Ranked Tiling

Thanh Le Van[1], Matthijs van Leeuwen[1], Siegfried Nijssen[1,2],
Ana Carolina Fierro[3], Kathleen Marchal[3,4,5], and Luc De Raedt[1]

[1] Department of Computer Science, KU Leuven, Belgium
`firstname.lastname@cs.kuleuven.be`
[2] Leiden Institute for Advanced Computer Science, Universiteit Leiden,
The Netherlands
[3] Department of Microbial and Molecular Systems, KU Leuven, Belgium
`carolina.fierro@biw.kuleuven.be`
[4] Department of Plant Biotechnology and Bioinformatics, Ghent University, Belgium
[5] Department of Information Technology, iMinds, Ghent University, Belgium
`kathleen.marchal@ugent.be`

Abstract. Tiling is a well-known pattern mining technique. Traditionally, it discovers large areas of ones in binary databases or matrices, where an area is defined by a set of rows and a set of columns. In this paper, we introduce the novel problem of *ranked* tiling, which is concerned with finding interesting areas in ranked data. In this data, each transaction defines a complete ranking of the columns. Ranked data occurs naturally in applications like sports or other competitions. It is also a useful abstraction when dealing with numeric data in which the rows are incomparable.

We introduce a scoring function for ranked tiling, as well as an algorithm using constraint programming and optimization principles. We empirically evaluate the approach on both synthetic and real-life datasets, and demonstrate the applicability of the framework in several case studies. One case study involves a heterogeneous dataset concerning the discovery of biomarkers for different subtypes of breast cancer patients. An analysis of the tiles by a domain expert shows that our approach can lead to the discovery of novel insights.

Keywords: tiling, ranked data, numerical data, pattern mining.

1 Introduction

The problem of tiling was introduced by Geerts et al. [1]. It is a popular pattern mining technique that searches for a set of tiles (that is, a *tiling*) in a 0/1 matrix. Such matrices often represent transactional data, where each transaction specifies the presence or absence of a set of items in the transaction. A tile is then a subset of the rows and columns of the matrix, for which the corresponding submatrix contains all 1s. Tilings are interesting as they provide groupings of both the rows and the columns that may give new insights in the data.

In this paper, we extend tiling towards a setting which is not binary. That is, we introduce the problem of *ranked tiling*, in which each transaction in the

T. Calders et al. (Eds.): ECML PKDD 2014, Part II, LNCS 8725, pp. 98–113, 2014.

data is a ranking of all available items. This type of data naturally occurs in many situations of interest. Consider, for instance, cycling competitions where the items could be the cyclists and each transaction would correspond to a race, or consider a business context, where the items could be companies and the transactions specify the rank of their quotation for a particular service. Ranking is also a natural abstraction for purely numeric data, which often arises in practice and may be noisy or imprecise. Numeric data is hard to analyse with many existing pattern mining approaches.

One real-life example that we shall use is concerned with the discovery of biomarkers to group cancer patients into *subtypes*. Finding a set of biomarkers that characterise different cancer subtypes is clearly important. Thanks to advances in genome sequencing and high throughput technologies, a lot of data is becoming available about patients. However, different types of data may be obtained with different technologies, for example, data concerning mRNA, miRNA, copy number variations, or proteins. This means that we are given a number of data matrices, each of which corresponds to one data type or experiment, and each of which is measured on a different scale. Still we would expect to find a tiling in which the same set of patients is shared across the different data matrices. We shall show that by using a ranked version of the data and ranked tiling, this is feasible.

To illustrate the problem of ranked tiling, let us consider the toy example in Figure 1. It depicts a rank matrix containing five rows and ten columns. Assuming no ties, each row contains each of the numbers one to ten exactly once. In this paper, we assume that a desirable high rank is indicated by a high number, i.e., in this case the highest possible rank is ten. Now, we are intuitively interested in rectangular areas in the matrix that have relatively high values, as these correspond to columns and rows which are highly ranked. In this particular example, the maximal ranked tile that we would like to find consists of five rows and three columns, i.e., the area defined by $\{R1, R2, R3, R5\}$ and $\{C1, C2, C3\}$.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
R1	8	10	9	4	2	6	1	7	3	5
R2	7	10	9	6	3	8	4	5	2	1
R3	6	7	9	5	8	4	1	2	10	3
R4	2	9	1	7	4	5	10	6	8	3
R5	9	5	8	3	7	1	10	4	2	6

Fig. 1. Example rank matrix, with maximal ranked tile $B = (\{R1, R2, R3, R5\}, \{C1, C2, C3\})$

The key contributions of our paper are 1) the introduction of the problem of ranked tiling, 2) the introduction of an optimisation model for ranked tiling and its implementation in a constraint programming solver, and 3) an empirical evaluation on synthetic and real-life datasets that shows the ability of our ranked tiler to discover interesting tiles, and shows the promise of the approach in practical discovery tasks such as that concerned with breast cancer.

2 Ranked Tiling

In this section, we formally define ranked data and introduce the problems that we consider: maximal ranked tile mining and ranked tiling.

Definition 1 (Rank matrix). *Let $\mathcal{M}$ 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 $\mathcal{M}$ is a rank matrix iff:*

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

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

Note that the values of a row can only be a (strict) subset of σ iff the row contains ties, otherwise the set of values must be exactly equal to σ. Given such a rank matrix, we would like to find a maximal ranked tile, i.e., a tile with relatively highly ranked values. Formally, we have the following problem.

Problem 1 (Maximal ranked tile mining) *Given a rank matrix $\mathcal{M} \in \sigma^{m \times n}$ and a threshold θ, find the ranked tile $B = (R^*, C^*)$, with $R^* \subseteq \mathcal{R}$ and $C^* \subseteq \mathcal{C}$, such that:*

$$B = (R^*, C^*) = \underset{R,C}{\operatorname{argmax}} \sum_{r \in R, c \in C} (\mathcal{M}_{r,c} - \theta). \tag{2}$$

where θ is an absolute-valued threshold.

Example 1. Going back to the example rank matrix in Figure 1 and choosing $\theta = 5$, the maximal ranked tile is defined by $R = \{1, 2, 3, 5\}$ and $C = \{1, 2, 3\}$. The score obtained by this tile is 37, and no more columns or rows can be added without decreasing the score. This result matches the desired outcome that we described in the introduction.

In practice, we often use a relative instead of an absolute threshold. We denote such a threshold as a percentage, i.e., $\theta = a\%$ implies $\theta = a\% \times n$.

The optimisation objective in Equation 2 rewards cells in a tile having values higher than θ, and vice versa for cells having lower values. Since we look for tiles that maximise this score, this threshold θ plays an important role. That is, higher values for θ result in smaller tiles with larger ranks. This implies that the threshold can be used to influence both 1) the size of the mined tiles, and 2) the extent to which the ranks deviate from the mean rank. An alternative interpretation is that the threshold can be used by the analyst to express her prior belief about how high the ranks should be to make a tile interesting.

In practice, it happens quite often that we have numerical data, either discrete or continuous, that we would like to analyse. In gene expression analysis, for example, we are given a matrix with continuous data, whose columns represent patients and rows represent genes. A value in a cell is then the expression level of the gene for a specific patient.

Fortunately, converting a matrix with numeric data to a rank matrix is straightforward. Given any ranking function, i.e., a function that sorts a set of values and assigns ranks based on the resulting order, each row of a numeric matrix can be transformed. In practice, ties may occur: the same value may occur more than once within a single row. Such ties can be broken using either average rank scores or minimum rank scores (in this paper, we use the former unless noted otherwise).

The maximal ranked tiling problem aims to find a single tile, but we are of course interested in finding a set of such tiles. In other words, we would like to discover a ranked tiling.

Problem 2 (Ranked tiling) *Given a rank matrix $\mathcal{M}$, a number k, a threshold θ, and a penalty term P, the ranked tiling problem is to find a set of ranked tiles $B_i = (R_i, C_i)$, $i = 1 \ldots k$, such that they together maximise the following objective function:*

$$\operatorname*{arg\,max}_{R_i, C_i} \sum_{r \in \mathcal{R}, c \in \mathcal{C}} \mathbb{1}_{(t_{r,c} \geq 1)}((\mathcal{M}_{r,c} - \theta) - (t_{r,c} - 1)P) \tag{3}$$

where $t_{r,c} = |\{i \in \{1, \ldots, k\} \mid r \in R_i, c \in C_i\}|$ indicates the number of tiles that cover a cell, and $\mathbb{1}_\varphi$ is an indicator function that returns 1 if the test φ is true, and 0 otherwise. P indicates a penalty that is assigned when tiles overlap.

For the remainder of this paper we will fix P to $\theta - 1$, i.e., we penalise overlap with the lowest possible rank minus the score θ.

Before we describe our approach to solving the aforementioned problems, we introduce two variations that allow the data analyst to do *query-based* tiling. By allowing the data analyst to provide queries to the system, it becomes possible to explicitly search for interesting patterns in specific areas of the data.

Problem 3 (Maximal query-based ranked tile mining) *Given a query Q, a rank matrix $\mathcal{M} \in \sigma^{m \times n}$, and a threshold θ, find the maximal ranked tile that satisfies the following additional constraint:*

$$\forall q \in Q : q \in C \tag{4}$$

In other words, a query-based tile is a maximal ranked tile whose columns contains those specified in the user-defined query.

Finally, an alternative method can be used to avoid overlap between tiles.

Problem 4 (Diverse ranked tiling) *Given a number k, the problem of diverse ranked tiling is to find a set of k ranked tiles, $B_i = (R_i, C_i), i = 1 \ldots k$, such that the following objective function is optimised:*

$$\operatorname*{arg\,max}_{R_i, C_i} \sum_{r \in \mathcal{R}, c \in \mathcal{C}} \mathbb{1}_{(t_{r,c} \geq 1)}(\mathcal{M}_{r,c} - \theta), \tag{5}$$

under the constraint that:

$$R_i \cap R_j = \varnothing, \forall i \neq j, i, j = 1 \ldots k, \tag{6}$$

i.e., no two tiles share the same row.

The diverse ranked tiling and maximal query-based ranked tile mining problems can be combined to find a *diverse query-based ranked tiling.*

3 Ranked Tiling Using Constraint Programming

In this section, we present the techniques that we propose to solve the problems introduced in the previous section. That is, we introduce a constraint-based model equivalent to Equation 2, but add two constraints to make solving more efficient without affecting the results.

The technique we use to solve the optimisation problem is constraint programming (CP). We follow the approach that was originally introduced by De Raedt et al. [2]. The idea is to formalise the problems as *constraint satisfaction problems* and then use existing solvers to find solutions. There are a number of advantages to this solving paradigm. First, it is a declarative approach, meaning that the data analyst can focus on modelling the problem rather than on complex procedural implementations. Second, CP is very flexible. As we will demonstrate, it is easy to implement small variations of a problem by adding or modifying constraints.

3.1 Constraint-Based Model

To speed up the search process, we add two redundant constraints to the optimisation problem of Equation 2. That is, we require that the average values in rows and columns in the selected submatrix $\mathcal{M}_{R,C}$ are higher than the threshold θ. The resulting constraint-based model is as follows:

$$\operatorname{argmax}_{R,C} \textstyle\sum_{r\in R,c\in C}(\mathcal{M}_{r,c}-\theta) \tag{7}$$

subject to

$$\forall r \in \mathcal{R} : r \in R \leftrightarrow \quad \frac{\sum_{c\in C}\mathcal{M}_{r,c}}{|C|} \geq \theta \tag{8}$$

$$\forall c \in \mathcal{C} : c \in C \leftrightarrow \quad \frac{\sum_{r\in R}\mathcal{M}_{r,c}}{|R|} \geq \theta \tag{9}$$

Theorem 1 (Model equivalence). *The constraint-based optimisation model in Equations 7–9 is equivalent to the optimisation model in Equation 2.*

Proof. The proof is given in the Appendix.

3.2 Problem Formalisation Using CP

The formalisation in Equations 7 – 9 is defined over set variables. Unfortunately, in earlier work it was shown [2] that set variables do not lead to good performance in CP and a reformalization in terms of boolean variables is necessary.

We therefore introduce two Boolean decision vectors: $T = (T_1, T_2, ..., T_m)$, with $T_i \in \{0, 1\}$, for rows and $I = (I_1, I_2, ..., I_n)$, with $I_i \in \{0, 1\}$, for columns. An assignment to the Boolean vectors T and I corresponds to an indication of rows and columns belonging to a tile. Given this, we have the following theorem.

Theorem 2 (Highly ranked rows constraint). *If the following constraint is satisfied:*

$$\forall t \in \mathcal{R} : T_t = 1 \leftrightarrow \sum_{i \in \mathcal{C}} (\mathcal{M}_{t,i} - \theta) * I_i \geq 0 \tag{10}$$

then rows that satisfy the inequality of Equation 8 are identified by:

$$\{r \in \mathcal{R} \mid T_r = 1\} \tag{11}$$

Proof.

$$\begin{aligned}\forall t \in \mathcal{R} : T_t = 1 \leftrightarrow \sum_{i \in \mathcal{C}} (\mathcal{M}_{t,i} - \theta) * I_i \geq 0 \leftrightarrow \sum_{i \in \mathcal{C}} \mathcal{M}_{t,i} * I_i \geq \theta * \sum_{i \in \mathcal{C}} I_i \\ \leftrightarrow \frac{\sum_{i \in \mathcal{C}} \mathcal{M}_{t,i} * I_i}{\sum_{i \in \mathcal{C}} I_i} \geq \theta \quad \leftrightarrow \frac{\sum_{i \in C} \mathcal{M}_{i,c}}{|C|} \geq \theta\end{aligned}$$

Here we assume that $\sum_{i \in \mathcal{C}} I_i \geq 1$. Overall, $T_t = 1 \leftrightarrow t \in R$, which concludes the proof.

A similar property can be obtained for the column constraint. This leads to a CP model which is equivalent to the constraint-based model in Equations 7 – 9:

$$\operatorname*{argmax}_{T,I} \sum_{t \in \mathcal{R}} T_t * (\sum_{i \in \mathcal{C}} (\mathcal{M}_{t,i} - \theta) * I_i) \tag{12}$$

subject to

$$\forall t \in \mathcal{R} : T_t = 1 \leftrightarrow \sum_{i \in \mathcal{C}} (\mathcal{M}_{t,i} - \theta) * I_i \geq 0 \tag{13}$$

$$\forall i \in \mathcal{C} : I_i = 1 \leftrightarrow \sum_{t \in \mathcal{R}} (\mathcal{M}_{t,i} - \theta) * T_t \geq 0 \tag{14}$$

The constraints in this formalisation are similar to those used to mine frequent itemsets in [2]. The main difference is that we also have an objective function.

3.3 Mining Maximal Ranked Tiles Using CP

Mining a single ranked tile is equivalent to finding an assignment to vectors T, I such that T and I satisfy constraints 13 – 14 and maximise objective function 12. We solve this constrained optimisation problem using constraint programming.

Solving a problem using CP is done in two phases: 1) modelling, and 2) solving. Equations 12 – 14 can be written down as a model in any CP solver. As we will see, however, the problem of ranked tiling is not easy to solve, and finding exact solutions is difficult. To allow for finding approximate solutions, we choose the OscaR[1] solver, which is an open source CP solver written in Scala. A distinguishing feature is that it provides good support for both exhaustive and heuristic search methods, which we will use for our approach.

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

When the CP solver is asked to perform exhaustive search, it essentially builds a search tree. The key idea here is that it uses the constraints to remove inconsistent values while searching. This removal of inconsistent values is called propagation and can reduce the search space significantly.

Variable and value ordering heuristic. The order in which variables are considered for branching, as well as the order in which values are assigned to the variables, determine the shape of the search tree and the effectiveness of constraint propagation. We use the following heuristic. We order column variables $I_c,\ c = 1 \dots n$, by their total sum scores, $\sum_{r \in \mathcal{R}} (\mathcal{M}_{r,c} - \theta)$, in ascending order. That is, variables that have lower scores will be branched on first, and the value zero will be assigned to a variable before the value one. Using this heuristic, CP has a higher chance to add variables having high scores to the solution when backtracking. Consequently, CP needs less backtracks to find a first valid assignment as variables having higher scores have higher probability of satisfying constraint 14.

Large neighbourhood search. Equation 13 shows that vector T can be completely determined given a complete assignment to I. Hence, the size of the search space is $O(2^n)$, where n is the number of columns. Even taking into account propagation, this search space is in practice often still too large to be traversed completely and hence we use a form of local search to speed up the search.

Large Neighbourhood Search (LNS) is a hybridization of local search and exhaustive search in CP. Local search refers to the idea that one solution can be transformed into another by changing the assignment of a number of variables. While traditional local search methods only change a limited number of variables (e.g., one variable at a time), LNS selects a relatively large subset of the variables in a problem (e.g., a random subset of half of the variables) and performs complete search over these variables while fixing the remaining variables. Two main questions involved with LNS include a) which variables should be selected to search over, and b) how to search on these variables? In our implementation, we use stochastic variable selection and an exhaustive search approach. The stochastic variable selection uniformly selects half of the column variables to search over.

3.4 Ranked Tiling

Ranked tiling was introduced in Problem 2. We propose to approximate the optimal solution by using a greedy approach, as is common for this type of pattern set mining problem. The first tile is found by solving the optimisation problem in Equations 12–14. Next, we remove that tile by setting all cells in the matrix that are covered to the lowest rank (or another value, depending on parameter P). Then, we search in the resulting matrix for the second tile. This process is repeated until the number of desired tiles is found. The sum of the scores of all discovered tiles will correspond to the score of Equation 3 for this solution. However, as the search is greedy, the solution is not necessarily optimal.

4 Experiments on Synthetic Datasets

In this section, we set up a number of experiments on synthetic data sets to evaluate the proposed approach. The results will show: 1) the gain in runtime by adding the redundant constraints; 2) the accuracy of the discovered tiles in two settings: ranked tiling and query-based ranked tiling; 3) the robustness of the algorithm with respect to noise thresholds and variations of the local search; 4) a comparison to constant-row bi-clustering algorithms.

4.1 Data Generation

By generating data in which rows are incomparable due to different scales, our aim is to show that our technique finds the relevant patterns in such data, whereas methods for bi-clustering cannot. Since bi-clustering methods only work on numeric data, we use a simple generative model to generate synthetic, continuous data. This numeric data is then transformed to a rank matrix to apply ranked tiling. For bi-clustering, we choose the constant-row setting, as there are many bi-clustering algorithms specifically designed for this type of pattern.

To generate synthetic datasets, we first generate background data, and then implant a number of constant-row bi-clusters with higher average values.

Background information is generated such that each row has a potentially different scale than the others. Values within each row are sampled from two distributions, with certain probability: one for modelling background noise, the other for interfering with the implanted patterns. First, for each row, we uniformly sample μ_r^1, μ_r^2 from two ranges:

$$\mu_r^1 \sim U(0,3), \forall r \in \mathcal{R} \tag{15}$$

$$\mu_r^2 \sim U(3,5), \forall r \in \mathcal{R} \tag{16}$$

Second, for every cell in a row, we sample a latent binary variable $X_{r,c}$ from a Bernoulli distribution $Bin(p, 1-p)$, given some p. Depending on the value of this latent variable, the background data is sampled either from the low-average or high-average distribution:

$$\mathcal{D}_{r,c} \sim \begin{cases} N(\mu_r^1, 1) & \text{if } X_{r,c} = 1 \\ N(\mu_r^2, 1) & \text{otherwise} \end{cases} \tag{17}$$

To plant a constant-row bi-cluster in a submatrix $\mathcal{D}_{R,C}$, we use the following two equations:

$$\forall r \in R, \quad \mu_r \sim U(3,5) \tag{18}$$

$$\forall r \in R, \quad \mathcal{D}_{r,c} \sim N(\mu_r, 1) \tag{19}$$

Equation 18 is used to sample the mean value for every row in a bi-cluster. The mean value is uniformly sampled from the range $[3 \ldots 5]$, which is higher than the main sampling range for the background ($[0 \ldots 3]$).

4.2 Evaluation

Performance Gain by Adding Constraints. To evaluate how the redundant constraints affect the time needed for search, we compare the runtime of the optimisation model in Equation 2, which does not have any constraints, to the runtime of the constraint-based model shown in Equations 7–9. To make the comparison fair, we perform exhaustive search in both cases.

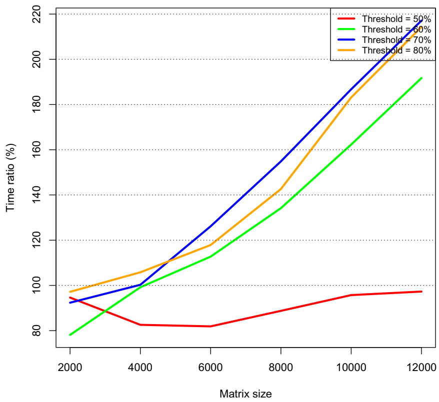


Fig. 2. Speed up achieved by adding the two redundant constraints. The y-axis indicates the ratio between the runtime needed with and without the constraints, and is computed for varying threshold θ values and matrix sizes.

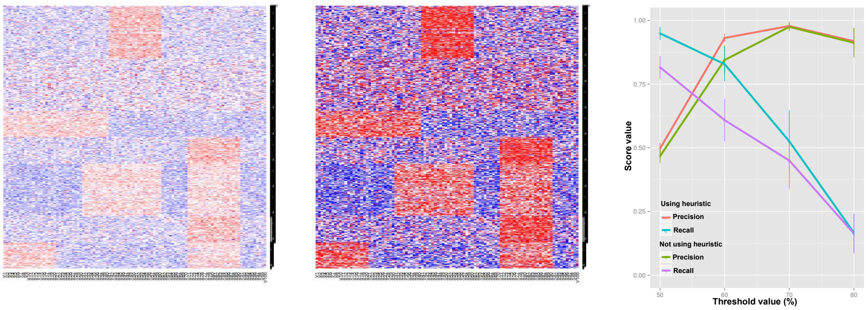
To this aim, we generate a number of datasets with varying sizes using the procedure described in Section 4.1. All datasets have the same number of columns, i.e. 20, and a varying number of rows: $\{100, 200, 300, 400, 500, 600\}$, with $p = 0.1$. They have the same two non-overlapping tiles: one is 10×5 and the other is 30×10. We execute the mentioned models on these datasets to find the maximal ranked tile. All experiments are executed single threaded on a desktop computer (Intel i7-2600 CPU @ 3.40GHz, 16GB RAM).

Figure 2 shows the ratio between the time needed to solve the problem with and without the extra constraints, for varying θ thresholds. We can see that adding the extra constraints reduces the search time when $\theta > 50\%$. In particular for larger datasets and values of the threshold, the model with the added constraints always outperforms the optimisation-only model. This demonstrates that adding the constraints results in better propagation and hence more efficient search. The absolute time needed to find the optimal solution on these datasets ranges from 1ms to 4h 37m 26s.

Accuracy of the recovered tiles. We now evaluate the ability of the algorithm to recover the implanted ranked 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.

We generate seven 1000 rows $\times$ 100 columns datasets for different p, i.e., $p \in \{0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40\}$, using the same procedure as before. In each dataset, we implant five ranked tiles. Figure 3a shows a heatmap of the numerical dataset for $p = 10\%$. Figure 3b shows its corresponding rank matrix, using the same color coding as in Figure 1 (blue = low rank, red = high rank).

We varied the threshold θ; for each value of θ and each dataset, we performed ranked tiling five times, each time mining $k = 5$ tiles. Then, we calculated average precision, recall and F1 score over these five runs. Figure 3c summarises the results obtained with and without using the variable ordering heuristic (see Section 3.3). It can be seen that the heuristic contributes to improved performance.



(a) Numerical data **(b)** Rank matrix **(c)** Precision and recall

Fig. 3. Recovering five implanted ranked tiles from synthetic data

When the threshold θ is around 60%, the algorithm achieves high prediction accuracy (average F1 = 86%). At lower thresholds, it has low precision, while higher thresholds result in lower recall. This completely matches our expectation, since higher thresholds result in smaller tiles with higher values.

Robustness of the local search. The error bars in the precision and recall curves shown in the Figure 3c show the robustness of the large neighbourhood search on the synthetic datasets. The variation is typically low with respect to the number of times the local search is repeated and the different noise levels.

Comparison to bi-clustering. In this experiment, we compare to several bi-clustering algorithms. SAMBA [3] was designed for coherent evolution bi-clusters, in which there is coherence of the signs of values, i.e., up or down. The other methods discover coherent-valued bi-clusters, of which a constant-row bi-cluster is a special case. CC [4], Spectral [5], and Plaid [6] are implemented in the R biclust[2] package. FABIA[3] [7] and SAMBA[4] are downloaded from their website. ISA [8] is from the R isa2 package[5].

Since large noise levels may conversely affect the performance of the algorithms, we use a dataset also used for the previous experiments, with $p = 0.20$ (average noise level). We ran all algorithms on this dataset and took the first five tiles/bi-clusters they produced. For most of the benchmarked algorithms, we used their default values. For CoreNode, we use $msr = 1.0$ and $overlap = 0.5$, as preliminary experiments showed that this combination produced the best result. For ISA, we applied its built-in normalised method before running the algorithm.

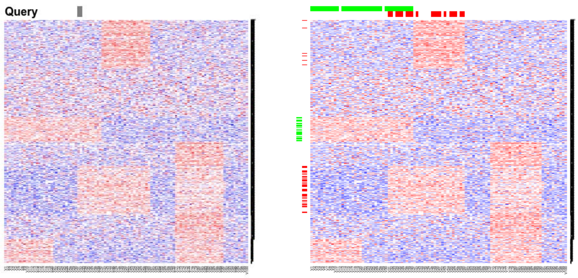
The results in Table 1 show that our algorithm achieves much higher precision and recall on this task than the bi-clustering methods. This indicates that when the rows in a numerical matrix are incomparable, converting the data to a ranked matrix and applying ranked tiling is a better solution than applying bi-clustering.

---

[2] `http://cran.r-project.org/web/packages/biclust/`
[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/`

Table 1. Comparison to bi-clustering. Precision, recall and F1 quantify how accurately the methods recover the five implanted tiles. $k = 5$.

Algorithm	Data type	Pattern	Precision	Recall	F1
Our algorithm	Ranks	Ranked tile	88%	83%	86%
CoreNode [9]	Numerical	Coherent values bicluster	43%	72%	58%
FABIA [7]	Numerical	Coherent values bicluster	40%	24%	32%
Plaid [6]	Numerical	Coherent values bicluster	90%	6%	48%
SAMBA [3]	Numerical	Coherent evolution bicluster	67%	3%	35%
ISA [8]	Numerical	Coherent values bicluster	64%	44%	54%
CC [4]	Numerical	Coherent values bicluster	35%	22%	29%
Spectral [5]	Numerical	Coherent values bicluster	-	-	-



(a) A query of 2 columns. **(b)** The two mined tiles.

Fig. 4. Query-based ranked tiling on a synthetic dataset; $\theta = 70\%$

Diverse query-based ranked tiling Finally, we use the same dataset, with $p = 0.2$, to illustrate diverse query-based ranked tiling. Figure 4a shows a query consisting of two columns, and Figure 4b shows the two discovered ranked tiles given that query. The rows and columns of each tile are marked by coloured bars (tile 1 = red, tile 2 = green). The results demonstrate the flexibility of our approach based on constraint programming: adding a few constraints results in ranked tiles for a given query. Both discovered tiles contain the query, but are also clearly present in the generated data. This allows the analyst to focus the search on specific areas of the data and still obtain high-quality results.

5 Real-world Case Studies

In this section, we present two applications of ranked tiling: 1) voting pattern discovery in Eurovision Song Contest data, and 2) biomarker discovery for different subtypes from a heterogeneous genomic dataset.

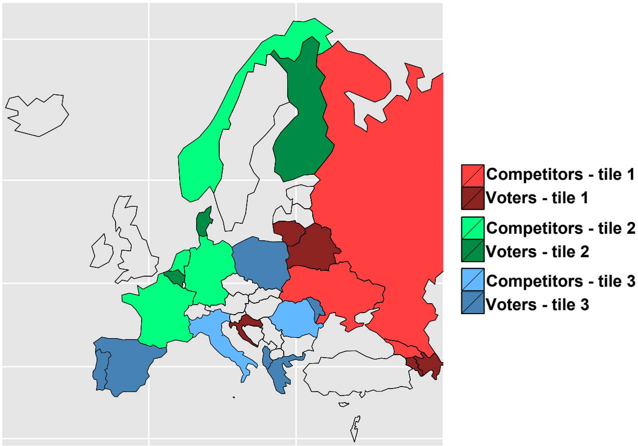


Fig. 5. Voting patterns in the Eurovision Song Contest data

5.1 European Song Contests

The Eurovision Song Contest (ESC) has been held annually since 1956. Each participating country gives voting scores, which are a combination of televoting and jury voting, to competing countries. Scores are in the range of $1 \dots 8$, 10 and 12. Each country awards 12 points to their most favourite country, 10 points to the second favourite, and $8 \dots 1$ to the third…tenth favourites respectively. The data can be represented by a matrix, in which rows correspond to voting countries, columns correspond to competing countries, and values are the scores.

We collected voting data for the final rounds of the period $2010 - 2013$. We filtered out countries participating in fewer than 3 years. Data is aggregated by calculating average scores that voting countries award to competing countries during the period considered. After the pre-processing step, the data consists of 44 rows and 37 columns. The pre-processed data is then transformed to ranked data, using minimum ranks in case of ties.

We ran the algorithm on the ESC dataset and obtained 10 ranked tiles with the threshold θ set to 80%. The first tile shows that Azerbaijan and Sweden were highly voted for by other 19 countries located in many regions in Europe. Other tiles reveal that there are local voting patterns in the contests: countries tend to distribute high scores to their neighbours. Figure 5 shows three representative tiles: one for eastern countries, one for western and northern countries, and one for (mostly) southern countries. In general, the discovered tiles confirm that countries give high scores to their neighbours, which matches our expectations.

5.2 Biomarker Discovery for Breast Cancer Subtypes

Breast cancer is known to be a heterogeneous disease that can be categorised in clinical and molecular subtypes. Assignment of patients to such subtypes is crucial to give adapted treatments to patients. Currently, breast cancer patients are categorised into 3 clinical subtypes, which receive either endocrine therapy, targeted HER2 therapy, or chemotherapy. The study of tumour samples at multiple molecular levels helps to understand the driving events in cancer. Most studies, however, focus on the analysis of each molecular data type separately [10], since each data type is measured with a different technology. Although raw data between molecular levels are incomparable, their ranked values can easily be compared. The goal of this case study is to identify groups of genes or copy number regions that are highly specific to a subset of patients.

Data pre-processing. A breast cancer dataset was downloaded from the Cancer Genome Atlas[6]. A subset of 94 tumour samples with measurements at four molecular levels (mRNA, miRNA, protein levels, and copy number variation (CNV)) was selected. Each tumour sample is associated to a molecular subtype according to the PAM50 gene signature [11]. The original dataset contains measurements for 17814 genes (mRNA) and 1222 microRNAs. Genes and microRNAs were selected based on their potential subtype-specific activity and their differential expression relative to normal (non-tumour) samples.

To capture subtype-specific expression changes, we evaluated the 5- and 95-percentile of the tumour samples. Genes in which the p-value for these percentiles was below 0.001 and their log-fold change relative to the mean normal expression was at least 2.5 were selected. The final dataset contains 1761 genes and 138 microRNAs. Segmented files with copy number alterations were pre-processed and filtered by germline aberrations as described by [10]. Significant copy number regions were identified with GISTIC2.0 [12]. No selection was done for protein levels and copy number regions. Each data level was converted to rank scores and combined into a single matrix, which consists of 2211 rows and 94 columns.

Ranked tiling analysis. The ten ranked tiles discovered by the algorithm are shown in Figure 6 ($\theta = 65\%$, 20 LNS repeats, in 3h 45m 9s). A first important observation is that each tile contains features from all four different data types. All PAM50 subtypes (LuminalA, LuminalB, Basal and HER2 subtypes) are covered by the tiles. Some discrepancies between the tiles and PAM50 subtypes are expected, since PAM50 is based on a 50-genes classifier derived from expression data only. We find that tile 1 captures the largest group of genes and samples, but due to its large size functional annotation did not lead to interesting insights.

Tiles 2 and 3 match a known basal subtype, with tile 3 containing most basal samples. Among the genes specific for tile 3, we observe MYC at the protein and mRNA levels, which has been previously suggested as a basal characteristic pattern [10]. Both tiles 2 and 3 largely overlap in terms of genes also with tile 5 (LuminalB). The genes in common to these 3 tiles are enriched for cell cycle and cell division (Gene Ontology and Reactome enrichment), consistent with the high proliferation present in LuminalB, Basal and Her2 subtypes.

Tile 6 only contains samples from the known HER2 subtype. The HER2 molecular subtype is known to over-express the ERBB2 gene and to contain copy number alterations for the same gene. Tile 6 captures all the molecular levels related to the gene, as it contains the amplified region of ERBB2, the protein

[6] `https://tcga-data.nci.nih.gov/tcga`

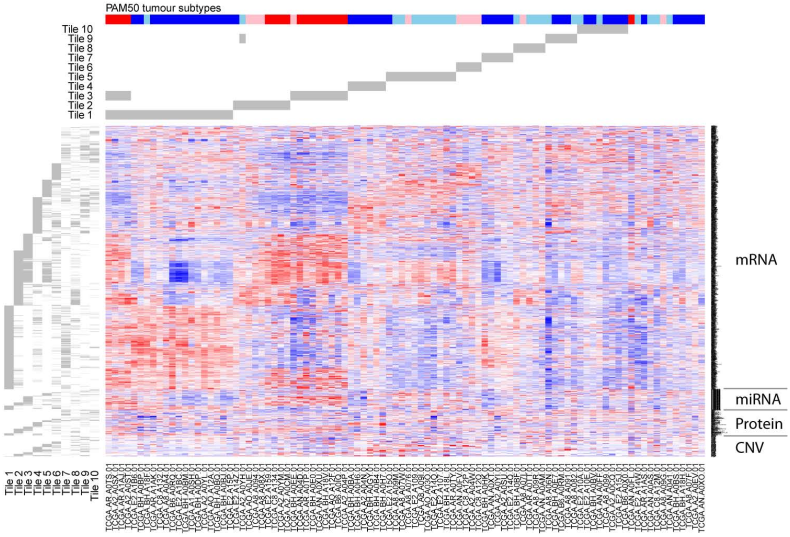


Fig. 6. Ranked tiling on heterogeneous breast cancer data. The rows correspond to mRNA, miRNA, protein and CNV levels, the columns correspond to breast cancer samples. The left and upper gray bars indicate the ten tiles. PAM50 subtypes are indicated at the top (LuminalA=blue, LuminalB=light blue, Basal=red and HER2=pink).

and the mRNA gene. LuminalA and LuminalB PAM50 subtypes correspond to the estrogen positive (ER+) clinical subtype which is the most common one and thus they are present in many tiles. Among them, tile 10 contains the estrogen receptor (ESR1) and other genes related to the pathway, suggesting that the pathway activity might be higher for the patients in tile 10.

Overall, we conclude that ranked tiling successfully identifies known subtypes and includes different data types in tiles, as desired. Such an integrated analysis of heterogeneous data has large potential for this type of application.

6 Related Work

Ranked tiling is related to bi-clustering, but is different because of the type of regularities it aims to find. The literature describes four types of bi-clusters: constant-valued, constant-row, constant-column and coherent [13]. In ranked tiling, the absolute values within the specified areas matter, i.e., the values must be higher than a given threshold. This is clearly different from the objectives of bi-clustering, as also demonstrated by the results presented in Section 4.

Calders et al. [14] use rank measures to find interesting patterns in numeric data. However, they rank values on individual columns and use frequent itemset mining-based techniques to find correlated sets of columns (items). Here, we consider ranks in rows and use optimisation-based techniques to find sets of rows and columns (tiles) in a matrix for which the ranks are relatively high.

Ranked tiling is also related to pattern mining in numeric data. In this direction, there has been work by Kaytoue et al. [15] that uses formal concept analysis to find interval patterns of itemsets. The recent work by Song et al. [16] proposes to mine association rules of numeric attributes. However, they did not consider ranked data and tilings on this type of data.

Kontonasios et al. [17] propose to use a Maximum Entropy model to iteratively mine interesting tiles in numeric data. This approach also aims to find sets of tiles whose content are contrasted to the background model. However, we do not contrast a tile against expected values given some prior beliefs, but consider absolute values relative to a given threshold. Apart from this, they do not provide an algorithm that directly searches for high-scoring tiles, while we propose a mining approach based on constraint programming.

7 Conclusions

We introduced the novel problem of ranked tiling, which is concerned with finding areas in ranked data in which the ranks are relatively high. Ranked data occurs naturally in many applications, but is also a useful abstraction when dealing with numeric data in which the rows are incomparable.

We presented an optimisation-based approach to solving the ranked tiling problem using constraint programming, and demonstrated the flexibility of this approach by extending it to query-based ranked tiling. The experiments on both synthetic and real data show that our approach finds high-quality ranked tiles that can lead to the discovery of novel insights in heterogeneous data.

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. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling Databases. In: Suzuki, E., Arikawa, S. (eds.) DS 2004. LNCS (LNAI), vol. 3245, pp. 278–289. Springer, Heidelberg (2004)
2. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: KDD, pp. 204–212 (2008)
3. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically significant biclusters in gene expression data. Bioinformatics 18(suppl. 1), S136–S144 (2002)
4. Cheng, Y., Church, G.M.: Biclustering of expression data. In: The 8th International Conference on Intelligent Systems for Molecular Biology, vol. 8, pp. 93–103 (2000)
5. Kluger, Y., Basri, R., Chang, J.T., Gerstein, M.: Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions. Genome Research 13, 703–716 (2003)

6. Turner, H., Bailey, T., Krzanowski, W.: Improved biclustering of microarray data demonstrated through systematic performance tests. Computational Statistics & Data Analysis 48(2), 235–254 (2005)
7. 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), 1520–1527 (2010)
8. Ihmels, J., Friedlander, G., Bergmann, S., Sarig, O., Ziv, Y., Barkai, N.: Revealing modular organization in the yeast transcriptional network. Nature Genetics 31(4), 370–377 (2002)
9. Truong, D.T., Battiti, R., Brunato, M.: Discovering Non-redundant Overlapping Biclusters on Gene Expression Data. In: ICDM 2013, pp. 747–756. IEEE (2013)
10. The Cancer Genome Atlas Network: Comprehensive molecular portraits of human breast tumours. Nature 490(7418), 61–70 (October 2012)
11. Parker, J.S., Mullins, M., Cheang, M.C.U., Leung, S., Voduc, D., Vickery, T., Davies, S., Fauron, C., He, X., Hu, Z., Quackenbush, J.F., Stijleman, I.J., Palazzo, J., Marron, J.S., Nobel, A.B., Mardis, E., Nielsen, T.O., Ellis, M.J., Perou, C.M., Bernard, P.S.: Supervised risk predictor of breast cancer based on intrinsic subtypes. Journal of Clinical Oncology 27(8), 1160–1167 (2009)
12. Mermel, C.H., Schumacher, S.E., Hill, B., Meyerson, M.L., Beroukhim, R., Getz, G.: GISTIC2.0 facilitates sensitive and confident localization of the targets of focal somatic copy-number alteration in human cancers. Genome Biology 12(4) (2011)
13. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. IEEE/ACM Transactions on Computational Biology and Bioinformatics 1(1), 24–45 (2004)
14. Calders, T., Goethals, B., Jaroszewicz, S.: Mining rank-correlated sets of numerical attributes. In: KDD 2006, pp. 96–105. ACM, New York (2006)
15. Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Revisiting Numerical Pattern Mining with Formal Concept Analysis. In: IJCAI, pp. 1342–1347 (2011)
16. Song, C., Ge, T.: Discovering and managing quantitative association rules. In: CIKM 2013, pp. 2429–2434. ACM, New York (2013)
17. Kontonasios, K.-N., Vreeken, J., De Bie, T.: Maximum entropy models for iteratively identifying subjectively interesting structure in real-valued data. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013, Part II. LNCS, vol. 8189, pp. 256–271. Springer, Heidelberg (2013)

Proof Theorem 1. Let us assume that (R, C) is the optimum solution found without additional constraints. Without loss of generality we can assume that (R, C) is maximal in both rows and columns, i.e., there is no row nor column that can be added to obtain the same score or a better score.

Then this optimal solution must also satisfy the constraint:

$$\forall r \in \mathcal{R} : r \in R \leftrightarrow \quad \frac{\sum_{c \in C} \mathcal{M}_{r,c}}{|C|} \geq \theta \leftrightarrow (\sum_{c \in C} \mathcal{M}_{r,c} - \theta) \geq 0. \tag{20}$$

Indeed, assume that $r \in R$ while (R, C) is optimal, then $(\sum_{c \in C} \mathcal{M}_{r,c} - \theta) \geq 0$ must hold, as otherwise the score $\sum_{c \in C, r \in R}(\mathcal{M}_{r,c} - \theta)$ could be improved by removing r from R while keeping C fixed. Conversely, if $r \notin R$ while (R, C) is optimal, it can only be the case that $(\sum_{c \in C} \mathcal{M}_{r,c} - \theta) < 0$, as otherwise the score of the tile could be improved by adding r to R while keeping C fixed.