *Given a matrix $\mathbf{A} \in \mathbb{R}_+^{n\times m}$ and an integer $k > 0$, find factor matrices $\mathbf{B} \in \mathbb{R}_+^{n\times k}$ and $\mathbf{C} \in \mathbb{R}_+^{k\times m}$ minimizing*

$$\|\mathbf{A} - \mathbf{B} \boxtimes \mathbf{C}\|_1 = \sum_{i,j} \left|\mathbf{A}_{ij} - (\mathbf{B} \boxtimes \mathbf{C})_{ij}\right|\ . \tag{3.2}$$

We measure the reconstruction error in terms of the L_1 (absolute) error here, as our target application are discrete-valued matrices. We could, naturally, use the Frobenius norm (sum-of-squares error) instead, without changing our algorithm.

3.3 Computational complexity, sparsity, and relations to other algebras. The main contribution of this paper is to present an algorithm for solving Problem 3.1. But before presenting the algorithm, let us present some results that explain the behaviour of max-times algebra. We start by studying the computational complexity of finding max-times decompositions, after which we present results regarding the sparsity of the factor matrices and relations to other algebras.

Computational complexity. We will now prove that the max-times matrix factorization problem is computationally hard, and thus the use of non-exact methods is justified. To this end, we consider the *max-times rank decision problem*: Given $\mathbf{A} \in \mathbb{R}_+^{n \times m}$ and an integer k, is the max-times rank of $\mathbf{A}$ at most k?

THEOREM 3.1. *Computing the max-times matrix rank is* NP-*complete.*

The proof of Theorem 3.1 is an easy reduction from the Boolean Matrix Factorization problem [11] and it can be found in the supplementary material.[1] As computing the rank is clearly a special case of Problem 3.1, we see that it is NP-hard. But furthermore, we have the following corollary on the hardness of *approximating* Problem 3.1:

COROLLARY 3.1. *It is* NP-*hard to approximate Problem 3.1 to within any polynomially computable factor.*

Proof. Any algorithm that can approximate Problem 3.1 to within a factor α must find a decomposition of error $\alpha \cdot 0 = 0$ if the input matrix has exact max-times rank-k decomposition. As this implies solving the max-times rank, per Theorem 3.1 it is only possible if P=NP. ■

Sparsity of the factors. It is often desirable to obtain sparse factor matrices if the original data is sparse, as well, and the sparsity of its factors is frequently mentioned as one of the benefits of using NMF. In general, however, the factors obtained by NMF might not be sparse, but if we do restrict ourselves to *dominated* decompositions, Gillis and Glineur [7] showed that the sparsity of the factors cannot be less than the sparsity of the original matrix.

The proof of Gillis and Glineur [7] relies on the anti-negativity, and hence their proof is easy to adapt to max-times setting. Let the *sparsity* of an n-by-m matrix $\mathbf{A}$, $s(\mathbf{A})$, be defined as

$$s(\mathbf{A}) = \frac{nm - \eta(\mathbf{A})}{nm} , \tag{3.3}$$

where $\eta(\mathbf{A})$ is the number of nonzero elements in $\mathbf{A}$. Now we have

THEOREM 3.2. *Let matrices $\mathbf{B} \in \mathbb{R}_+^{n \times k}$ and $\mathbf{C} \in \mathbb{R}_+^{k \times m}$ be such that their max-times product is dominated by an n-by-m matrix $\mathbf{A}$. Then the following estimate holds*

$$s(\mathbf{B}) + s(\mathbf{C}) \geq s(\mathbf{A}) . \tag{3.4}$$

Proof. We first prove (3.4) for $k = 1$. Let $\mathbf{b} \in \mathbb{R}_+^n$ and $\mathbf{c} \in \mathbb{R}_+^m$ be such that $\mathbf{b}_i \mathbf{c}_j^T \leq \mathbf{A}_{ij}$ for all $1 \leq i \leq n$, $1 \leq j \leq m$. Since $(\mathbf{bc}^T)_{ij} > 0$ if and only if $\mathbf{b}_i > 0$ and $\mathbf{c}_j > 0$, we have

$$\eta(\mathbf{bc}^T) = \eta(\mathbf{b})\,\eta(\mathbf{c}) . \tag{3.5}$$

[1] `http://people.mpi-inf.mpg.de/~pmiettin/tropical/`

By (3.3) we have $\eta(\mathbf{bc}^T) = nm(1 - s(\mathbf{bc}^T))$, $\eta(\mathbf{b}) = n(1 - s(\mathbf{b}))$ and $\eta(\mathbf{c}) = m(1 - s(\mathbf{c}))$. Plugging these expressions into (3.5) we obtain $(1 - s(\mathbf{bc}^T)) = (1 - s(\mathbf{b}))(1 - s(\mathbf{c}))$. Hence the number of zeros in a rank-1 dominated approximation of $\mathbf{A}$ is

$$s(\mathbf{b}) + s(\mathbf{c}) \geq s(\mathbf{bc}^T) . \tag{3.6}$$

From (3.6) and the fact that the number of nonzero elements in $\mathbf{bc}^T$ is no greater than in $\mathbf{A}$, it follows that

$$s(\mathbf{b}) + s(\mathbf{c}) \geq s(\mathbf{A}) . \tag{3.7}$$

Now let $\mathbf{B} \in \mathbb{R}_+^{n \times k}$ and $\mathbf{C} \in \mathbb{R}_+^{k \times m}$ be such that $\mathbf{B} \boxtimes \mathbf{C}$ is dominated by $\mathbf{A}$. Then $\mathbf{B}_{il}\mathbf{C}_{lj} \leq \mathbf{A}_{ij}$ for all $i \in [n]$, $j \in [m]$, and $l \in [k]$, which means that for each $l \in [k]$, $\mathbf{B}_{:,l}\mathbf{C}_{l,:}$ is dominated by $\mathbf{A}$. To complete the proof observe that $s(\mathbf{B}) = k^{-1} \sum_{l=1}^{k} \mathbf{B}_{:,l}$ and $s(\mathbf{C}) = k^{-1} \sum_{l=1}^{k} \mathbf{C}_{l,:}$ and that for each l estimate (3.7) holds. ■

Relation to other algebras. Let us now study how the max-times algebra relates to other algebras, especially the standard one, the Boolean one, and the max-plus algebra. For the first two, we compare the ranks, and for the last, the reconstruction error.

The standard rank and the max-times rank are incommensurable, that is, there are matrices that have smaller max-times rank than standard rank and others that have higher max-times rank than standard rank. Let us consider an example of the first kind,

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{pmatrix} \boxtimes \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \end{pmatrix} .$$

As the decomposition shows, this matrix has max-times rank of 2, while its normal rank is easily verified to be 3. Indeed, it is easy to see that the complement of the n-by-n identity matrix $\bar{\mathbf{I}}_n$, that is, the matrix that has 0s at the diagonal and 1s everywhere else, has max-times rank of $O(\log n)$ while its standard rank is n (the result follows from similar results regarding the Boolean rank, see, e.g. [11]).

A *pattern* of a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ is an n-by-m binary matrix $\mathbf{P}$ such that $\mathbf{P}_{ij} = 0$ if and only if $\mathbf{A}_{ij} = 0$ and otherwise $\mathbf{P}_{ij} = 1$. The max-times rank of $\mathbf{A}$ can be bounded from below by the Boolean rank of its pattern $\mathbf{B}$; the proof of this statement is provided in the supplementary material.

As we have discussed earlier, max-plus and max-times algebras are isomorphic, and consequently max-times and max-plus ranks agree. Yet, the errors obtained in approximate decompositions do not have to (and usually will not) agree. In what follows we characterize the relationship between max-plus and max-times errors. We denote by $\overline{\mathbb{R}}$ the extended real line $\mathbb{R} \cup \{-\infty\}$ and by $\mathbf{A} \diamond \mathbf{B}$ the *tropical* (or *max-plus*) matrix product, i.e. the matrix product taken over the max-plus algebra.

THEOREM 3.3. *Let* $\mathbf{A} \in \overline{\mathbb{R}}^{n\times m}$, $\mathbf{B} \in \overline{\mathbb{R}}^{n\times k}$ *and* $\mathbf{C} \in \overline{\mathbb{R}}^{k\times m}$. *Let*

$$N = \max_{\substack{i\in[n]\\ j\in[m]}} \left\{ \max\left\{ \mathbf{A}_{ij}, \max_{1\le d\le k}\left\{\mathbf{B}_{id}+\mathbf{C}_{dj}\right\}\right\}\right\}$$

and let $M = \exp\{N\}$.

If an error can be bounded in max-plus algebra

$$\|\mathbf{A}-\mathbf{B}\diamond\mathbf{C}\|_F^2 \le \lambda \tag{3.8}$$

then the following estimate holds with respect to the max-times algebra

$$\|\exp\{\mathbf{A}\}-\exp\{\mathbf{B}\}\boxtimes\exp\{\mathbf{C}\}\|_F^2 \le M^2\lambda\ . \tag{3.9}$$

Proof. Denote $\alpha_{ij} = \max_{d=1}^{k}\{\mathbf{B}_{id}+\mathbf{C}_{dj}\}$. From (3.8) it follows that there exists a set of numbers $\{\lambda_{ij}\ge 0 \mid i\in[n], j\in[m]\}$ s.t. for any i,j we have $(A_{ij}-\alpha_{ij})^2 \le \lambda_{ij}$ and $\sum_{ij}\lambda_{ij} = \lambda$. By mean-value theorem for every i, j we obtain

$$\begin{aligned}&\left|\exp\{\mathbf{A}_{ij}\}-\exp\{\alpha_{ij}\}\right| = \left|\mathbf{A}_{ij}-\alpha_{ij}\right|\exp\{\alpha^*_{ij}\}\\ &\le \sqrt{\lambda_{ij}}\exp\{\alpha^*_{ij}\},\end{aligned}$$

for some $\min\{\mathbf{A}_{ij},\alpha_{ij}\} \le \alpha^*_{ij} \le \max\{\mathbf{A}_{ij},\alpha_{ij}\}$. Hence,

$$(\exp\{\mathbf{A}_{ij}\}-\exp\{\alpha_{ij}\})^2 \le \lambda_{ij}(\exp\{\max\{\mathbf{A}_{ij},\alpha_{ij}\}\})^2$$

The estimate for the max-times error now follows from the monotonicity of the exponent:

$$\begin{aligned}&\|\exp\{\mathbf{A}\}-\exp\{\mathbf{B}\}\boxtimes\exp\{\mathbf{C}\}\|_F^2\\ &\le \sum_{ij}\left(\exp\{\alpha^*_{ij}\}\right)^2\lambda_{ij} \le \sum_{ij}\left(\exp\{\max\{\mathbf{A}_{ij},\alpha_{ij}\}\}\right)^2\lambda_{ij}\\ &\le M^2\lambda\ ,\end{aligned}$$

proving the claim. ∎

4 Algorithm

Here we present an algorithm called `Capricorn` for finding the best low-rank matrix factorization over max-times algebra. That is, given a matrix $\mathbf{A}\in\mathbb{R}_+^{n\times m}$ and an integer $k>0$, it finds factor matrices $\mathbf{B}\in\mathbb{R}_+^{n\times k}$ and $\mathbf{C}\in\mathbb{R}_+^{k\times m}$ that try to minimize the error $E(\mathbf{A},\mathbf{B},\mathbf{C}) = \|\mathbf{A}-\mathbf{B}\boxtimes\mathbf{C}\|_1$.

4.1 The main algorithm. From Definition 3.2 it is clear that a matrix that has the pure max-times structure is essentially a union of overlapping rank-1 blocks – whichever block dominates the matrix at any given region determines its local landscape. The pseudo-code of the `Capricorn` algorithm is given in Algorithm 1. The algorithm accepts as input a matrix $\mathbf{A}$, a positive integer k (rank of the decomposition), and additional parameters $bucketSize, \delta, \theta, \tau$, and M, the last five being parameters for the algorithm that will be explained in the following.

`Capricorn` finds the rank-1 blocks one-by-one using the `FindBlock` procedure that we will explain in the next subsection (line 7). But max-times decompositions, similar to, say, NMF and unlike SVD, do not have the *hierarchy* property where smaller-rank decompositions are always part of higher-rank decompositions. Consequently, finding the blocks one-by-one often yields into suboptimal solutions where the first selected blocks cover large area badly, and the later blocks cannot correct for that, due to the anti-negativity of max-times algebra.

To combat that problem, after finding the first k blocks, `Capricorn` tries to discard the oldest found block and replace it with a new one. After it has tried to replace all k original blocks, it will start discarding the blocks that replaced the original blocks, continuing this process until it has found in total M times the k blocks; the best rank-k decomposition that `Capricorn` saw during this process is then returned (lines 6–14). At first glance it appears to be more intuitive to get rid of the block that is the least useful in terms of the error instead of the oldest one. But it can be very hard to predict the impact that a given block is going to have once other blocks are in place. In particular the very first found block tends to cover a big portion of the data, which is beneficial at the beginning, but becomes redundant as the algorithm progresses. By removing the oldest block we make sure that all parts of the decomposition are updated regularly.

The new blocks are found using the residual matrix $\mathbf{R}$, storing only the values that are not yet covered by the factorization. The building of the residual (line 14) reflects the winner takes it all property of the max-times algebra: if an element of $\mathbf{A}$ is approximated by any smaller value, it appears as such in the residual; if value is equal or larger, though, the corresponding residual element is *NaN*, indicating that this value is already covered.

4.2 Finding the rank-1 blocks: the `FindBlock` procedure. The key element of `Capricorn` is of course how we find the rank-1 blocks. The main idea behind the algorithm is to spot potential blocks by considering ratios of matrix rows. Consider an arbitrary rank-1 block $\mathbf{X} = \mathbf{bc}$, where $\mathbf{b}\in\mathbb{R}_+^{n\times 1}$ and $\mathbf{c}\in\mathbb{R}_+^{1\times m}$. For any indices i and j such that $\mathbf{b}_i > 0$ and $\mathbf{b}_j > 0$ we have $\mathbf{X}_j = \frac{\mathbf{b}_j}{\mathbf{b}_i}\mathbf{X}_i$. This is a characteristic property of rank-1 matrices – all rows are multiples of one another. Hence if a block $\mathbf{X}$ dominates some region Γ of a matrix $\mathbf{A}$, then rows of $\mathbf{A}$ should all be multiples of each other within Γ. These rows might have different lengths due to block overlap, in which case the rule only applies to their common part.

`FindBlock` (Algorithm 2) starts by selecting a seed row (line 2), with the intention of growing a block around it. We choose the row with the largest sum as this increases the chances of finding the most prominent block. In order to

Algorithm 1 `Capricorn`

Input: $\mathbf{A} \in \mathbb{R}_+^{n\times m}$, $k > 0$, $bucketSize > 0$, $\delta > 0$, $\theta > 0$, $\tau \in [0,1]$, $M > 0$
Output: $\mathbf{B}^* \in \mathbb{R}_+^{n\times k}$, $\mathbf{C}^* \in \mathbb{R}_+^{k\times m}$
1: **function** `Capricorn`$(\mathbf{A}, k, bucketSize, \delta, \theta, \tau, M)$
2: $\quad\mathbf{B} \leftarrow 0^{n\times 0}$, $\mathbf{C} \leftarrow 0^{0\times m}$
3: $\quad\mathbf{B}^* \leftarrow \mathbf{B}, \mathbf{C}^* \leftarrow \mathbf{C}$
4: $\quad bestError \leftarrow E(\mathbf{A},\mathbf{B},\mathbf{C})$
5: $\quad\mathbf{R} \leftarrow \mathbf{A}$ ▷ Residual matrix
6: $\quad$**for** $count \leftarrow 1$ **to** $k \times M$ **do**
7: $\quad\quad[\mathbf{b},\mathbf{c}] \leftarrow$ `FindBlock`$(\mathbf{A},\mathbf{R}, bucketSize, \delta, \theta, \tau)$
8: $\quad\quad\mathbf{B} \leftarrow [\mathbf{B},\mathbf{b}]$, $\mathbf{C} \leftarrow \begin{bmatrix}\mathbf{C}\\ \mathbf{c}\end{bmatrix}$
9: $\quad\quad$**if** $E(\mathbf{A},\mathbf{B},\mathbf{C}) < bestError$ **then**
10: $\quad\quad\quad\mathbf{B}^* \leftarrow \mathbf{B}, \mathbf{C}^* \leftarrow \mathbf{C}$
11: $\quad\quad\quad bestError \leftarrow E(\mathbf{A},\mathbf{B},\mathbf{C})$
12: $\quad\quad$**if B** has more than k columns **then**
13: $\quad\quad\quad\mathbf{B}_{:,1} \leftarrow [\,]$, $\mathbf{C}_{:,1} \leftarrow [\,]$
14: $\quad\quad\mathbf{R}_{ij} \leftarrow \begin{cases}\mathbf{A}_{ij} & (\mathbf{B}\boxtimes\mathbf{C})_{ij} < \mathbf{A}_{ij}\\ NaN & \text{otherwise}\end{cases}$
15: $\quad$**return** $\mathbf{B}^*$, $\mathbf{C}^*$

Algorithm 2 `FindBlock`

Input: $\mathbf{A} \in \mathbb{R}_+^{n\times m}, \mathbf{R} \in \mathbb{R}_+^{n\times m}$, $bucketSize > 0$, $\delta > 0$, $\theta > 0$, $\tau \in [0,1]$
Output: $\mathbf{b} \in \mathbb{R}_+^{n\times 1}$, $\mathbf{c} \in \mathbb{R}_+^{1\times m}$
1: **function** `FindBlock`$(\mathbf{A},\mathbf{R}, bucketSize, \delta, \theta, \tau)$
2: $\quad idx \leftarrow \arg\max_i \sum_j r_{ij}$
3: $\quad\mathbf{H} \leftarrow$ `CorrelationsWithRow`$(\mathbf{R}, idx, bucketSize, \delta, \tau)$
4: $\quad r \leftarrow \arg\max_i \sum_j h_{ij}$
5: $\quad c \leftarrow \arg\max_j \sum_i h_{ij}$
6: $\quad b_{idx} \leftarrow \{i \mid \mathbf{H}_{ic} = 1\}$
7: $\quad c_{idx} \leftarrow \{i \mid \mathbf{H}_{ri} = 1\}$
8: $\quad[\mathbf{b},\mathbf{c}] \leftarrow$ `RecoverBlock`$(\mathbf{R}, b_{idx}, c_{idx})$
9: $\quad\mathbf{b} \leftarrow$ `AddRows`$(\mathbf{b},\mathbf{c},\mathbf{A},\theta, bucketSize, \delta)$
10: $\quad\mathbf{c} \leftarrow$ `AddRows`$(\mathbf{c}^T,\mathbf{b}^T,\mathbf{A}^T,\theta, bucketSize, \delta)^T$
11: $\quad$**return b**, **c**

find the best block **X** that the seed row passes through, we first find a binary matrix **H** that represents the pattern of **X** (line 3). Next, on lines 4–7 we choose an approximation of the block pattern with index sets b_{idx} and c_{idx}, which define what elements of **b** and **c** should be nonzero. The next step is to find the actual values of elements within the block with the function `RecoverBlock` (line 8). Finally, we inflate the found core block with `ExpandBlock` (line 9).

The function `CorrelationsWithRow` (Algorithm 3) finds the pattern of a new block. It does so by comparing a given seed row to other rows of the matrix and extracting sets where the ratio of the rows is almost constant. As was mentioned before, if two rows locally represent the same block, then one should be a multiple of the other, and the ratios of their corresponding elements should remain level. `CorrelationsWithRow` processes the input matrix row by

Algorithm 3 `CorrelationsWithRow`

Input: $\mathbf{R} \in \mathbb{R}_+^{n\times m}$, $idx \in [n]$, $bucketSize > 0$, $\delta > 0$, $\tau \in [0,1]$
Output: $\mathbf{H} \in \{0,1\}^{n\times m}$
1: **function** `CorrelationsWithRow`$(\mathbf{R}, idx, bucketSize, \delta, \tau)$
2: $\quad$turn all NaN elements of **R** to 0
3: $\quad\mathbf{H} \leftarrow 0^{n\times m}$
4: $\quad$**for** $i \leftarrow 1$ **to** n **do**
5: $\quad\quad V_i \leftarrow$ `FindRowSet`$(\mathbf{R}_{idx,:}, \mathbf{R}_{i,:}, bucketSize, \delta)$
6: $\quad\quad\mathbf{H}(i, V_i) \leftarrow 1$
7: $\quad s \leftarrow \arg\max_{i:\, i\neq s} \sum_j h_{ij}$
8: $\quad\mathbf{H}_{idx\,:} \leftarrow \mathbf{H}_{s\,:}$
9: $\quad$**for** $i \leftarrow 1$ **to** n **do**
10: $\quad\quad$**if** $\phi(\mathbf{H}, idx, i) < \phi(\mathbf{H}, idx, s) - \tau$ **then**
11: $\quad\quad\quad\mathbf{H}_{i\,:} \leftarrow 0$
12: $\quad$**return H**

row using the function `FindRowSet`, which for every row outputs the most likely set of indices, where it is correlated with the seed row (lines 4–6). Since the seed row is obviously the most correlated with itself, we compensate for this by replacing its pattern with that of the second most correlated row (lines 7–8). Finally, we drop some of the least correlated rows after comparing their correlation value ϕ to that of the second most correlated row (after the seed row). The correlation function ϕ is defined as follows

$$\phi(\mathbf{H}, idx, i) = \frac{\langle \mathbf{H}_{i,:}, \mathbf{H}_{idx,:}\rangle}{\langle \mathbf{H}_{i,:}, \mathbf{H}_{i,:}\rangle + 1}\,. \tag{4.10}$$

The parameter τ is a threshold determining whether a row should be discarded or retained. The auxiliary function `FindRowSet` (Algorithm 4) compares two vectors and finds the biggest set of indices where their ratio remains almost constant. It does so by sorting the log-ratio of the input vectors into buckets of fixed size and then choosing the bucked with the most elements. It accepts two additional parameters: *bucketSize* and δ. If the largest bucket has fewer than *bucketSize* elements, the function will return an empty set – this is done because very small patterns do not reveal much structure and are mostly accidental. The width of the buckets is determined by the parameter δ.

At this point we know the pattern of the new block, that is, the locations of its non-zeros. To fill in the actual values, we consider the submatrix defined by the pattern, and find the best (in L_2 sense) rank-1 approximation of it. The pseudo-code of the `RecoverBlock` procedure doing this is presented in the supplementary material.

4.3 Extending found blocks. Since blocks often heavily overlap, we are susceptible to finding only fragments of patterns in the data – some parts of a block can be dominated by another block and subsequently not recognized. Hence we need to expand found blocks to make them complete. This is done separately for rows and columns in the method

Algorithm 4 `FindRowSet`

Input: $\mathbf{u} \in \mathbb{R}_+^m, \mathbf{v} \in \mathbb{R}_+^m, bucketSize > 0, \delta > 0$
Output: $V \subset [m]$

1: **function** `FindRowSet`$(\mathbf{u}, \mathbf{v}, bucketSize, \delta)$
2: $\quad \mathbf{r} \leftarrow \log(\mathbf{u}\,./\,\mathbf{v})$
3: $\quad nBuckets \leftarrow \lceil(\max\{r\} - \min\{r\})/\delta\rceil$
4: $\quad$ **for** $i \leftarrow 0$ **to** $nBuckets$ **do**
5: $\quad\quad V_i \leftarrow \{idx \in [m] \mid \min\{\mathbf{r}\} + i\delta \leq r_{idx} < \min\{\mathbf{r}\} + (i+1)\delta\}$
6: $\quad V \leftarrow \arg\max\{|V_i| \mid i = 1, \ldots, nBuckets\}$
7: $\quad$ **if** $|V| < bucketSize$ **then**
8: $\quad\quad V \leftarrow \emptyset$
9: $\quad$ **return** V

Algorithm 5 `AddRows`

Input: $\mathbf{b} \in \mathbb{R}_+^{n\times 1}$, $\mathbf{c} \in \mathbb{R}_+^{1\times m}$, $\mathbf{A} \in \mathbb{R}_+^{n\times m}$, $\theta > 0$, $bucketSize > 0$, $\delta > 0$
Output: $\mathbf{b} \in \mathbb{R}_+^{n\times 1}$

1: **function** `AddRows`$(\mathbf{b}, \mathbf{c}, \mathbf{A}, \theta, bucketSize, \delta)$
2: $\quad b_{idx} \leftarrow \{t \mid \mathbf{b}_t > 0\}$
3: $\quad$ **for** $i \in [n] \setminus b_{idx}$ **do**
4: $\quad\quad V_i \leftarrow$ `FindRowSet`$(\mathbf{c}, \mathbf{R}_{i,:}, bucketSize, \delta)$
5: $\quad\quad$ **if** $V_i = \emptyset$ **then**
6: $\quad\quad\quad$ **continue**
7: $\quad\quad \alpha \leftarrow mean(\mathbf{R}_{iV_i}./\mathbf{c}_{V_i})$
8: $\quad\quad impact \leftarrow \frac{\sum_{s\in V_i} \max\{0, \alpha c_s - \mathbf{A}_{is}\}}{\sum_{s\in V_i} \mathbf{A}_{is} - |\mathbf{A}_{is} - \alpha c_s|}$
9: $\quad\quad$ **if** $impact \leq \theta$ **then**
10: $\quad\quad\quad \mathbf{b}_i \leftarrow \alpha$
11: $\quad$ **return** $\mathbf{b}$

called `AddRows` (Algorithm 5), which, given a starting block $\mathbf{X} = \mathbf{bc}$ and the original matrix $\mathbf{A}$, tries to add new nonzero elements to $\mathbf{b}$. It iterates through all rows of $\mathbf{A}$ and adds those that would make a positive impact on the objective without unnecessarily overcovering the data. In order to decide whether a given row should be added, it first extracts a set V_i of indices where this row is a multiple of the row vector $\mathbf{c}$ of the block (if they are not sufficiently correlated, then the row does not belong to the block) (line 4). A row is added if the evaluation of the following function (line 8)

$$\psi(\alpha) = \frac{\sum_{s\in V_i} \max\{0, \alpha c_s - \mathbf{A}_{is}\}}{\sum_{s\in V_i} \mathbf{A}_{is} - |\mathbf{A}_{is} - \alpha c_s|} \tag{4.11}$$

is below the threshold θ. In (4.11) the numerator measures by how much the new row would overcover the original matrix, and the denominator reflects the improvement in the objective compared to a zero row.

4.4 Complexity. In order to estimate the theoretical complexity of `Capricorn` it suffices to do this for the `FindBlock` routine since it does most of the work. There are three main contributors to its runtime: `CorrelationsWithRow`, `RecoverBlock`, and `AddRows`. `CorrelationsWithRow` compares every row to the seed row, each time calling `FindRowSet`, which in turn has to process all m elements of both rows. This gives the total complexity of `CorrelationsWithRow` to be $O(nm)$. To find the complexity of `RecoverBlock`, first observe that any "pure" block $\mathbf{X}$ can be represented as $\mathbf{X} = \mathbf{bc}$, where $\mathbf{b} \in \mathbb{R}^{n'\times 1}$ and $\mathbf{c} \in \mathbb{R}^{1\times m'}$ with $n' \leq n$ and $m' \leq m$. `RecoverBlock` selects $\mathbf{c}$ from the rows of $\mathbf{X}$ and then finds the corresponding column vector $\mathbf{b}$ that minimizes $\|\mathbf{X} - \mathbf{bc}\|_F$. In order to select the best row, we have to try each of the n' candidates, and since finding the corresponding $\mathbf{b}$ for each of them takes time $O(n'm')$, this gives the runtime of `RecoverBlock` as $O(n')O(n'm') = O(n^2m)$. The most computationally expensive parts of `AddRows` are `FindRowSet` (line 4), finding the mean (line 7), and computing the impact (line 8), which all run in $O(m)$ time. All of these operations have to be repeated $O(n)$ times, and hence the runtime of `AddRows` is $O(nm)$. Note that if we have k blocks, then the `FindBlock` routine will execute Mk times, where M is a fixed parameter. Thus, we can now estimate the complexity of `Capricorn` to be $O(k)(O(nm) + O(n^2m) + O(nm)) = O(n^2mk)$.

5 Experiments

We tested `Capricorn` with both synthetic and real-world data. The purpose of the synthetic experiments is to evaluate the properties of the algorithm in controlled environments where we know the data has the max-times structure. The purpose of the real-world experiments is to confirm that these observations also hold true in real-world data, and to study what kinds of data sets actually have max-times structure. The implementation of `Capricorn` is freely available.[2]

5.1 Other methods. We compared `Capricorn` against three other methods: `SVD` and two versions of NMF. For `SVD`, we used Matlab's built-in implementation. The two versions of NMF are differentiated by their capability to handle missing values. The first method, called simply `NMF`, by Kim and Park [9], cannot handle missing values, while the other method, which we call here `WNMF`, by Li and Ngom [10], is designed to deal with them.

5.2 Synthetic experiments. The synthetic experiments were performed over 1000-by-800 matrices with true max-times rank 10. All results presented in this section are averaged over 10 instances. For reconstruction error tests, we compared against `SVD` and `NMF` on the data with no unknown values. As the data contains max-times structure, we expect to be consistently better than either of them (and indeed, we are). The reconstruction error is measured as the relative Frobenius norm $\|\tilde{\mathbf{A}} - \mathbf{A}\|_F / \|\mathbf{A}\|$, where $\mathbf{A}$ is the data and $\tilde{\mathbf{A}}$ its approximation, as that is the measure both `SVD` and `NMF`

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

aim at minimizing; the results with L_1 norm are presented in the supplementary material.

Varying density. In our first experiment we studied the effects of varying the density of the factor matrices. We varied the density of the factors from 10% to 100% with an increment of 10%. There are two different versions of this setup – the first one having 10% noise (Figure 1a), and a high-noise variation containing 50% noise (Figure 1b). In both cases Capricorn is consistently the best method, obtaining almost perfect reconstruction consistently in low-noise case. With higher density and noise levels, the reconstruction error starts to increase slightly, reaching level of approximately 10% at 100% density. That SVD and NMF start behaving better at higher levels of density indicates that these matrices can be explained relatively well using standard algebra.

Varying noise. Next we tested the effects of noise. The amount of noise is always with respect to the number of nonzero elements in a matrix, that is, for a matrix $\mathbf{A}$ with $\kappa(\mathbf{A})$ nonzero elements and noise level α, we flip $\alpha\kappa(\mathbf{A})$ elements to random values. The level of noise varied from 0% to 110% with increments of 10%. Again, this experiment was performed in two different setups: one with 30% factor density (Figure 1c), and the other with factor density of 60% (Figure 1d).

In the low-density case, Capricorn is consistently the best method with essentially perfect reconstruction for up to 80% of noise. In the high-density case, however, the noise has more severe effects, and in particular after 60% of noise, SVD and NMF are already better than Capricorn. The severity of the noise is, at least partially, explained by the fact that in the denser data we flip more elements than in sparser data: as the data matrices are essentially full, by 50%, we have replaced half of the values in the matrices with random values. Further, the quick increase of the reconstruction error for Capricorn hints strongly that the max-times structure of the data is mostly gone at these noise levels.

Varying the rank. For the last reconstruction error test we tested the effects of the (max-times) rank, with the assumption that higher-rank matrices are harder to reconstruct. The true max-times rank of the data varied from 2 to 20 with increments of 2. There are four variations of this experiment: with 30% factor density and 10% noise (Figure 1e), with 30% factor density and 50% noise (Figure 1f), with 60% factor density and 10% noise (Figure 1g), and with 60% factor density and 50% noise (Figure 1h).

Capricorn obtains perfect reconstruction in all cases except high noise and high density (Figure 1h), where we can see that higher ranks increase the reconstruction error. Interestingly, the inverse is true for SVD and NMF.

Prediction. In this experiment we choose a random holdout set and remove it from the data (elements of this set are marked as missing values). We then try to learn the structure of the data from the remaining part of the data using Capricorn and WNMF, and finally test how well they predict the values inside the holdout set. All input matrices are integer-valued and since the recovered data produced by the algorithms can be continuous-valued, we round it to the nearest integer. The quality of the prediction is measured as the fraction of correct values in the hold-out set, and the results are reported in Figure 2.

As can be seen in Figure 2, as the fraction of held-out data increases, Capricorn's results get worse, as expected, but it still is consistently better than WNMF that does not seem to be able to recover any specific structure.

Discussion. The synthetic experiments confirm that Capricorn is able to recover matrices with max-times structure even when this structure has been perturbed with high levels of noise. Furthermore, the experiments confirm that NMF or SVD cannot recover structure from matrices with max-times structure, that is, we cannot use existing methods as a substitute to find the max-times structure neither for the reconstruction nor for the prediction tasks.

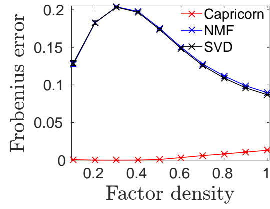
5.3 Real-world experiments. As was mentioned before, Capricorn is not exactly an approximation algorithm since it only returns nontrivial decomposition if the subtropical structure is present in the data. This is because it works by finding patterns in the data and adding them to the decomposition rather than minimizing the distance to the input matrix. Consequently, if an existing pattern can be completed by covering some zeros in the data, Capricorn is likely to do that, even if it means increasing the reconstruction error. This behaviour is beneficial when dealing with missing values, but the main drawback is that some zeros may be filled with positive values, which will increase the error. Unlike with synthetic experiments, here we are not so much interested in getting all the elements of the input matrix exactly, but rather we aim to extract as much subtropical structure as possible. In order to facilitate this we turn all zero elements in the data into missing values and when computing the error, we ignore them. It is important to mention that neither SVD nor NMF support missing values – so we feed them the original data without replacing zeros. Although they can find the structure in the nonzero parts of the data, they also tend to fit the zeros, which generally increases the measured error since we only care about nonzero entries. For this reason we also run the same setup with WNMF, which can ignore missing values.

We used the following two datasets, both of which are available at the University of Florida Sparse Matrix Collection[3] [6]. The first one is called *Bas1LP* and represents a linear program.[4] The other dataset is called *Trec12* and it is a brute force disjoint product matrix in tree algebra on *n* nodes.[5]

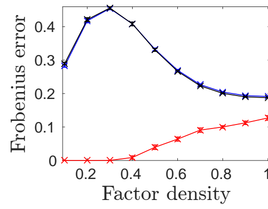
[3] http://www.cise.ufl.edu/research/sparse/matrices/

[4] Submitted to the matrix repository by Csaba Meszaros.

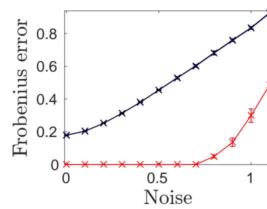
[5] Submitted by Nicolas Thiery.



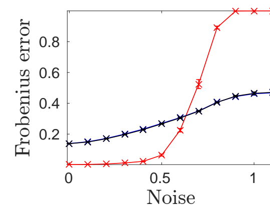
(a) Varying density with low level of noise.



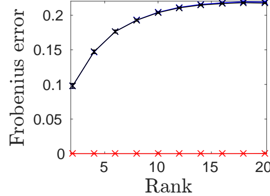
(b) Varying density test with high level of noise.



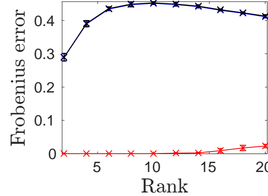
(c) Varying noise with low density.



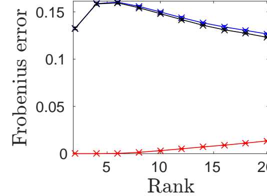
(d) Varying noise with high density.



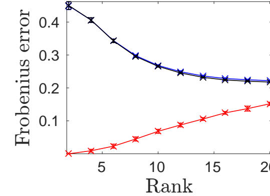
(e) Varying the rank with 10% noise and 30% factor density.



(f) Varying the rank with 50% noise and 30% factor density.



(g) Varying the rank with 10% noise and 60% factor density.



(h) Varying the rank with 50% noise and 60% factor density.

Fig. 1: **Reconstruction errors on synthetic data** using different parameter settings. x-axis is the parameter varied and y-axis is the relative Frobenius norm. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

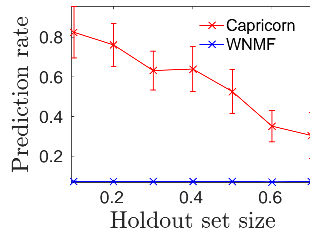


Fig. 2: **Prediction rate on synthetic data**. x-axis represents the size of the holdout set and y-axis is the correct prediction rate (higher is better). All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

Tab. 1: Reconstruction error with respect to the L_1 norm on various real-world data sets.

	Bas1LP		Trec12	
Algorithm	$k = 20$	$k = 30$	$k = 20$	$k = 30$
`Capricorn`	32.1	26.2	57.5	54.8
`NMF`	48.0	40.6	48.5	45.5
`WNMF`	6.4	4.3	21.8	18.9
`SVD`	45.4	37.6	44.0	39.7

Tab. 2: Reconstruction error on non-zero elements with respect to the L_1 norm on various real-world data sets.

	Bas1LP		Trec12	
Algorithm	$k = 20$	$k = 30$	$k = 20$	$k = 30$
`Capricorn`	18.7	19.6	42.3	40.9
`NMF`	48.0	40.6	48.5	45.5
`WNMF`	6.4	4.3	21.8	18.9
`SVD`	45.4	37.6	44.0	39.7

In the following we conduct two types of experiments on both datasets. In the first setup we try to reconstruct the original data using the algorithms and then measure the resulting error with respect to L_1 and Frobenius errors (we present the former results here and postpone the latter ones to the supplement). In the second setup we select a random holdout set and test how well the algorithms can predict values in it based on the rest of the data.

Real-world data will probably not have pure subtropical structure, hence the purpose with the real-world tests is to study if we can recover sensible amount of structure. For reconstruction (Tables 1 and 2), both real-world data sets can be expressed best using `WNMF`. With Bas1LP, `Capricorn` is second-best (although with a wide margin), but worst with Trec12 in Table 1. When we consider only the non-zero

Tab. 3: Prediction accuracy on various real-world data sets.

Algorithm	Bas1LP	Trec12
Capricorn	74.0	19.8
NMF	23.4	18.3
WNMF	85.2	39.9
SVD	28.2	20.5

Tab. 4: Prediction accuracy on non-zero predictions on various real-world data sets.

Algorithm	Bas1LP	Trec12
Capricorn	85.2	39.3
NMF	29.1	19.6
WNMF	93.1	49.8
SVD	29.1	22.5

elements, though, Capricorn is second or third best with Trec12, depending on the rank (Table 2).

When predicting the missing values, WNMF is again the best method, although with much smaller margin (Tables 3 and 4), whereas Capricorn is consistently the second-best. Overall the real-world experiments show that Capricorn is capable of extracting fair amount of subtropical structure from the tested real-world data sets.

Running times. Table 5 shows the execution time of Capricorn on two real-world datasets. All experiments were performed on a machine with eight Intel Xeon 2.4GHz cores and 48GB RAM. It is worth noting though that Capricorn is not easily parallelizable, and was effectively using only one core.

6 Conclusions

Doing the matrix factorization over the max-times (or subtropical) algebra allows us to recover structure that is hard to capture using the standard algebra. While finding the best max-times decomposition is computationally hard, our experiments show that our algorithm, Capricorn, is capable of successfully recovering at least some structure. Not every data exhibits subtropical structure, though, but in these cases Capricorn generally will not claim to have found any structure, either.

Tab. 5: Runtime of Capricorn on real-world datasets in seconds.

k	Bas1LP	Trec12
10	3612	404
15	5281	538
20	6467	677
25	8539	809
30	10346	941

Notwithstanding the good results we showed, Capricorn has its limitations. Most importantly, it will not work well on matrices with continuous values, as the FindBlock procedure assumes discrete values. Extending Capricorn to continuous-valued matrices hence remains an important step for future research. That would also help on finding more data sets that exhibit subtropical structure and thus help us gauge the usefulness of the subtropical algebra in data analysis.

References

[1] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and linearity*, volume 3. Wiley New York, 1992.

[2] R. Bělohlávek and M. Krmelova. Factor analysis of ordinal data via decomposition of matrices with grades. *Ann Math Artif Intell*, 72(1-2):23–44, Jan. 2014.

[3] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems (The International Series on Discrete Event Dynamic Systems)*. Springer, 1 edition, Sept. 1999.

[4] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. John Wiley & Sons, Chichester, 2009.

[5] G. Cohen, S. Gaubert, and J.-P. Quadrat. Max-plus algebra and system theory: where we are and where to go now. *Annual Reviews in Control*, 23:207–219, 1999.

[6] T. A. Davis and Y. Hu. The university of Florida sparse matrix collection. *ACM Trans Math Soft*, 38(1):1, 2011.

[7] N. Gillis and F. Glineur. Using underapproximations for sparse nonnegative matrix factorization. *Pattern Recognition*, 43(4):1676–1687, 2010.

[8] L. Hogben. *Handbook of linear algebra*. CRC Press, 2006.

[9] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 353–362. IEEE, 2008.

[10] Y. Li and A. Ngom. The non-negative matrix factorization toolbox for biological data mining. *Source code for biology and medicine*, 8(1):1–15, 2013.

[11] P. Miettinen. *Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms*. PhD thesis, Department of Computer Science, University of Helsinki, 2009.

[12] J. Weston, R. J. Weiss, and H. Yee. Nonlinear latent factorization by embedding multiple user interests. In *RecSys '13*, pages 65–68, 2013.