Rank Matrix Factorisation

Thanh Le Van[†], Matthijs van Leeuwen[†], Siegfried Nijssen[†,*], Luc De Raedt[†]

[†] Department of Computer Science, KU Leuven, Belgium
[*] Leiden Institute for Advanced Computer Science, Universiteit Leiden, The Netherlands

Abstract. We introduce the problem of *rank matrix factorisation* (RMF). That is, we consider the decomposition of a rank matrix, in which each row is a (partial or complete) ranking of all columns. Rank matrices naturally appear in many applications of interest, such as sports competitions. Summarising such a rank matrix by two smaller matrices, in which one contains partial rankings that can be interpreted as local patterns, is therefore an important problem.
After introducing the general problem, we consider a specific instance called Sparse RMF, in which we enforce the rank profiles to be sparse, i.e., to contain many zeroes. We propose a greedy algorithm for this problem based on integer linear programming. Experiments on both synthetic and real data demonstrate the potential of rank matrix factorisation.

Keywords: matrix factorisation; rank data; integer linear programming

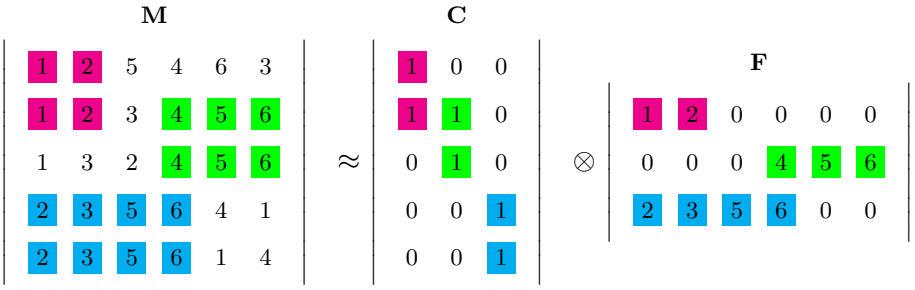
1 Introduction

In this paper, we study specific type of matrix called *rank matrices*, in which each row is a (partial or complete) ranking of all columns. This type of data naturally occurs in many situations of interest. Consider, for instance, sailing competitions where the columns could be sailors and each row would correspond to a race, or consider a business context, where the columns could be companies and the rows specify the rank of their quotation for a particular service. Rankings are also a natural abstraction of numeric data, which often arises in practice and may be noisy or imprecise. Especially when the rows are incomparable, e.g., when they contain measurements on different scales, transforming the data to rankings may result in a more informative representation.

Given a rank matrix, we are interested in discovering a set of rankings that repeatedly occur in the data. Such sets of rankings can be used to succinctly summarise the given rank matrix. With this aim, we introduce the problem of *rank matrix factorisation* (RMF). That is, we consider the decomposition of a rank matrix into two smaller matrices.

To illustrate the problem of rank matrix factorisation, let us consider the toy example in Figure 1. It depicts a rank matrix that is approximated by the product of two smaller matrices. Rank matrix $\mathbf{M}$ consists of five rows and six columns. Assuming no ties and complete rankings, each row contains each of the

Fig. 1: Rank matrix factorization toy example. Rank matrix $\mathbf{M}$ is approximated by the product of indicator matrix $\mathbf{C}$ and sparse profile matrix $\mathbf{F}$ ($k = 3$).



numbers one to six exactly once. Now, the task is to decompose a $n \times m$ rank matrix $\mathbf{M}$ into a $n \times k$ matrix $\mathbf{C}$ and a $k \times m$ matrix $\mathbf{F}$, where $\mathbf{C}$ is a binary indicator matrix, $\mathbf{F}$ consists of *rank profiles*, and k is a user-specified parameter. Intuitively the rank profiles in $\mathbf{F}$ are (partial) rankings and can be interpreted as *local patterns*. For example, together $\mathbf{C}$ and $\mathbf{F}$ show that the first two columns are ranked first and second in the first row.

In this paper we focus on a specific rank matrix factorisation problem: the problem of finding sparse rank profiles where rows of $\mathbf{F}$ contain zeroes. This allows us to discover recurrent structure that occurs in the rankings of $\mathbf{M}$, and not to focus on any noise that may be present. Within this setting we do not necessarily aim at finding a factorisation that approximates the original matrix as closely as possible; the *reconstructed rank matrix* $\mathbf{C} \otimes \mathbf{F}$ may deviate from $\mathbf{M}$, as long as its overall structure is captured. Hence, here we focus on one specific of choices within the RMF framework; we would like to stress that within the generic framework many other choices are possible. The same can be said with regard to the choices made for, e.g., rank profile aggregation and quantification of the reconstruction error. RMF is a general framework with numerous possibilities, and we propose and solve a first instance to demonstrate its potential.

The key contributions of our paper are 1) the introduction of the problem of rank matrix factorisation (RMF), 2) the introduction of a scoring function and an algorithm, based on integer linear programming, for Sparse RMF, an instance of rank matrix factorisation, and 3) an empirical evaluation on synthetic and real-life datasets that demonstrates the potential of RMF. It is shown that rank matrix factorisations can provide useful insights by revealing the rankings that underlie the data.

2 Related work

To the best of our knowledge, we are the first to investigate the problem of rank matrix factorisation. Mining rank data, although a very new topic, has attracted some attention by the community lately. In our earlier work [1] we proposed

to mine *ranked tiles*, e.g., rectangles with high ranks, and we will empirically compare to them in the experiments. Furthermore, Henzgen and Hüllermeier [2] proposed to mine frequent subrankings. The latter approach aims to mine individual patterns, whereas we aim to find a *set of patterns* that together covers most of the data.

RMF is clearly related to matrix factorisation approaches such as NMF [3, 4], BMF [5, 6], and positive integer matrix factorisation (PIMF) [7]. NMF, PIMF, and RMF have in common that the values in the factorisation are constrained to be positive, but are quite different otherwise. RMF specifically targets rank data, which requires integer values, making the results easier to interpret, a different scoring function, and a different algebra. RMF considers rank matrices instead of Boolean matrices and is therefore clearly different from BMF.

3 Rank matrix factorisation

In this section we formally define rank matrices and introduce the rank matrix factorisation problem that we consider.

Definition 1 (Rank matrix). *Let $\boldsymbol{M}$ be a matrix consisting of m rows and n columns. Let $\mathcal{R} = \{1, ..., m\}$, $\mathcal{C} = \{1, ..., n\}$ be index sets for rows and for columns respectively. The matrix $\boldsymbol{M}$ is a rank matrix iff:*

$$\forall r \in \mathcal{R} : \cup_{c \in \mathcal{C}} \boldsymbol{M}_{r,c} \subseteq \sigma, \tag{1}$$

where $\sigma = \{1, 2, ..., n\} \cup \{0\}$.

In our setting, columns are items or products that need to be ranked; rows are rankings of items. Here, the rank value 0 has a special meaning. It denotes *unknown* rankings. For example, in rating datasets, it might happen that there are items that are not rated. Such items will have rank value 0.

Given a rank matrix, we would like to find a short description of the rank matrix in terms of a fixed number of rank profiles, or patterns, consisting of partial rankings. We formalise this problem as a matrix factorisation problem.

Problem 1 (Rank matrix factorisation) *Given a rank matrix $\boldsymbol{M} \in \sigma^{m \times n}$ and an integer k, find a matrix $\boldsymbol{C}^* \in \{0, 1\}^{m \times k}$ and a matrix $\boldsymbol{F}^* \in \sigma^{k \times n}$ such that:*

$$(\boldsymbol{C}^*, \boldsymbol{F}^*) \equiv \operatorname*{argmax}_{\boldsymbol{C}, \boldsymbol{F}} \; d(\boldsymbol{M}, \boldsymbol{C} \otimes \boldsymbol{F}). \tag{2}$$

where $d(,)$ is a scoring function that measures how similar the rankings in the two matrices are, and $\otimes$ is an operator that creates a data matrix based on two factor matrices. Rows $\boldsymbol{F}_{i,:}$ of matrix $\boldsymbol{F}$ indicate partial rankings, columns $\boldsymbol{C}_{:,i}$ of matrix $\boldsymbol{C}$ indicate in which rows a partial ranking appears.

Within our generic problem statement we first need to specify the operator $\otimes$. If multiple patterns are present in one row, this operator essentially needs to combine the different partial rankings into a single ranking. This problem

is well-known in the literature as the problem of *rank aggregation*. In this first study, we use a very simple aggregation operator, namely, we use normal matrix multiplication to combine the matrices. More complex types of aggregation are left for future work.

An important drawback of normal matrix multiplication is that the product $\mathbf{CF}$ is not necessarily a rank matrix even if $\mathbf{C}$ is binary and $\mathbf{F}$ contains partial rankings. We address this here by restricting the set of acceptable matrices to those for which $(\mathbf{CF})_{ij} \leq n$ for all $i \in \mathcal{R}$ and $j \in \mathcal{C}$.

Next, we need to define the scoring function d. In the definition of this function we first need the concept of a *cover* for a rank matrix factorisation. The cover of a factorisation is the set of cells in the reconstructed matrix where at least one pattern occurs, i.e., where the reconstructed matrix is non-zero.

Definition 2 (Ranked factorisation cover).

$$cover(\boldsymbol{C}, \boldsymbol{F}) \equiv \{(i,j) | i \in \mathcal{R}, j \in \mathcal{C}, (\boldsymbol{CF})_{i,j} \neq 0\}. \tag{3}$$

Coverage is the size of the cover, i.e., $coverage(\mathbf{C}, \mathbf{F}) = |cover(\mathbf{C}, \mathbf{F})|$.

To support the aim of mining patterns in rank matrices, the scoring function $d(,)$ in Equation 2 needs to be designed in such a way that it: 1) rewards patterns that have a high coverage, 2) penalises patterns that make a large error within the cover of the factorisation.

To penalise patterns that make a large error, we define an error term that quantifies the disagreements between the reconstructed and the original rank matrix. We first define notation for the data matrix identified by the cover of a factorization.

Definition 3 (Ranked data cover). *The ranked data cover matrix $\boldsymbol{U}(\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{F})$ is a matrix with cells u_{ij}, where:*

$$u_{ij} = \begin{cases} \boldsymbol{M}_{i,j} & \text{if } (i,j) \in coverage(\boldsymbol{C}, \boldsymbol{F}) \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Now the ranked factorisation error is defined as follows.

Definition 4 (Ranked factorisation error).

$$error(\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{F}) = \sum_{i=1}^{m} d(\boldsymbol{U}(\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{F})_{i,:}, \boldsymbol{C}_{i,:}\boldsymbol{F}) \tag{5}$$

Here, $d(\cdot, \cdot)$ is a function that measures the disagreement between two rankings over the same items.

Hence, the ranked factorisation error is the total of rank disagreements between the reconstructed rank matrix and the true ranks in the original rank matrix. The score is calculated row by row.

Many scoring functions can be used to measure the disagreement between rows, for instance, Kendall's tau or Spearman's Footrule (see [8] for a survey). For an efficient computation, we choose the Footrule scoring function.

Definition 5 (Footrule scoring function). *Given two rank vectors,* $\boldsymbol{u} = (u_1, \ldots, u_n)$ *and* $\boldsymbol{v} = (v_1, \ldots, v_n)$*, the Footrule scoring function is defined as* $d_F(\boldsymbol{u}, \boldsymbol{v}) = \sum_{i=1}^{n} |u_i - v_i|$.

Having defined the ranked factorisation coverage and ranked factorisation error, we now can completely define the Sparse Rank Matrix Factorisation (Sparse RMF) problem as solving the following maximisation problem:

$$(\mathbf{C}^*, \mathbf{F}^*) \equiv \underset{\mathbf{C},\mathbf{F}}{\operatorname{argmax}}\ d(\mathbf{M}, \mathbf{CF}) \tag{6}$$

$$\equiv \underset{\mathbf{C},\mathbf{F}}{\operatorname{argmax}}\ \alpha * coverage(\mathbf{C}, \mathbf{F}) - error(\mathbf{M}, \mathbf{C}, \mathbf{F}) \tag{7}$$

$$= \underset{\mathbf{C},\mathbf{F}}{\operatorname{argmax}} \sum_{i=1}^{m}\sum_{j=1}^{n}(\alpha[(i,j) \in coverage(\mathbf{C}, \mathbf{F})] - |\mathbf{U}(\mathbf{M}, \mathbf{C}, \mathbf{F})_{ij} - \sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,j}|) \tag{8}$$

where α is a threshold and [.] are the Iverson brackets.

Note that in this scoring function, for each cell we have a positive term if the error is smaller than α; we have a negative term if the error is larger than α. In practice, we often use a relative instead of an absolute threshold. We denote such a threshold as a percentage, i.e., $\alpha = a\%$ implies $\alpha = a\% \times n$.

4 Sparse RMF using integer linear programming

We propose a greedy algorithm that uses integer linear programming (ILP). First, we present two theorems that can be used to calculate the ranked factorisation coverage and ranked factorisation error. Then, we present the algorithm.

Theorem 1. *Let* $\boldsymbol{CF}$ *be a decomposition of a rank matrix* $\boldsymbol{M}$*. Let* $\boldsymbol{A} \in \{0,1\}^{m \times n}$ *satisfy the following two properties:*

$$\boldsymbol{A}_{i,j} \leq \sum_{t=1}^{k} \boldsymbol{C}_{i,t}\boldsymbol{F}_{t,j} \tag{9}$$

$$n\boldsymbol{A}_{i,j} \geq \sum_{t=1}^{k} \boldsymbol{C}_{i,t}\boldsymbol{F}_{t,i} \tag{10}$$

then

$$\mathbf{A}_{ij} = 1 \leftrightarrow (i,j) \in cover(\boldsymbol{C}, \boldsymbol{F}) \tag{11}$$

Theorem 2. *Let* $\boldsymbol{A}$ *be a binary matrix that satisfies Theorem 1, then*

$$error(\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{F}) = \sum_{i=1}^{m}\sum_{j=1}^{n} |\boldsymbol{M}_{i,j}\boldsymbol{A}_{i,j} - \sum_{t=1}^{k} \boldsymbol{C}_{i,t}\boldsymbol{F}_{t,j}| \tag{12}$$

Given a binary matrix $\mathbf{A}$ satisfying Theorem 1, the ranked factorisation in Equation 8 can be formulated as:

$$\arg\max_{C,F,Y} \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha\mathbf{A}_{i,j} - \mathbf{Y}_{i,j} \tag{13}$$

subject to

$$\mathbf{M}_{i,j} - \sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,j} \leq \mathbf{Y}_{i,j} \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n \tag{14}$$

$$-\mathbf{M}_{i,j} + \sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,j} \leq \mathbf{Y}_{i,j} \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n \tag{15}$$

$$\mathbf{A}_{i,j} \leq \sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,j} \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n \tag{16}$$

$$n\mathbf{A}_{i,j} \geq \sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,i} \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n \tag{17}$$

$$\sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,j} \leq n \qquad \text{for } i = 1,\ldots,m,\ j = 1,\ldots,n \tag{18}$$

$$\mathbf{C}_{i,t} \in \{0,1\} \qquad \text{for } i = 1,\ldots,m,\ t = 1,\ldots,k \tag{19}$$

$$\mathbf{F}_{t,j} \in \sigma \qquad \text{for } j = 1,\ldots,n,\ t = 1,\ldots,k \tag{20}$$

where $\mathbf{Y}_{i,j}$ is the upper bound of $|\mathbf{M}_{i,j} - \sum_{t=1}^{k} \mathbf{C}_{i,t}\mathbf{F}_{t,j}|, i = 1,\ldots,m,\ j = 1,\ldots,n$.

Inequalities (14) and (15) are introduced to remove the absolute operator of the summations in Equation (12). Inequalities (16) and (17) are due to Theorem 1. Inequality (18) ensures that the reconstructed matrix is a rank matrix.

Note that the newly introduced optimisation problem in (13) - (20) is an ILP problem if either $\mathbf{C}$ or $\mathbf{F}$ is known. This makes it possible to apply an EM-style algorithm as shown in Algorithm 1, in which the matrix $\mathbf{F}$ is optimised given matrix $\mathbf{C}$, and matrix $\mathbf{C}$ is optimised given matrix $\mathbf{F}$, and we repeat the iterative optimisation till the optimal score cannot be improved any more.

To avoid local maxima, we need to initialise the iterative process in a reasonable way, i.e., smarter than random. The solution we choose is to initialise the matrix $\mathbf{C}$ using the well-known K-means algorithm. To compute the similarities of rank vectors in K-means, we use the Footrule scoring function. The K-means algorithm clusters the rows in k groups, which can be used to initialise the k columns of $\mathbf{C}$. Note that this results in initially disjoint patterns, in terms of their covers, but the iterative optimisation approach may introduce overlap.

We implemented the algorithm in OscaR[1], which is an open source Scala toolkit for solving Operations Research problems. OscaR supports a modelling

[1] `https://bitbucket.org/oscarlib/oscar/wiki/Home`.

language for ILP. We configured OscaR to use Gurobi[2] as the back-end solver. Source code can be downloaded from our website, `http://dtai.cs.kuleuven.be/CP4IM/RMF`.

Algorithm 1 *Sparse RMF algorithm*

Require: Rank matrix $\mathbf{M}$, integer k, threshold α
Ensure: Factorisation $\mathbf{C}$, $\mathbf{F}$
1: Initialise $\mathbf{C}$ using K-means algorithm
2: **while** not converged **do**
3: $\mathbf{F} \leftarrow$ Optimise (13) - (20) given $\mathbf{C}$
4: $\mathbf{C} \leftarrow$ Optimise (13) - (20) given $\mathbf{F}$
5: **end while**

5 Experiments on synthetic datasets

The goal of Sparse RMF is to find a set of rank profiles (local patterns), which can be used to summarise a given rank matrix. Alternative methods for summarising matrices are bi-clustering [9] and ranked tiling [1]. While ranked tiling and Sparse RMF work on ranked data, bi-clustering algorithms are mostly applied to numeric data. Hence, to compare to all of these algorithms, we first generate continuous data and then convert them to ranked data as in [1]. The main idea is to benchmark the performance of the considered algorithms in terms of recovering implanted tiles in the synthetic data. Different from ranked tiling [1], where implanted tiles only have high average values, we now implant tiles that have both low and high average values.

Data generation We use the generative model that we introduced in [1] to generate continuous data. First, we generate background data whose values are sampled from normal distributions having mid-ranged mean values. Second, we implant a number of constant-row tiles whose values are sampled from normal distributions having low/high mean values. Finally, we perform a complete ranking of columns in every row to obtain a rank matrix.

Formally, background data is generated by this generative model:

$$\forall r \in \mathcal{R}, \forall c \in \mathcal{C},\ \mathbf{M}_{r,c} \sim \begin{cases} N(\mu_r^1, 1) & \text{if } x_1 = 1 \\ N(\mu_r^2, 1) & \text{if } x_2 = 1 \\ N(\mu_r^3, 1) & \text{if } x_3 = 1 \end{cases} \tag{21}$$

where $\mu_r^1 \sim U(3,5)$, $\mu_r^2 \sim U(-5,-3), \mu_r^3 \sim U(-3,3)$; $x = (x_1, x_2, x_3), x_i \in \{0,1\}, \sum_i x_i = 1, x$ has mass probability function $\mu = (p, p, 1-2p), 0 \leq p \leq 0.5$.

A constant-row tile $\mathbf{M}_{R,C}$ having high average values is generated as:

$$\forall r \in R, \quad \mu_r \sim U(3,5) \tag{22}$$

$$\forall r \in R, \forall c \in C, \quad \mathbf{M}_{r,c} \sim N(\mu_r, 1) \tag{23}$$

[2] `http://www.gurobi.com/`

Tiles having low average values are generated in a similar way. However their mean values are sampled from a different uniform distribution: $U(-5, -3)$.

Setup We generate four 500 rows $\times$ 100 columns datasets for different p, i.e., $p \in \{0.05, 0.10, 0.15, 0.20\}$. In each dataset, we implant seven constant-row tiles. Three tiles have low average values, the other four have high average values.

We evaluate the ability of the algorithms to recover the implanted set of tiles. We do this by measuring *recall* and *precision*, using the implanted tiles as ground truth. Overall performance is quantified by the *F1* measure, which is the average of the two scores.

Varying the parameters We varied the parameters k and α in Equation 8 and then applied the Sparse RMF algorithm on the four synthetic datasets. For each parameter combination, the algorithm was executed ten times and the result maximising the score was used. This is to get rid of effects that are due to differences in the initialization based on K-means.

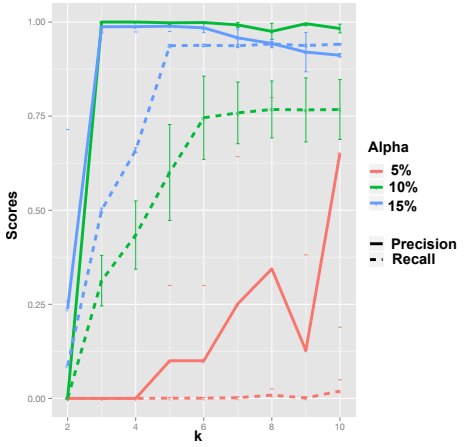


Fig. 2: Average precision and recall on the four synthetic datasets with varying parameters α and k.

Precision and recall are calculated based on the union of the coverage area, which has non-zero values, by the k components $\mathbf{C}(:, i)\mathbf{F}(i, :), i = 1 \ldots k$. The average performance of the algorithm on the four datasets is summarised in Figure 2. The figure shows that the Sparse RMF can recover implanted tiles and remove noise outside when $k = 5$ and $\alpha \sim 15\%$. When α is too small, i.e., 5%, the algorithm cannot recover the tiles. In general, the algorithm has high performances when k is large. This matches our expectation though, since a higher α results in higher tolerance to noise and a larger k results in more patterns and hence a more detailed description of the data.

Comparison to other algorithms In this experiment, we compare our approach to ranked tiling [1] and several bi-clustering algorithms. These include CC [10], Spectral [11], Plaid [12], FABIA[3] [13], SAMBA[4] [14]and ISA[5] [15]. CC, Spectral and Plaid are part of the R biclust[6] package.

Since large noise levels may conversely affect the performance of the algorithms, we use a dataset also used for the previous experiments, with $p = 0.05$ (low noise level). We ran all algorithms on this dataset and took the first seven

[3] `http://www.bioinf.jku.at/software/fabia/fabia.html`
[4] `http://acgt.cs.tau.ac.il/expander/`
[5] `http://cran.r-project.org/web/packages/isa2/`
[6] `http://cran.r-project.org/web/packages/biclust/`

Table 1: Comparison to ranked tiling and bi-clustering. Precision, recall and F1 quantify how accurately the methods recover the seven implanted tiles.

Algorithm	Data type	Pattern type	Prec.	Recall	F1
Sparse RMF	Ranks	Rank profile	99%	94%	96%
Ranked tiling [1]	Ranks	Ranked tile	37%	41%	39%
CoreNode [16]	Numerical	Coherent values bicluster	30%	8%	19%
FABIA [13]	Numerical	Coherent values bicluster	99%	51%	75%
Plaid [12]	Numerical	Coherent values bicluster	91%	46%	67%
SAMBA [14]	Numerical	Coherent evolution bicluster	52%	9%	31%
ISA [15]	Numerical	Coherent values bicluster	43%	17%	30%
CC [10]	Numerical	Coherent values bicluster	7%	5%	6%
Spectral [11]	Numerical	Coherent values bicluster	-	-	-

tiles/bi-clusters they produced, which have the highest scores (SAMBA) or largest sizes (all other). For most of the benchmarked algorithms, we used their default values. For CoreNode, we use $msr = 1.0$ and $overlap = 0.5$. For ISA, we applied its built-in normalised method before running the algorithm itself.

The results in Table 1 show that our algorithm achieves much higher precision and recall on this task than any of the ranked tiling and bi-clustering methods. Note that Spectral could not find any patterns. Ranked tiling only finds highly ranked tiles, whereas our rank matrix factorisation is more general and allows to capture any recurrent partial rankings in the rank matrix. Some of the bi-clustering methods attain quite high precision, e.g., FABIA and Plaid, but their recall is much lower than for Sparse RMF. The reason is that the synthetic data contains incomparable rows, with values on different scales. These results confirm that converting such data to rank matrices is likely to lead to better results.

6 Real world experiments

This section presents results on three real world datasets: 1) Eurovision Song Contest voting data, 2) Sushi preferences, and 3) NBA basketball team rankings.

We previously collected the European Song Contest (ESC) dataset [1]. This dataset contains aggregated voting scores that participating countries gave to competing countries during the period from 2010 to 2013. We aggregated the data by calculating average scores that voting countries award to competing countries and transformed it to ranked data. The NBA basketball team ranking dataset was collected by the authors of [17]. It consists of rankings of 30 NBA basketball teams by a group of professional agencies and groups of students. The Sushi dataset was collected by the authors of [18]. It contains preferences of five thousand people over ten different sushi types.

We initially applied sparse rank matrix factorisation on these datasets with varying α and k. Based on these preliminary experiments, we used the following

Table 2: Dataset properties, parameter settings, and performance statistics of Sparse RMF on three datasets.

Dataset	EU Song Contests [1]	NBA team rankings [17]	Sushi dataset [18]
Size	44x37	34x30	5000x10
0s in data	40%	60.4%	0%
#runs	200	200	10
k	10	9	8
α	10%	5%	20%
Coverage	30%	86%	78.2%
Average error	1.59	1.0	1.31
0s/pattern	59.7%	12.6%	13.8%
Overlapping	2%	0%	0%
Convergence	6.1±1.7	3.2±1.6	6.2±1.1
Time/run	3s	1.2s	53min

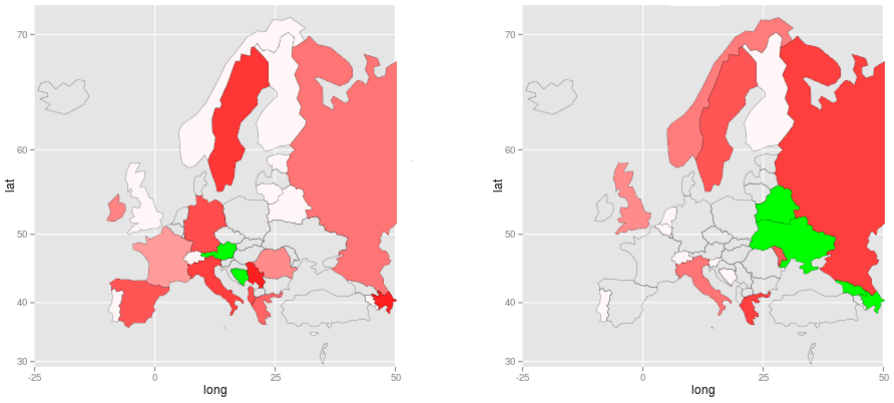
heuristics to choose reasonable parameter values to report on. We choose α such that it results in high coverage and low error. Given the chosen α, for k we choose the largest value such that each resulting pattern is used in at least two rows. When k is further increased, patterns are introduced that are used in only one row of the rank matrix, or even in none. This would clearly result in redundancy, which we would like to avoid.

Table 2 presents a summary of the results obtained by the Sparse RMF algorithm on all datasets. The upper five rows describe dataset properties and the used parameter values. For each dataset, the algorithm is executed a number of times (#runs) and the highest-scoring result is used for the remaining statistics (except for convergence and time/run, for which all runs are used).

The coverages and average errors (per covered cell) show that the algorithm can achieve high coverage with low error. With the Sushi dataset, for example, 78% of the matrix can be covered by just 8 rank profiles, and on average the ranks in the reconstructed rank matrix differs just 1.3 from those in the covered part of the matrix. The numbers of zeroes per pattern demonstrate that the algorithm successfully finds local patterns in the three studied datasets: partial rankings are used to cover the matrix. The overlapping statistic indicates that only the ESC dataset needs multiple patterns per row to cover a large part of the matrix: 2% of the rows are covered by more than one pattern.

Convergence indicates the average number of iterations in which the run converges, with standard deviation. This shows that the algorithm needs only few iterations to converge, typically 3 to 6. Finally, the average time per run shows that our algorithm runs very efficiently on modestly sized rank matrices, but making it more efficient for larger datasets is left for future work.

To show how rank patterns can provide insight into the data, we visualise two typical rank profiles obtained on the European Song Contest data in Figure 3. Both depict a set of voting countries (in green) and their typical voting behaviour



(a) Example 1 (b) Example 2

Fig. 3: Rank patterns discovered on the ESC dataset. Voters are painted green. Competitors are painted by their ranks: the darker the red, the higher the score.

(in red). For example, countries in Eastern Europe tend to give higher scores to Russia and nordic countries than to other countries.

7 Conclusions

We introduced the novel problem of rank matrix factorisation (RMF), which concerns the decomposition of rank data. RMF is a generic problem and we therefore introduced Sparse RMF, a concrete instance with the goal to discover a set of local patterns that capture the recurrent rankings in the data.

We formalised Sparse RMF as an optimisation problem and proposed a greedy, alternate optimisation algorithm to solve it using integer linear programming. Experiments on both synthetic and real datasets demonstrate that our proposed approach can successfully summarise rank matrices by a small number of rank profiles with high coverage and low error.

Acknowledgements. This research was supported by the DBOF 10/044 Project, the Natural and Artifical Genetic Variation in Microbes project, Post-doctoral Fellowships of the Research Foundation Flanders (FWO) for Siegfried Nijssen and Matthijs van Leeuwen, and the EU FET Open project Inductive Constraint Programming.

References

1. Le Van, T., van Leeuwen, M., Nijssen, S., Fierro, A.C., Marchal, K., De Raedt, L.: Ranked Tiling. In: Proc. of ECML/PKDD 2014 (2), Springer (2014) 98–113
2. Henzgen, S., Hüllermeier, E.: Mining rank data. In: Proc. of Discovery Science 2014, Springer (2014) 123–134
3. Paatero, P., Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. Environmetrics **5**(2) (1994) 111–126
4. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature **401**(6755) (October 1999) 788–791
5. Monson, S., Pullman, N., Rees, R.: A survey of clique and biclique coverings and factorizations of (0, 1)-matrices. Bull. Inst. Combinatorics and Its Applications **14** (1995) 17–86
6. Miettinen, P., Mielikainen, T., Gionis, A., Das, G., Mannila, H.: The discrete basis problem. IEEE Transactions on Knowledge and Data Engineering **20**(10) (2008) 1348–1362
7. Lust, T., Teghem, J.: Multiobjective decomposition of positive integer matrices: application to radiotherapy. In: EMO 2009, Springer (2009) 335–349
8. Marden, J.I.: Analyzing and Modeling Rank Data. Chapman & Hall (1995)
9. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: a survey. IEEE/ACM transactions on computational biology and bioinformatics **1**(1) (2004) 24–45
10. Cheng, Y., Church, G.M.: Biclustering of expression data. Proc. of the 8th International Conference on Intelligent Systems for Molecular Biology **8** (2000) 93–103
11. Kluger, Y., Basri, R., Chang, J.T., Gerstein, M.: Spectral Biclustering of Microarray Data : Coclustering Genes and Conditions. Genome Research **13** (2003) 703–716
12. Turner, H., Bailey, T., Krzanowski, W.: Improved biclustering of microarray data demonstrated through systematic performance tests. Computational Statistics & Data Analysis **48**(2) (February 2005) 235–254
13. Hochreiter, S., Bodenhofer, U., Heusel, M., Mayr, A., Mitterecker, A., Kasim, A., Khamiakova, T., Van Sanden, S., Lin, D., Talloen, W., Bijnens, L., Göhlmann, H.W.H., Shkedy, Z., Clevert, D.A.: FABIA: factor analysis for bicluster acquisition. Bioinformatics **26**(12) (June 2010) 1520–7
14. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. Bioinformatics **18**(Suppl. 1) (2002) S136–S144
15. Ihmels, J., Friedlander, G., Bergmann, S., Sarig, O., Ziv, Y., Barkai, N.: Revealing modular organization in the yeast transcriptional network. Nature genetics **31**(4) (August 2002) 370–7
16. Truong, D.T., Battiti, R., Brunato, M.: Discovering Non-redundant Overlapping Biclusters on Gene Expression Data. In: ICDM 2013, IEEE (2013) 747–756
17. Deng, K., Han, S., Li, K.J., Liu, J.S.: Bayesian Aggregation of Order-Based Rank Data. Journal of the American Statistical Association **109**(507) (October 2014) 1023–1039
18. Kamishima, T.: Nantonac collaborative filtering: Recommendation based on order responses. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '03, New York, NY, USA, ACM (2003) 583–588