# Banded structure in binary matrices

Gemma Garriga, Esa Junttila, Heikki Mannila

# Banded Structure in Binary Matrices*

Gemma C. Garriga[1], Esa Junttila[2], and Heikki Mannila[1,2]

[1] HIIT, Helsinki University of Technology, Finland
[2] HIIT, University of Helsinki, Finland

**Abstract.** A binary matrix has a banded structure if both rows and columns can be permuted so that the non-zero entries exhibit a staircase pattern of overlapping rows. The concept of banded matrices has its origins in numerical analysis, where entries can be viewed as descriptions between the problem variables; the bandedness corresponds to variables that are coupled over short distances. Banded data occurs also in other applications, for example in the physical mapping problem of the human genome, in paleontological data, in network data and in the discovery of overlapping communities without cycles.
We study the banded structure of binary matrices, give a formal definition of the concept and discuss its theoretical properties. We consider the algorithmic problems of computing how far a matrix is from being banded, and of finding a good submatrix of the original data that exhibits approximate bandedness. Finally, we show by experiments on real data from ecology and other applications the usefulness of the concept. Our results reveal that bands exist in real datasets and that the final obtained orderings of rows and columns have natural interpretations.

## 1 Introduction

Matrices with binary values occur in many different applications. A typical example is market basket data gathered by retail companies [1]. Further than this, binary matrices abound in a large variety of fields ranging from information retrieval (documents and words occurrences) [5], to bioinformatics and computational biology (genes and probes mappings) [2, 26], or ecology and paleontology (sites and species occurrences) [3, 29]. Understanding the properties of such matrices is therefore important for many applications. A fundamental problem is to uncover structures that will reveal the nature of the relations between the rows and the columns of the binary dataset.

In this paper we study the *banded structure* of binary matrices. A binary matrix is fully banded if both rows and columns can be permuted so that the non-zero entries exhibit a staircase pattern of overlapping rows. See Figure 1 for an illustration of a fully banded matrix.

* A preliminary version of this paper appeared in the proceedings of SigKDD 2008 (see [16]). This current submission corresponds to a significant extension which (1) includes proofs of formal statements; (2) presents new algorithms that outperform the ones presented in the previous paper and lift previous fixed-column permutation

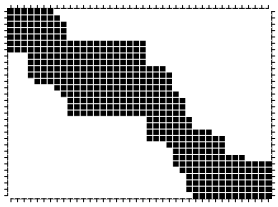


**Fig. 1.** An example of a fully banded matrix.

The concept of banded matrices has its origins in numerical analysis, where the matrix entries indicate connections between variables. From the computational point of view, working with banded matrices is always preferable: The work load involved in performing certain operations, such as multiplication, falls significantly for banded matrices [13], often leading to huge savings in terms of calculation time and complexity. There has been much research focused on minimizing the bandwidth of a matrix (broadly, the distance of non-zero entries from the main diagonal of the matrix) by applying permutations on the original matrix [4, 13, 31].

From the data analysis perspective banded structures can occur in many applications. Consider for example the physical mapping problem of the human genome. Genome biologists break the genome into pieces (clones) which by recursive breaking can be eventually sequenced together. Unfortunately, the information about the relative positions of clones is lost during the breaking process. The physical mapping process starts with the experimental data from which information about the clone overlaps can be derived. The biological community has invested considerable efforts in the analysis of clone–probe matrices, in order to determine useful properties of both clone and probe orderings [2].

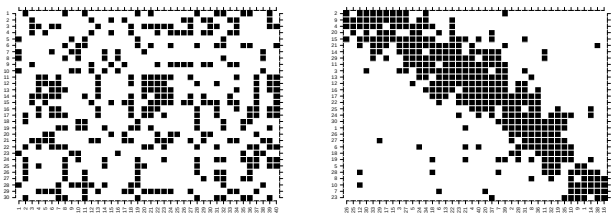
For another application consider the presence/absence data from paleontology. Rows represent sites and columns represent species. A banded structure signifies an overlapping pattern between a set of species occurring in a spatially correlated set of sites. Or similarly, consider dialect word data described by a binary matrix of words used in several locations or municipalities. For this linguistic application a band provides a comprehensive visualization of the spatial distribution of dialects across the different municipalities of a country.

Another application where bands are potentially visible is in the discovery of overlapping communities without cycles in network data [6]. After finding a suitable permutation of rows and columns, a band of 1s from the adjacency matrix of a network should reveal communities of nodes that are strongly connected in a overlapping fashion. For example, consider the Football network dataset [17] containing a match graph of football teams in US colleges in year 2000: a band corresponds to teams that frequently play together and are geographically close

requirements; (3) compares to previous related work; and (4) includes an extended experimental evaluation with a view to new applications.

to each other. Or the network of characters in the novel Les Misérables [19] by Victor Hugo: a band shows the coappearence patterns by groups in the novel. Notice however, that communities in networks define smaller groups of highly interacting components and they cannot be always mapped in the shape of a band. The idea of bandedness is that the locality structure of the communities can be mapped in a overlapping fashion if this exists.

Fully banded matrices are not expected to arise in a real noisy environment. Therefore we study the problem of determining the minimum number of transformations one needs to do on the original binary matrix to uncover a banded structure. The number of such transformations will measure how far a matrix is from being banded. A simple example is shown in Figure 2. In its original form the matrix seems to be random, yet when permuted suitably, it exhibits a high concentration of 1s confined close to the main diagonal band.

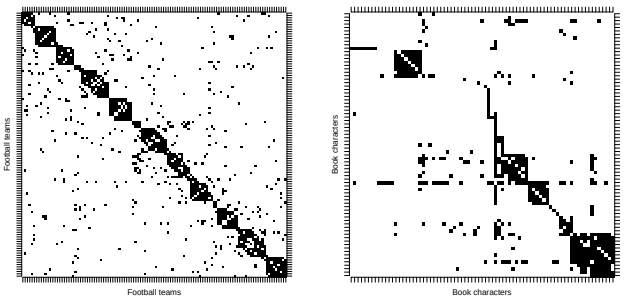


(a) (b)

**Fig. 2.** An example of a binary matrix in its original form (a), and after permuting rows and columns to uncover its banded structure (b).

To better illustrate this optimization problem let us examine in more detail the network data applications mentioned above. Both Football and Les Misérables datasets are small matrices of sizes $115 \times 115$ and $77 \times 77$ respectively. The approximate bands exhibited by the datasets are shown in Figure 3. The band shown by the Football dataset is only 534 bit flips away (that is, number of 0-to-1 or 1-to-0 transformations) from being fully banded, while the band exhibited by Les Misérables only requires 201 flips. In both cases the ordering of rows and columns reveals clusters that are linked in an overlapping fashion. For the football dataset we observe a clear organization of teams in small clusters; indeed, each of those non-overlapping communities represent teams that are geographically close to each other and thus, they usually play one against each other. For Les Misérables dataset we can clearly see which group of characters co-occur together in the novel, and on the other hand, which characters appear more independently across the chapters. Not all network datasets might contain bands, but still they might contain communities. The notion of a community expresses a group of highly interacting nodes, which is more general. The concept

of bandedness we study here forces the mapping of the discovered communities in a overlapping fashion if this is possible.




(a) Football (b) Les Misérables

**Fig. 3.** Band exhibited in two network datasets after proper row and column permutation: Football (a) and Les Misérables (b).

These simple examples already reflect the complexity involved in this problem. Another optimization problem we consider to cope with the unavoidable noise is finding a maximum subset of rows and columns from the original matrix that will exhibit an almost banded structure. Here we explore the combinatorial properties related to bandedness and propose algorithmic solutions to solve the associated optimization problems. Finally, we demonstrate the usefulness of the band-concept by applying our methods to an extensive collection of synthetic datasets and several real life-sciences applications: paleontological dataset, mammal dataset, DNA amplification data and Finnish dialect word data.

The rest of this paper is organized as follows. In Section 2 we give a formal definition of the concept of bandedness and study its theoretical properties. Section 3 introduces the definition of the banded optimization problems. In Section 4 and Section 5 we propose algorithmic solutions for the banded augmentation problem; and in Section 6 for the banded submatrix problem. Finally Section 7 shows empirical results and Section 8 discusses related work.

## 2 Bandedness and its combinatorial properties

This section studies the combinatorial nature of banded matrices. We introduce an incompatibility graph between the columns of the matrix that will characterize the proper permutations for obtaining a fully banded structure. We also define an intuitive binary relation between the rows which, under those proper column permutations, will lead to a banded structure. The combination of these two results drive to a polynomial test for bandedness.

### 2.1 Bandedness

Consider a $n \times m$ binary matrix $M$. In the genetic fingerprinting application, rows would correspond to probes and columns to clones; in the ecological application, rows would be sites and columns would be species. We denote the $i$-th row of $M$ by $M_i$ and the $j$-th column of $M$ by $M^j$. Given a permutation $\pi$ of natural numbers $\{1, \ldots, m\}$ and a row vector $M_i$, we denote with $M_i^\pi$ the permutation of the vector $M_i$ induced by $\pi$. In general $M^\pi$ represents the matrix resulting from applying the column permutation $\pi$ to each one of the rows of $M$. Similarly, $M_\kappa^\pi$ represents the permutation of rows and columns according to $\kappa$ and $\pi$ respectively. Sometimes it is useful to interpret the row and column vectors of $M$ as sets of indices, that is a row $M_i$ is as well a set of column indices that appear in the row.

Roughly speaking, a matrix is fully banded if both rows and columns can be permuted in a way that the non-zero entries exhibit a staircase pattern of overlapping rows (see Figure 1). A formal definition follows.

**Definition 1.** *A binary matrix $M$ is fully banded if there exists a permutation of rows $\kappa$ and a permutation of columns $\pi$ such that (1) for every row $i$ in $M_\kappa^\pi$ the entries with 1s occur in consecutive column indices $\{a_i, a_i + 1, \ldots, b_i\}$ and (2) these indices satisfy $a_i \leq a_{i+1}$ and $b_i \leq b_{i+1}$.*

From the definition we observe that bandedness is a hereditary property: submatrices of a fully banded matrix preserve that property. Also, if $M$ is fully banded also its transpose, $M^T$, is fully banded. Another observation is that for a matrix to be banded it has to satisfy at least the *consecutive-ones property* on the rows and the columns. Formally, a binary matrix $M$ is a consecutive-ones matrix if it is possible to order the columns so that, in every row, the non-zero entries occur in consecutive positions [35]. A consecutive interval of 1s from a row will be denoted as $[a, b]$: all the entries between columns $a$ and $b$ (inclusive) are 1s and the rest 0s. Unless explicitly stated, when a binary matrix $M$ is said to satisfy the consecutive-ones property we assume that it comes permuted to exhibit this consecutive arrangement of ones in all rows. Thus, we can omit permutation $\pi$ from the notation. Testing the consecutive-ones property of a binary matrix and representing permutations $\pi$ (if they exist) can be done in polynomial time via PQ-trees [9], and more recently, in linear time with a certifying algorithm [24]; unfortunately, these algorithms do not handle noise.

Not all matrices with the consecutive-ones property are banded, i.e., part (2) in the definition is not superfluous. For example, the following is a consecutive-ones matrix for the rows and the columns yet none of the row–column permutations leads to a staircase pattern of ones.

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1}$$

On the other hand, even if the matrix is fully banded, we cannot expect that all column permutations from the consecutive-ones property will lead to an overlapping sequence of rows. An example is the following matrix, which as shown

in Equation 2 has consecutive ones for the rows but it is only banded when the last column is placed first.

$$\begin{pmatrix} 0\,0\,0\,1 \\ 1\,1\,1\,1 \\ 0\,1\,1\,0 \end{pmatrix} \tag{2}$$

Next we characterize exactly the relation of banded matrices with consecutive-ones matrices. This will be possible via the following binary relation between rows.

**Definition 2.** *Let $M_i = [a, b]$ and $M_j = [a', b']$ be two rows with the consecutive-ones property. We say that $M_j$ is* properly included *in $M_i$, denoted $M_j \prec M_i$, if and only if $a < a'$ and $b' < b$. We denote by $M_i \frown M_j$ whenever $M_i \not\prec M_j$ and $M_j \not\prec M_i$.*

As an example, the third row in the small matrix of Equation 2 is properly included in the second row. The rows of a consecutive-ones matrix $M$ are a *Sperner family of intervals* if for any two rows $M_i, M_j$ we have $M_i \frown M_j$. This Sperner property on the family of row intervals can be seen as a restricted version of the Sperner property on the family of row sets: two rows $M_i$ and $M_j$ with $M_i \subset M_j$ might, or might not satisfy $M_i^\pi \not\prec M_j^\pi$. This depends on the column permutation $\pi$ of the consecutive-ones arrangement for $M$, which, at the end, determines the starting and ending points of the row intervals. For example, the third row in Equation 2 would not be properly included in the second row if the last column was placed first. The following statement characterizes exactly this relation.

**Lemma 1.** *A binary matrix $M$ is fully banded if and only if $M$ is a consecutive-ones matrix for a permutation of columns $\pi$ where every two rows $i$ and $j$ satisfy $M_j^\pi \frown M_i^\pi$.*

*Proof.* If for every two rows $M_i^\pi = [a_i, b_i]$ and $M_j^\pi = [a_j, b_j]$ with consecutive ones we have $M_i^\pi \frown M_j^\pi$, then: either $a_i \leq a_j$ and $b_i \leq b_j$, or $a_j \leq a_i$ and $b_j \leq b_i$. Under these conditions we can easily establish a preorder of rows by sorting them in ascending value of $a_i$'s, while resolving ties with the ascending value of their $b_i$'s; if two rows are exactly the same, then ties between them can be resolved arbitrarily. This preorder defines directly the banded structure of Definition 1. The other direction of the statement is trivially implied from the definition of fully banded matrices. □

### 2.2 An incompatibility graph for bandedness

The set of orderings that satisfies bandedness is a subset of those satisfying consecutive-ones, as argued with Lemma 1. Unfortunately, checking whether a matrix is fully banded would be unfeasible if we had to go through all the permutations of the consecutive-ones property of the same matrix: there might be exponentially many such permutations. The key lies in identifying the difference

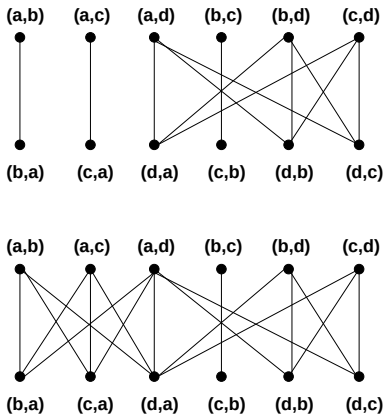



**Fig. 4.** The incompatibility graph corresponding to the matrix from Equation 1, with row sets $M_1 = \{a, b, c\}$, $M_2 = \{b, c, d\}$, and $M_3 = \{c\}$. The upper figure shows the incompatibilities introduced by only the first row $M_1 = \{a, b, c\}$, together with the rule $(x, y)$ and $(y, x)$ are incompatible. The lower figure shows the whole graph obtained by adding the incompatibilities from all rows. $M$ has the consecutive-ones property as this graph is bipartite.

between bandedness and consecutive-ones. In the remaining of this section we will study the incompatibilities that arise under certain permutations of the columns to derive a polynomial test for bandedness.

A very convenient data structure to deal with the valid column orderings of a consecutive-ones relation is the *incompatibility graph*, as introduced in [24]. An incompatibility graph is simply an undirected graph whose vertices are pairs of elements $(a, b)$, for all column indices $a \neq b$, and whose edges reflect the incompatibility relation derived from the following observations.

Suppose we want to discover the consecutive-ones ordering of a set of columns. For each pair of columns $a, b$ of $M$, we have that the order $(a, b)$ (column $a$ before $b$) is incompatible with order $(b, a)$. Moreover, for a triplet of columns $a, b, c$ and a row $i$ of $M$ with $a, c \in M_i$ and $b \notin M_i$, the pairs $(a, b)$ and $(b, c)$ cannot appear in the same consecutive-ones relation. Namely, they would imply that $b$ goes between $a$ and $c$ in the column ordering, which would imply that $M$ fails to be consecutively ordered in the $i$-th row. Therefore, $(a, b)$ and $(b, c)$ are incompatible in this case. It follows from definition that only these types can violate consecutive-ones property. The incompatibility graph $G(M)$ of $M$ is an undirected graph formed by adding all edges of these types. A toy example of an incompatibility graph is shown in Figure 4. It is then clear that a consecutive-ones relation must not contain incompatible pairs, thus giving the following result.

**Proposition 1 ([24]).** *A binary matrix $M$ has the consecutive-ones property if and only if its incompatibility graph $G(M)$ is bipartite.*

Next we study further the observation given by Lemma 1 and use the incompatibility graph to test the bandedness property in polynomial time. The idea is to augment the graph $G(M)$ with more incompatibility edges; basically, those incompatibility ordering pairs that restrict the consecutive-ones orderings to those preserving a Sperner family of row intervals. We denote the augmented graph with $\hat{G}(M)$.

Suppose we have two rows $i$ and $j$ of $M$ and $M_i \subset M_j$. Let $a, b, c \in M_j$ be column indices such that $a \in M_i$ but $b, c \notin M_i$. Then if the matrix is fully banded, the orderings $(b, a)$ and $(a, c)$ cannot both hold. Namely, if $b$ is before $a$ and $a$ is before $c$, we have entries as follows.

$$\begin{array}{lccc} \text{row} & b & a & c \\ \hline M_i\text{:} & 0 & 1 & 0 \\ M_j\text{:} & 1 & 1 & 1 \\ \hline \end{array} \tag{3}$$

But in this case, the ordering cannot satisfy the conditions of a full band as one interval is included in the other (as seen in Lemma 1 this is a necessary condition for bandedness). We call these new edges added to $G(M)$ to form $\hat{G}(M)$ the set of *Sperner conflicting pairs*. As an example consider the matrix in Equation 1: it has the consecutive-ones property, as shown in the incompatibility graph of Figure 4. However, when adding to that graph the incompatible pairs from $M_1 \backslash M_3 = \{a, b\}$ and $M_2 \backslash M_3 = \{b, d\}$, the final graph would not be bipartite anymore. This implies that the matrix in Equation 1 is not banded in any of the consecutive-ones permutation.

Adding Sperner conflicting pairs to graph $\hat{G}(M)$ restricts the set of consecutive-ones orderings to fully banded orderings. To see this, recall that part (2) in Definition 1 separates bandedness from consecutive-ones. The only way for matrix $M$ to violate part (2) is that the matrix in Equation 3 is a submatrix of column-ordered $M$. The new edges prevent this. Notice that by construction the incompatibility graph is complete: for each edge there is an incompatiblity pair (due to part (1) or part (2) of Definition 1) and for every incompatibility pair there will be an edge reflecting such conflict in the graph.

The number of Sperner conflicting pairs that can be potentially added to the graph $\hat{G}(M)$ is at most quadratic in $n$. A way of testing bandedness in linear time in the number of edges of the incompatibility graph is then the following.

**Proposition 2.** *A matrix $M$ is fully banded if and only if its incompatibility graph augmented with the set of Sperner conflicting pairs is bipartite.*

*Proof.* It follows directly from the construction of the incompatibility graph and the property stated by Lemma 1. In the incompatibility graph an ordering is defined by a set of nodes. An ordering of columns satisfying the fully banded property must have no incompatible pairs in the graph augmented with the Sperner conflicting pairs. Thus, a proper column ordering for the fully banded property is an independent set in this graph that consists of half of the vertices.

The reverse of a fully-banded ordering is also a fully-banded ordering, so the remaining vertices of the incompatibility graph must be also an independent set. Therefore, the incompatibility graph must be bipartite whenever the matrix is fully-banded. This is also a sufficient condition: the incompatibility graph fails to be bipartite whenever the matrix fails to be fully-banded. It follows that if the graph is not bipartite then it must have an odd cycle, and therefore there is no independent set that contains half of the vertices in the graph, implying that there is no column ordering that would make the matrix fully banded. □

We derive next a generalization from Proposition 2. The advantage is that it will summarize the conflicting Sperner pairs directly into an augmentation of $M$. This will be particularly important for the algorithmic solutions we later present to our optimization problems.

**Lemma 2.** *Let $\hat{M}$ be the binary matrix $M$ augmented with a set of new rows $M_{ij} = M_j \backslash M_i$ for every two rows $M_i \subset M_j$. Then we have that $M$ is fully banded if and only if $\hat{M}$ has the consecutive-ones property.*

*Proof.* If any two rows $i$ and $j$ of $M$ such that $M_i \subset M_j$ satisfy $M_i \frown M_j$ under a certain permutation $\pi$ of the consecutive-ones, then $M_j \backslash M_i$ will be consecutive-ones for that permutation $\pi$ as well. On the other direction: any permutation $\pi$ of the consecutive-ones on $\hat{M}$, preserves both the consecutive-ones property of $M$ and the Sperner family of intervals of their rows. □

The maximum number of rows that can be potentially added to the original $M$ is at most $n \cdot (n-1)/2$.

## 3 Problem definitions

Real world matrices are not expected to be fully banded. Therefore we introduce a measure that computes how far a matrix is from being banded. We will do so by looking at the minimum number of 0s that need to be transformed into 1s to make a matrix banded. We name this optimization problem as the Minimum Banded Augmentation problem.

*Problem 1.* (Minimum Banded Augmentation, Mba) Given a binary matrix $M$, find the minimum number of 0s that need to be transformed into 1s so that $M$ becomes fully banded.

The formulation of the problem above offers a way to estimate how strong the underlying banded pattern is; this will be closely related to the permutations of rows and columns needed to find the full band. In addition, if a dataset contains a banded pattern, we can identify entries that are likely errors by checking which entries were transformed. In the best case, the fully banded matrix induced by transformations is closer to reality than original erroneous data is. From now on, the transformations are also called *flips*, and the solutions to the Mba problem are called *minimum-flips values*.

*Example 1.* Finding the minimum number of 0s that need to be transformed into 1s in order to make the matrix band is not easy. Consider the following input matrix and the transformation.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & \mathbf{1} \end{pmatrix}$$

Despite the appearances, only one flip (bottom right) is enough to transform the matrix into a full band. Notice that after this flip, the third of the columns in the matrix can be placed in the second position to display the full band.

We denote with $\beta(M)$ the optimal number of flips to make the matrix fully banded. Matrices that are fully banded satisfy $\beta(M) = 0$. A trivial observation following from the symmetry of our definition of fully banded matrices is the following.

**Proposition 3.** *For a binary matrix $M$, we have $\beta(M) = \beta(M^T)$, where $M^T$ is the transpose of $M$.*

For ecological and paleontological data we often have the situation where the 1s are reasonably certain, but the 0s can be missing values. Thus the MBA problem is quite natural. However, for other types of applications it is useful to allow flips in both directions, from 1 to 0 and from 0 to 1. We refer to the version of MBA as the BIDIRECTIONAL MBA. We will denote with $\beta_B(M)$ the optimal number of such bidirectional flips to obtain a fully banded matrix. Note that $\beta_B(M) \leq \beta(M)$.

A variation of the BIDIRECTIONAL MBA is to consider its weighted version: suppose we have a cost associated to 0-to-1 flips and a cost associated 1-to-0 flips; the weighted problem becomes that of finding a transformation with the minimum cost. This weighted variation of the BIDIRECTIONAL MBA can be especially useful in ecological applications, as it will be justified in the experiments.

*Example 2.* To illustrate the importance of bidirectional flips, consider the following input matrix as in Example 1 with one extra row.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & \mathbf{0} \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Even if we have one more row, only one flip is necessary to transform the band into a full band: in this case, the last 1 on the second row is flipped. Again, move the third column to the second position to see the full band.

The algorithmic complexity of the MBA problem is expected to be hard. The basis of this assumption arises from Lemma 2 and the known fact that the CONSECUTIVE-ONES MATRIX AUGMENTATION problem (with 0-to-1 flips)

is NP-complete [27]. Also symmetric banded matrices have an interpretation as proper interval graphs [30], and general banded matrices as proper interval digraphs [32]. Both the edition and deletion of edges to transform a general graph into an interval graph is NP-complete [10].

A second problem we study here is the following.

*Problem 2.* (Maximum Banded Submatrix, Mbs) Given a binary matrix $M$ and an integer $k$, find the maximum submatrix $M'$ of $M$ such that it is banded after at most $k$ flips.

The Mbs problem will be useful in datasets exhibiting several independent band structures, or also, when noise is too high to identify a clear band from the complete dataset. In essence, a solution extracts the most relevant banded pattern in a dataset. Since Mbs is only a generalization of the Mba and Bidirectional Mba problem, the algorithmic complexity is also expected to be hard.

## 4 Algorithms with fixed column permutation

Since the values of $\beta(M)$ and $\beta_B(M)$ cannot be computed exactly for larger instances, we consider algorithms for their estimation in both the Mba and the Mbs problems. We denote those upper bounds with $\tilde{\beta}(M)$ and $\tilde{\beta}_B(M)$.

We first investigate a special form of the problems where we assume that a column permutation that establishes the banded structure is already known. Later we will show ways to lift this requirement.

### 4.1 Algorithms for Mba

By Lemma 1, when having a fixed column permutation the Mba problem decomposes into enforcing the consecutive-ones property for single rows, and after that, making some final flips to ensure that rows form a Sperner family of intervals. Algorithm 1 outlines this `Fixed Permutation` solution.

Assuming that the column permutation $\pi$ is given and that Mba only allows 0-to-1 flips, there is only one exact way of producing a consecutive-ones relation: by flipping all possible 0s falling between 1s for each row of $M^\pi$ (lines 4–6). A second phase has to ensure that all row intervals will be pairwise overlapping, that is $M_i^\pi \frown M_j^\pi$ for all rows $i \neq j$ (lines 7–13). Since only 0-to-1 flips are allowed, the solution is simply to extend the row intervals. An *extension* of $M_i = [a, b]$ means to update the endpoints of the interval for a new $[a', b']$ such that $a' \leq a \leq b \leq b'$.

At every step the algorithm takes a row $M_i^\pi$ and calculates its optimal extension (if this is needed). To do so it will select all the superintervals $M_j^\pi \succ M_i^\pi$ and check all the potential extensions for $M_i^\pi$ that would resolve the Sperner conflicts that the $i$-th row has with those $j$-th rows.

An extension of $M_i^\pi$ that will always resolve all Sperner conflicts for that row can either be a left-hand side extension to the leftmost $M_j^\pi \succ M_i^\pi$ (line 10 (a)); a

**Algorithm 1** The `Fixed Permutation` algorithm for MBA

1: **Input:** An $n \times m$ binary matrix $M$ and a permutation $\pi$
2: **Output:** A permutation $\kappa$ of rows
3: Fix the column permutation of $M$ to be $\pi$
4: **for each** row $i$ in $M^\pi$ **do**
5: Flip all those 0s falling between 1s
6: **end for**
7: **for each** row $i$ in $M^\pi$ s.t. $M_i^\pi = [a, b]$ **do**
8: Let $\mathcal{C} = \{M_j^\pi = [a_j, b_j] \mid M_i^\pi \prec M_j^\pi\}$
9: Extend $M_i^\pi = [x, y]$ with the best from the following options:
10: (A) $x = \min\{a_j \mid [a_j, b_j] \in \mathcal{C}\}$ and $y = b$
11: (B) $x = a$ and $y = \max\{b_j \mid [a_j, b_j] \in \mathcal{C}\}$
12: (C) $x = a_j$ and $y = \max\{b_k \mid [a_k, b_k] \in \mathcal{C}, a_k < a_j\}$, for every $M_j^\pi = [a_j, b_j] \in \mathcal{C}$
13: **end for**
14: Sort the rows $[a, b]$ of $M^\pi$ in ascending order of $a$s, resolving ties with the ascending order of their $b$s.

right-hand side extension to the rightmost $M_j^\pi \succ M_i^\pi$ (line 11 (B)); or, extending $M_i^\pi$ to both left and right-hand sides with a combination of two superintervals (line 12 (C)). This only requires checking the starting point of each $M_j^\pi \succ M_i^\pi$ and combining it with the right-most ending point from all other superintervals $M_k^\pi \succ M_i^\pi$ whose starting point comes before $M_j^\pi$. Eventually, for a row $i$ the algorithm takes the extension that represents fewest transformations.

Resolving the Sperner conflicts for a row $i$ does not change the optimal extension of other rows $j \neq i$ (there are no cascade effects). To see this suppose that $i$ includes a new interval $k$ after its extension, and let $j$ be the interval that previously contained $i$ and that originated the best optimal extension of $i$ (in lines 10–12). This interval $j$ contains $k$ by transitivity, and indeed, it also contained $k$ even before $i$ was extended. This implies that when processing $k$, its optimal extension cannot be changed because of a previous extension of interval $i$.

*Example 3.* To illustrate the `Fixed Permutation` algorithm, consider again the small matrix in Example 1.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & \mathbf{1} & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & \mathbf{1} \end{pmatrix}$$

Under this given fixed column permutation, the algorithm would transform first all 0s falling between 1s: this corresponds to flipping one 0 in the first row of the matrix. Then, the algorithm would resolve the Sperner conflicts between the rows: there is a conflict between the second and the third row, which can be resolved by flipping the bottom right 0 into a 1. The matrix is fully banded for this permutation after doing two 0-to-1 flips. The number of flips done in this illustration is 2, while we showed in Example 1 that only one flip would be

**Algorithm 2** The `Bidirectional Fixed Permutation` algorithm for BIDIRECTIONAL MBA

**Input:** An $n \times m$ binary matrix $M$ and a permutation $\pi$
**Output:** A permutation $\kappa$ of rows
Fix the column permutation of $M$ to be $\pi$
**for each** row $i$ in $M^\pi$ **do**
  Let $W_i^\pi$ be the weight vector for row $i$ of $M$
  Let $[a, b]$ be the maximum consecutive subarray on $W_i^\pi$
  Update $M_i^\pi = [a, b]$
**end for**
**for each** pair of rows $i, j$ in $M^\pi$ **do**
  **if** $M_i^\pi \subset M_j^\pi$ **then**
    Let $A = M_j^\pi \backslash M_i^\pi$
    Let $W_A$ be the weight vector for $A$
    Let $[a, b]$ be the solution of the maximum consecutive subarray of $W_A$
    Update $M_i^\pi$ preserving $M_j^\pi \backslash M_i^\pi = [a, b]$
  **end if**
**end for**
Sort rows $[a, b]$ of $M^\pi$ in an ascending order of $a$s, while deciding ties with the ascending order of their $b$s.

necessary to make the matrix fully banded. Therefore, an important component of this fixed-permutation algorithm is the initial permutation that is given to the algorithm.

For a fixed permutation $\pi$ of the columns, the `Fixed Permutation` algorithm computes exactly $\beta(M^\pi)$ in polynomial time. This will be always an upper bound of $\tilde{\beta}(M)$. The number of comparisons between rows made by the algorithm will be at most $n \cdot (n-1)/2$. Since every comparison requires at most $n$ comparisons (line 12 (C)), the final complexity of the algorithm is of the order $\mathcal{O}(n^3)$.

Finally we have the following property derived from the above algorithm.

**Proposition 4.** *If a permutation $\pi$ of the columns is fixed,* MBA *for $M^\pi$ is solvable in polynomial time.*

### 4.2 Algorithms for BIDIRECTIONAL MBA

Next we study the properties of the algorithms for the BIDIRECTIONAL MBA problem. Again we will use the same principle of assuming a fixed column permutation $\pi$. The basic idea here is also simple: first solving optimally the consecutive-ones property for bidirectional flips on the $M^\pi$ and after that, resolving Sperner conflicts between rows. Algorithm 2 gives the outline of the `Bidirectional Fixed Permutation` algorithm.

The first simple observation we use is the following.

**Proposition 5.** *If a permutation $\pi$ of the columns is fixed, the minimum number of bidirectional flips that lead to a consecutive-ones relation on $M^\pi$ can be solved exactly in linear time.*

To see this consider a $n \times m$ matrix of weights $W$ where each non-zero entry of $M$ will be assigned a $+1$ in $W$, and each zero entry of $M$ will be assigned a weight of $-1$. This kind of weight matrix assumes equal cost between the two types of flips.

$$W_i^j = \begin{cases} +1 & \text{if } M_i^j = 1 \\ -1 & \text{if } M_i^j = 0 \end{cases}$$

In this form, the problem of finding the optimal consecutive-ones solution with bidirectional flips for $M_i^\pi$ corresponds to solving the *maximum subarray problem* on $W_i^\pi$. The aim of the maximum subarray problem is, for a given array of numbers, to find a consecutive subarray such that the sum of the numbers in the subarray is maximum. This can be done in linear time in the size of the array by making use of a scan-line algorithm [12]. In essence, for a solution $[a, b]$ of the maximum subarray algorithm over $W_i^\pi$, we should update $M_i^\pi$ with consecutive ones between the columns $a$ and $b$, while setting to 0s the rest of its entries. This corresponds to lines 4–8 in Algorithm 2. Note that weights of the matrix $W$ can be tuned according to the apriori knowledge of the application. In this way we would be solving the weighted Bidirectional Mba. As an example, after substituting $+2$ for $+1$, 0-to-1 flips have half the cost of 1-to-0 flips.

In the second phase (lines 9–16) the algorithm proceeds by removing the Sperner conflicts between the rows of $M^\pi$, which at this stage already exhibit a consecutive-ones property. The useful technical observation comes from Lemma 2: A matrix $M$ will be fully banded as long as the augmented $\hat{M}$ has consecutive ones. Therefore, to eliminate all possible Sperner conflicts between the row intervals of $M^\pi$, the algorithm simply has to go through all the extra rows described in $\hat{M}$ and make them consecutive ones. When propagating the changes back to $M^\pi$ we would end up with a banded matrix. As above, we can make use of the maximum subarray problem on the extra rows of $\hat{M}$ to solve the problem exactly for bidirectional flips. It only remains to update rows in $M^\pi$ so that they are kept consistent with the changes made over $\hat{M}$. The final obtained solution on $M^\pi$ will be always banded. Basically, this corresponds exactly to deciding the best number of bidirectional flips that will eliminate Sperner conflicts in a pairwise comparison of rows in $M^\pi$. The final complexity of the algorithm is $\mathcal{O}(n^2 m)$.

The `Bidirectional Fixed Permutation` algorithm, however, is not optimal for the Bidirectional Mba problem on $M^\pi$. The problem resides in the second phase: rows are compared in a pairwise fashion and globally beneficial updates may be missed. In general, it seems that the Bidirectional Mba problem would be still hard for a fixed permutation of the columns.

*Example 4.* To illustrate the `Bidirectional Fixed Permutation` algorithm consider the matrix in Example 2.

$$\begin{pmatrix} 1\,0\,1\,0 \\ 1\,1\,1\,1 \\ 0\,1\,1\,0 \\ 0\,1\,1\,0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1\,\mathbf{1}\,1\,0 \\ 1\,1\,1\,1 \\ 0\,1\,1\,0 \\ 0\,1\,1\,0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1\,1\,1\,0 \\ 1\,1\,1\,\mathbf{0} \\ 0\,1\,1\,0 \\ 0\,1\,1\,0 \end{pmatrix}$$

Under this given fix column permutation, the algorithm would transform first the matrix into a consecutive-ones relation; as shown before, this can be solved exactly by finding the consecutive maximum subarray problem for each row. In the example, this corresponds to flipping second 0 in the first row to a 1 (notice that we could have also flipped any of the 1s of the first row into a 0 with the same final cost). In the second phase, the algorithm resolves the conflicts between the rows: there are two conflicts between the second row and the third and fourth row respectively, which will be resolved after flipping the last 1 of the second row into a 0.

As a final step in both Algorithm 1 and Algorithm 2, it only remains to sort the rows with the preorder given by the row intervals to visually exhibit the fully banded structure. The banded pattern in original dataset is described by this row-ordering.

### 4.3 Finding good column permutations

An essential component of Algorithm 1 and Algorithm 2 is to identify a good column permutation that will be fixed since the beginning; new algorithms that lift this restriction will be introduced in the next section. Intuition says that a good permutation will tend to put similar columns close enough to each other in the order. If the matrix exhibits banded structure, an order based on the similarity between columns should also preserve the band. We define next several similarity measures to compare columns and use two different methods to find a proper order.

One measure of similarity between two columns is the *correlation similarity*. Given two columns $M^a$ and $M^b$ the correlation similarity is:

$$\mathrm{CorrS}(M^a, M^b) = (1 + \rho_{ab})/2,$$

where $\rho_{ab}$ represents the Pearson coefficient between the columns. Values range from 1 (identical columns) to 0 (anticorrelated columns).

As an alternative we will use the overlapping measure computed by the *Jaccard coefficient*:

$$\mathrm{J}(M^a, M^b) = \frac{|M^a \cap M^b|}{|M^a \cup M^b|}.$$

This similarity measure captures the particularities of our problem where columns should be increasingly overlapping one with the other. It takes the highest value of 1 when columns $M^a$ and $M^b$ are exactly the same. The overlapping similarity for non-intersecting columns is 0. Other straightforward measures we will use are the *dot product* and the *Hamming distance* metric between columns.

In order to find a good permutation of the columns preserving the similarity or distance relationship, we consider a complete undirected graph whose nodes are the columns of the input matrix $M$. The weight of an edge $\{a, b\}$ is defined by the similarity between $a$ and $b$. This similarity can be defined by any of the measures mentioned above. We will use two methods to find a good order between

**Algorithm 3** The `Alternating` algorithm for MBA or BIDIRECTIONAL MBA

1: **Input:** An $n \times m$ binary matrix $M$, a number of iterations $t$
2: **Output:** A permutation $\kappa$ of rows and a permutation $\pi$ of columns
3: Initialize $\pi$ with a random permutation of the columns
4: Let $A = M^{\pi}$
5: **repeat**
6: Apply a fixed column permutation algorithm on $A$
7: Transpose the current matrix $A = A^T$
8: **until** reaching $t$ iterations
9: Return the pair of row–column permutations, $\kappa$ and $\pi$, with the best band found.

columns: spectral ordering [3] and approximation of the minimum Hamiltonian path (by means of constructing a minimum spanning tree) [12]. Note that similarity measures will be transformed into distances for the Hamiltonian path.

## 5 Lifting the fixed column permutation requirement

Algorithm 1 and Algorithm 2 are dependent on fixing a column permutation beforehand. In this section we propose algorithmic strategies that avoid this dependency: two new methods (`Alternating` and `Simulated Annealing`) and a pre-existing method (`Barycentric`), now adapted to solve band-problems.

### 5.1 Alternating method

The first of these strategies follows from Proposition 3. It means that in practice we can solve the MBA problem or BIDIRECTIONAL MBA problem on either $M$ or $M^T$. This suggests the following alternating approach: solving the problem of finding a good row permutation given the current column permutation (with any of the fixed permutation algorithms 1 or 2) and then, transposing the matrix to iterate with the current configuration until convergence, or until a certain number of iterations is reached. The pseudo-code of this `Alternating` algorithm is shown in Algorithm 3.

Notice that the alternating strategy just described does not necessarily converge for all matrices. Indeed, it is not possible to guarantee that the value of $\tilde{\beta}(M)$ or $\tilde{\beta}_B(M)$—the approximated value for $\beta(M)$ and $\beta_B(M)$ respectively—will decrease after each iteration. To stop the alternating process, we bound the number of iterations by an input parameter $t$. The output of this algorithm is the pair of row–column permutations that achieved the best value of $\tilde{\beta}(M)$ among the $t$ iterations.

### 5.2 Barycentric method

Another strategy that transposes the matrix is presented by the `Barycentric` algorithm, originally proposed in [33]. In essence, the `Barycentric` algorithm

finds good permutations of both rows and columns based on a barycenter measure (average position of 1s in a row). Let the barycenter of row $i$ be defined as follows:

$$\text{Barycenter}(i) = \frac{\sum_{j=1..m} j \cdot M_i^j}{\sum_{j=1..m} M_i^j}.$$

The `Barycentric` algorithm first computes the barycenter for all rows, then it orders (stable ordering) the rows from smallest to largest barycenter, and finally, it transposes the matrix $M$ to iterate again following the same strategy until convergence. Notice that this sorting process does not use flips or compute in any way the borders of the band at any iteration. Indeed, `Barycentric` only orders rows and columns according to the barycenter measure in an iterative fashion.

### 5.3 Simulated annealing method

Finally, we present a simulated annealing metaheuristic to find a good approximation of the global minimum for the MBA and BIDIRECTIONAL MBA problems. We do not expect this stochastic method to be as competitive in terms of running time as the other algorithms presented this far. In general, the simulated annealing strategy comes with a high computational cost, but it yields near-optimal results in most cases. For this reason we use this strategy as a reference method for comparing the different bandedness solutions.

Simulated annealing [18] is a general stochastic optimization method, used successfully to solve various combinatorial problems. In band-related problems, different row–column permutations of the matrix are interpreted as "states" and minimum-flips costs as "energy values". We seek an order of rows and columns for the matrix $M$ that produces the smallest energy, that is $\beta_B(M)$. The search space includes all permutations of rows and columns.

The simulated annealing algorithm starts with a random state and a large temperature parameter $T$, for example 10.0. The algorithm keeps track of the current state $S$ of the system all the time. The temperature is decreased at each iteration of the algorithm until a pre-determined number of iterations has passed. At each iteration, a new candidate state $S'$ is generated stochastically by function `Neighbor`$(S)$. The new state $S'$ is then accepted as a new current state with probability $\min\{1, e^{((E(S)-E(S'))/T)}\}$, where $E$ is the energy function whose minimum we seek. In our case, this energy function is the minimum number of flips needed to make the matrix fully banded, with the restriction that both current row and column permutations remain fixed. An optimal polynomial-time algorithm (`Optimal Visual Band`) for the problem is given shortly. Algorithm 4 shows the scheme of the simulated annealing strategy.

We next define the technical details for the auxiliary functions `Neighbor` and `Optimal Visual Band` for the simulated annealing algorithm.

**Picking a neighbor** The most important requirement for the function `Neighbor` is that all states (in our case, permutations) in the search space should be ac-

**Algorithm 4** The `Simulated Annealing` algorithm for MBA or BIDIRECTIONAL MBA

**Input:** An $n \times m$ binary matrix $M$, a number of iterations $t$, a large temperature $T$ and a temperature multiplier $\alpha \in [0.95, 1)$
**Output:** A permutation $\kappa$ of rows and a permutation $\pi$ of columns
Initialize state $S = M$ and energy $E = \texttt{Optimal\_Visual\_Band}(S)$
Initialize best solution: $S_{\text{best}} = S$ and $E_{\text{best}} = E$
**repeat**
  Pick a neighbor $S' = \texttt{Neighbor}(S)$
  Compute energy $E' = \texttt{Optimal\_Visual\_Band}(S')$
  **if** $E' < E$ **then**
    Update best solution $S_{\text{best}} = S'$ and $E_{\text{best}} = E'$
  **end if**
  Update $S = S'$ and $E = E'$ with probability $\min\{1, e^{((E-E')/T)}\}$
  Decrease temperature $T = T \cdot \alpha$
**until** reaching $t$ iterations
Return the row–column permutations of the best solution $S_{\text{best}}$.

cessible by repeating the candidate generation process sequentially. We consider several candidate generation methods [34] for finding good permutations and we apply them in our 2-dimensional permutation problem. For all these methods we treat an ordering as a cycle: after the last row the first row follows. We summarize the methods here.

- **Swap-$k$**: Choose two random rows and swap them; do the same for two random columns. Repeat $k$ times.
- **Adj-swap-$k$**: Randomly choose a row index $r$, then swap (adjacent) rows $r$ and $r + 1$; do the same for a randomly chosen column. Repeat $k$ times.
- **Reverse**: Choose two random row indices $r$ and $r'$. Starting from $r$, move forward in row-order until $r'$ is found. Reverse the order of all encountered rows, including $r$ and $r'$. Do the same for columns.
- **Relocate**: Choose two random row indices $r$, $r'$, and a random integer $k \in \{1, \ldots, n\}$. Starting from $r$, move forward in row-order until $r'$ is found. Shift all encountered rows $k$ steps forward in row-order. Do the same for columns.
- **Reverse+Relocate**: Apply both reverse and relocate methods simultaneously: first choose a set of rows, then relocate the rows and reverse their order. Do the same for columns.

**Finding the optimal visual band** For `Simulated Annealing` to produce good results, we must compute energy values accurately and efficiently. To this end, we propose a polynomial-time exact algorithm `Optimal Visual Band` when both row and column permutations are fixed.

A fully banded matrix is *visually banded*, if the row–column permutations of the matrix conform to bandedness. The problem to be solved is as follows: given a binary matrix $M$ with fixed row–column permutations and a non-negative

weight matrix $W$, find the minimum cost of transformations needed to make the matrix visually banded.

Let a tuple of column indices $\langle s, e\rangle$ represent the interval $[s, e-1]$ including columns positions $s, \ldots, e-1$; the tuple of indices $\langle s, s\rangle$ represents an empty interval. The *cost of an interval* on a row is the total cost for transforming the row so that the values inside the interval are all 1s and other values are 0s. The goal is to define intervals on each row $r$ in $M$ so that the intervals of 1s across the rows have visual bandedness property and the total cost is minimum.

In order to simplify the algorithm, assume that the weight matrix $W$ has been split in two separate matrices: $W1$ for the costs of transforming 1s and $W0$ for the costs of transforming 0s. If $M_r^c = 1$, then $W1_r^c = W_r^c$ and $W0_r^c = 0$, otherwise $W1_r^c = 0$ and $W0_r^c = W_r^c$.

The algorithm, which is based on dynamic programming, goes through all rows one by one. It ensures that the 3-dimensional array $C$ contains optimal cumulative costs for intervals on the previous rows. Using array $C$, it evaluates the cost of each interval on current row with respect to $M$ and $W$ and stores the cumulative interval cost in $C$. Visual bandedness of intervals is preserved at all times. For each triplet of indices $r, s, e$, varying along the row indices for $r$ and along the column indices for $s$ and $e$, we update array $C(r, \langle e, s\rangle)$ via a classical recurrence:

$$
\begin{aligned}
C(1, \langle 1, 1\rangle) &\leftarrow \sum_{j=1\ldots m} W1_1^j \\
C(r, \langle s, e\rangle) &\leftarrow \min\{\mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3\}, \text{ where} \\
&\qquad \mathrm{R}_1 = C(r, \langle s, e-1\rangle) + W0_r^{e-1} - W1_r^{e-1} \\
&\qquad \mathrm{R}_2 = C(r, \langle s-1, e\rangle) - W0_r^{s-1} + W1_r^{s-1} \\
&\qquad \mathrm{R}_3 = C(r-1, \langle s, e\rangle) + \sum_{j=1\ldots s-1} W1_r^j + \sum_{j=s\ldots e-1} W0_r^j + \sum_{j=e\ldots m} W1_r^j
\end{aligned}
$$

The recurrence is only applied to triplets of indices $r, s, e$ that satisfy $1 \le r \le n$ and $1 \le s \le e \le m+1$, where $n$ is the number of rows and $m$ is the number of columns. As seen in the initialization of the recurrence, the interval of the first row starts as empty; the cost of this interval is the sum of costs of all errors: all 1s on that row. We next describe the three recurrence operators $\mathrm{R}_1, \mathrm{R}_2, \mathrm{R}_3$ used in the recurrence.

- The value of $\mathrm{R}_1$ is available if and only if $s < e$. It represents moving the end of the interval one position ahead on row $r$, essentially adding one entry to the interval. This either introduces an error with cost $W0_r^{e-1}$, or eliminates an error with cost $W1_r^{e-1}$.
- The value of $\mathrm{R}_2$ is available if and only if $s > 1$. It represents moving the start of the interval one position ahead on row $r$, thus removing the leftmost entry from the interval. This either introduces an error with cost $W1_r^{s-1}$, or eliminates an error with cost $W0_r^{s-1}$.

– Value $\text{R}_3$ is available if and only if $r > 1$. It represents fixing the interval on previous row $r-1$ and proceeding to the next row $r$ of $M$. Initially, the interval for row $r$ is identical to that of row $r-1$, updating the cumulative cost as follows: all 1s outside the current interval are errors, as well as all 0s inside the interval. Each error contributes to the total cumulative cost.

Once the recurrence has completed, the optimal solution is the minimum cumulative cost among the last-row entries of $C$: $\min_{1 \leq s \leq e \leq m+1}\{C(n, \langle s, e \rangle)\}$. An exact sequence of optimal intervals can be traced by storing the operator used on each entry.

The time complexity for the `Optimal Visual Band` is $\mathcal{O}(nm^2)$, caused by the size of array $C$. Each sum-operation in the recurrence can be evaluated in constant time: instead of using the costs in $W1$ and $W0$ directly, we construct cumulative weight matrices of both $W1$ and $W0$ on rows beforehand and evaluate the sums using these matrices. Memory consumption can be lowered to $\mathcal{O}(m^2)$, if cumulative costs for row $r-1$ are replaced by costs for row $r$ as they are computed; the time consumption remains unchanged.

## 6 Algorithms for MBS

Next we turn to the banded submatrix problem MBS. Consider a solution matrix of a band-algorithm. As is common in information retrieval, we can compute, for each row and column the number of *true positive* (TP) entries as the number of 1s in the row (or column) that were not transformed into a 0. That is, they correspond to relevant 1s contained in the row (column) for bandedness. Similarly, the *false positive* (FP) entries of a row (or column) correspond to original 0s that were transformed into 1s by the algorithm. Finally, *false negatives* (FN) are represented by those original 1s that were switched into 0 by the banded algorithm.

From here many different measures for evaluating the performance of the banded retrieval system can be computed. For example we can consider precision =TP/(FP+TP), recall=TP/(FP+FN) or accuracy=(TP+ TN)/$n$ of each one of the row and column vectors. The algorithm proposed is called `Iterative row-column elimination`. As the name indicates, it will eliminate rows/columns one at a time, by selecting the one with worst performance so far. Pseudocode of the algorithm is shown in Algorithm 5.

## 7 Experiments

In this section we give experimental ground of the bandedness concept. We demonstrate that our algorithms are computationally effective and produce good results in synthetic datasets. On top of that, we show that the algorithms are able to find banded patterns in a variety of real datasets from life sciences. The banded structures in Figure 2 were found by `Alternating` algorithm.

**Algorithm 5** The `Iterative row-column elimination` algorithm for MBS

**Input:** An $n \times m$ binary matrix $M$ and an integer $k$
**Output:** A submatrix $M'$ of $M$
**repeat**
    Compute $\epsilon$, the number of flips estimated for MBA or BIDIRECTIONAL MBA problem on $M$
    Rank columns/rows with a performance measure
    Select the column/row $i$ with worst performance
    Remove $i$ from $M$
**until** $\epsilon \leq k$

### 7.1 Synthetic Data

**Data generation** Briefly, we generated synthetic data as follows. For given $n$ and $m$ and a width parameter $2w$, we generate a fully banded matrix by means of a random walk in the matrix grid. Starting at coordinate $(0, 0)$ of an $n \times m$ matrix initially set to all zeros, the random walk chooses to move either one step down (i.e. from $(i, j)$ to $(i+1, j)$) or one step to the right (i.e. from $(i, j)$ to $(i, j+1)$) with equal probabilities. Whenever a step to the right is chosen, we will set to 1 all the $w$ entries above the current position and all the $w$ entries below the current position (or less than $w$, if $i < w$). The random walk is designed so that it always reaches the final position $(n, m)$. When it does, the matrix has a clear solid band of maximum thickness $2w$ like shown in Figure 1.

Additionally we can introduce noise by flipping the original values, 0 to 1 or 1 to 0, according to given probabilities. We use $\Pr(0 \to 1)$ as the noise probability for 0-to-1 flips and $\Pr(1 \to 0)$ for 1-to-0 flips.

We generated samples of $50 \times 55$ binary matrices, which are used in all synthetic data experiments. The width parameter was $2w = 30$, which yields approximately 50% fill of 1s. Before running the experiments on these matrices, noise was added and row–column permutations were randomized. We prepared a variety of test settings, including both *balanced noise* (when $\Pr(1 \to 0)$ is the same as $\Pr(0 \to 1)$) and *unbalanced noise* (when they are different), and different weighting schemes.

In all the synthetic experiments we generated 30 sample matrices for each one of our parameter settings. The values in the figures of this section are the averages of the values obtained from the samples. We will next summarize the performance exhibited by our algorithms on synthetic data.

**Methods** The algorithms used in experiments include `Alternating`, `Barycentric`, and `Simulated Annealing`, as well as `Fixed Permutation` and `Bidirectional Fixed Permutation` coupled with several similarity measures and column-ordering methods. Eventually, we used `Optimal Visual Band` method to evaluate minimum-flips values $\tilde{\beta}_B(M)$ for each algorithm after row–column permutations were fixed by each one of the methods.

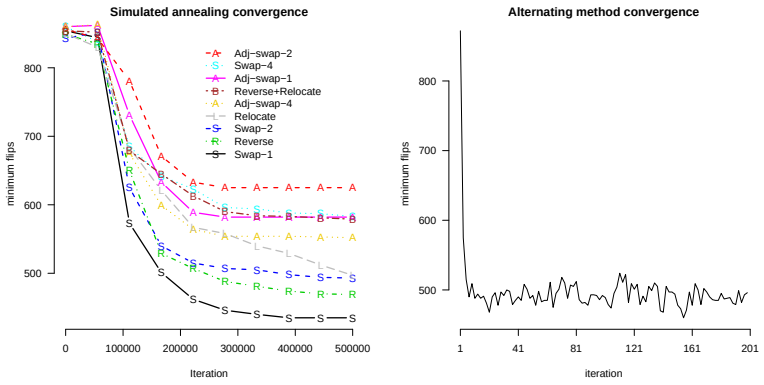
In order to make the evaluation of algorithms more reliable, we introduce two new methods, `Original` and `Random`. In `Random` method, row–column permutations are selected randomly. Another competitor is `Original` method, which has access to the original "correct" permutations. By correct we mean that the original fully banded structure was generated under these permutations. Once noise has been added, the matrix is no longer fully banded, but `Original` gives good permutations nonetheless.

As expected, the `Bidirectional Fixed Permutation` strategy outperformed the `Fixed Permutation` in all cases where noise exists, so we decided to leave the `Fixed Permutation` algorithm out of the results presented next. For the `Bidirectional Fixed Permutation` algorithm, we sorted columns via the spectral ordering with three similarity measures: Pearson correlation, dot product, and Jaccard similarity. We also experimented with finding good columns-orderings via the Hamiltonian path approximation (specifically, minimum spanning tree approximation) with both Hamming distance and Jaccard distance. For visual clarity, only the best results are shown: spectral ordering using dot product and Hamiltonian ordering with Jaccard distance.

For the `Simulated Annealing` method, we first compare the different neighbor schemes: **Swap-**$k$ and **Adj-swap-**$k$ for $k \in \{1, 2, 4\}$; **Reverse**, **Relocate**, and **Reverse**+**Relocate**. Results are shown in Figure 5(a). The number of iterations was one million and the starting temperature was 10. These parameters produced results that we consider near-optimal. In terms of convergence speed, **Swap-**1 proved to be the fastest, as seen in Figure 5(a). This scheme was chosen to be the neighbor scheme for all experiments that include `Simulated Annealing`.

Because of the 2-dimensional nature of band-problems, the number of iterations that `Simulated Annealing` needs is larger than other ordinary permutation problems of same size. We studied the convergence of `Simulated Annealing` with **Swap-**1 scheme extensively by using different number of iterations and temperature-multipliers. Using a large multiplier requires significantly more iterations, which makes the largest multiplier values impractical for this experimental setting. We chose a running-time limit of 2 minutes and settled for multiplier $\alpha = 0.9999$ with 100000 iterations. Other parameter choices that satisfy the time limit were tested but they failed to produce superior results consistently. The chosen parameter combination is conservative: temperature is allowed to drop low enough so that further improvement is unlikely, thus reducing variance. These parameters yield solutions that have, on average, 3% higher cost than those from one million iterations.

In most cases, both `Alternating` algorithm and `Barycentric` algorithm converge very fast, taking less than 30 iterations in average to find good permutations, regardless of matrix size and initial permutations. After the first iterations, the `Alternating` algorithm does not strictly converge towards an optimal solution, but starts oscillating between good and very good solutions, as seen in Figure 5(b). Because of this, we return the best solution among the first 100 iterations.



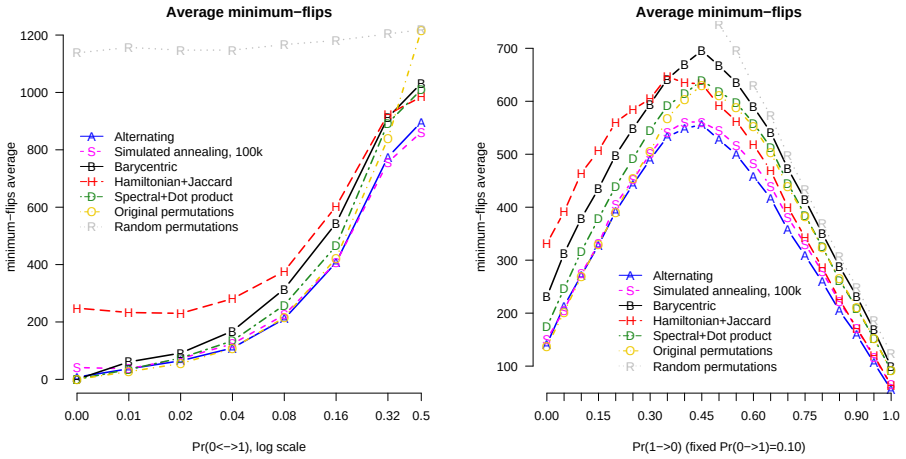

(a) `Simulated Annealing` (b) `Alternating`

**Fig. 5.** Left part (a) shows the convergence analysis and comparison of neighbor schemes for `Simulated Annealing` algorithm. The fastest converging schemes are **swap-**1 and **reverse**. Right part (b) shows the convergence analysis for the `Alternating` algorithm. On the $y$-axis: minimum number of flips necessary to make the matrix fully banded; on the $x$-axis: number of iterations used.

**Results** A summary of the main results follows; first, we use the minimum-flips as quality measure and then a notion of rank correlation measure.

We added balanced noise to the sample of 30 synthetically generated fully banded matrices and then run each of the presented algorithms to obtain the $\tilde{\beta}_B(M)$ values for the matrices. The average results from the 30 runs were then computed for each method, as shown in Figure 6(a). We can see that both `Alternating` algorithm and `Simulated Annealing` algorithm consistently perform as well as the `Original` method, and at higher noise levels, they perform even better. The reason is that the original banded structure starts to disappear with more noise—the original permutation is no longer the best one with respect to minimum-flips value. The other methods do not beat `Original` until extreme noise levels. We discovered that the Hamiltonian path method (in a fixed permutation algorithm) has large variance, caused by non-optimal cycle-to-path transformations.

In the unbalanced noise case we chose to fix one of the noise probabilities $\Pr(0 \to 1) = 0.1$ and added different noise levels of $\Pr(1 \to 0)$ to the input matrices; again, with a sample of 30 matrices in total. To make the comparison reliable, we use an equal cost weighting scheme. As in the balanced noise case, the average result of 30 runs was collected for each method; the results are shown in Figure 6(b). Again, the `Alternating` algorithm and the `Simulated Annealing` algorithm perform very well, beating the `Original` method at noise levels $\Pr(1 \to 0) > 0.25$. Overall, `Alternating` seems to produce best average results, although we noticed its variance to be a bit larger than that of the `Simulated Annealing` algorithm. Curiously, the `Barycentric` method performs



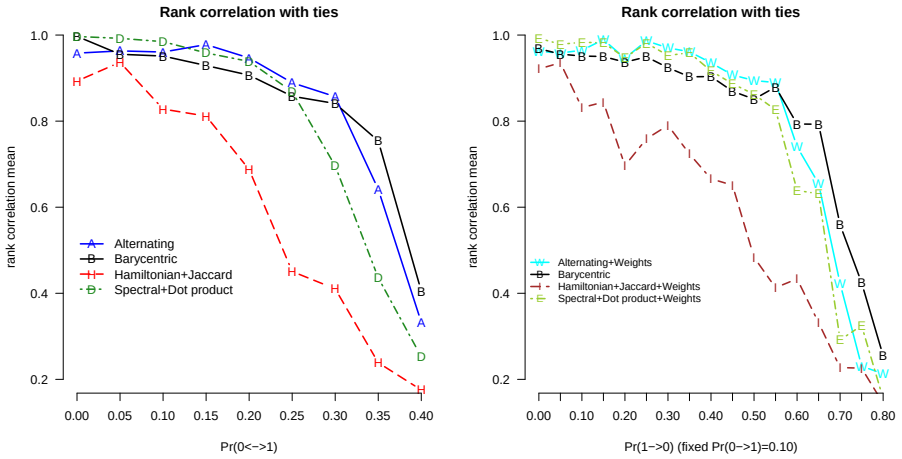

(a) Balanced noise (b) Unbalanced noise

**Fig. 6.** Left part (a) shows the average minimum-flips comparison of the different methods on synthetic data when balanced noise has been added to the original matrices; the $x$-axis corresponds to the variation of noise for both $\Pr(0 \to 1)$ and $\Pr(1 \to 0)$. Right part (b) shows the average minimum-flips comparison of methods on synthetic data with unbalanced noise; the $x$-axis corresponds to the variation of noise for $\Pr(1 \to 0)$ (where $\Pr(0 \to 1) = 0.1$ is fixed all the time). The average value of 30 samples is shown for both plots. `Alternating` and `Simulated Annealing` are the best performers.

poorly here: at high noise levels, the results hardly beat a random permutation. Spectral methods perform a bit better, whereas the results for Hamiltonian methods are two-fold: mediocre results at low noise levels; good results at high noise levels.

Finally, we would like to know how well our methods can recover the original permutations after we have added noise and randomly reordered the rows and columns. For this we use *Spearman rank correlation*, which compares two sets of rankings and computes a correlation coefficient that describes the similarity of the rankings. Here we understand a ranking as an ordering of rows and columns. We compare two rankings, namely *original* and *recovered*: the original ranking matches the row order in the generation process of noiseless data; the recovered ranking comes from the new row ordering retrieved by any of our algorithms. Correlation 1.0 means perfect agreement between rankings; values close to 0.0 indicate that there is no correlation between the two.

The rank correlation results for balanced noise are shown in Figure 7(a). We see that none of the methods dominates others: at low noise levels under 0.15, Spectral methods are able to recover the original order almost perfectly; from 0.15 to 0.30, the `Alternating` algorithm seems to be the best choice; at levels of noise over 0.30, the `Barycentric` method is the best. Overall, these three methods are very good in recovering the original row order. Hamiltonian




(a) Balanced noise (b) Unbalanced noise

**Fig. 7.** Left part (a) shows the average rank correlation comparison of all the methods on synthetic data with balanced noise. On the $y$-axis: Spearman rank correlation measure; on the $x$-axis: variation of noise for both $\Pr(0 \to 1)$ and $\Pr(1 \to 0)$ (balanced noise scenario). Spectral methods, `Alternating`, and `Barycentric` all produce good results. Right part (b) shows the average rank correlation comparison of all the methods on synthetic data with unbalanced noise. On the $y$-axis: Spearman rank correlation measure; on the $x$-axis: variation of noise for $\Pr(1 \to 0)$ (unbalanced noise scenario, where $\Pr(0 \to 1) = 0.1$ is fixed). Basic `Barycentric` method performs well by itself; Spectral methods and `Alternating` are equally good, when the weight parameter is employed.

methods, however, are able to find the original order only at lowest noise levels. Occasionally the `Alternating` method converges towards local minimum, as demonstrated by correlation value below 1.0 with noiseless data.

Figure 7(b) shows the rank correlation results for unbalanced noise. To deal with unbalanced noise we analyze the input data and pass a weight parameter to the methods (except for `Barycentric`). We use a simple but effective weighting scheme: compute the relative proportion of 1s and 0s in input matrix, namely $p_1$ and $p_0$, where $p_1 + p_0 = 1$. If the weight for a 0-to-1 flip is 1, then the weight of a 1-to-0 flip is $\log(p_1)/\log(p_0)$.

Unlike other methods, `Barycentric` is unaffected by the nature of noise (balanced or unbalanced), and does not need a weight parameter. Once the appropriate weight parameter is analyzed from input data, `Alternating` and Spectral methods perform very well. Until noise level 0.6 these methods are the best at recovering original band-order. As it was the case with balanced noise, Hamiltonian methods have larger variance and difficulties at recovering the original order.

The following table shows the running times (ms) for the methods on a 3.1 GHz Intel machine. All synthetic matrices have 50% fill and 10% balanced noise.

| Runtime (ms) Matrix size | `Barycentric` (100 iterations) | `Alternating` (100 iterations) | Hamiltonian | Spectral | `Simulated Annealing` |
|---|---|---|---|---|---|
| $50 \times 50$ | 2 | 12 | 4 | 8 | 136000 |
| $200 \times 200$ | 24 | 150 | 120 | 170 | – |
| $800 \times 800$ | 240 | 2110 | 4850 | 12050 | – |
| $3200 \times 3200$ | 4250 | 44300 | 344800 | 1163000 | – |

The `Barycentric` and `Alternating` are the fastest methods, even though both perform 100 iterations. Their running times increase steadily along the matrix size, while the increase is more pronounced for Hamiltonian and Spectral methods. Overall, all these methods are fast and able to handle even larger matrices in reasonable time.

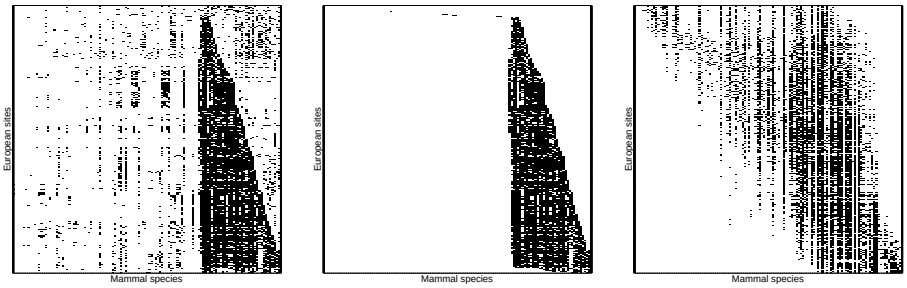
In the next sections we will present a series of real data applications. We will use the algorithms that are most appropriate for the data. Given the analysis in synthetic data these algorithms will be either `Alternating`, `Barycentric` or `Bidirectional Fixed Permutations` that uses the Spectral method and dot product for fixing the column permutation. Notice that even if `Simulated Annealing` could give good results, it is too time consuming for practical real data.

### 7.2 Mammals Data

The Mammal dataset consists of presence/absence records of European mammals: for 2179 cells of size $40 \times 40$ km$^2$ we have binary information about 124 different species. The full version of the Mammal dataset is available for research purposes upon request from the Societas Europaea Mammacologica (`www.european-mammals.org`).

Figure 8(a) shows the band obtained by the `Alternating` method that allows bidirectional flips with equal costs. The discovered structure of sites and mammals is interesting in itself: it shows a highly nested pattern from sites and species so far unknown in the dataset. Nested patterns have been widely studied in ecological applications because they give insight into the processes that govern the distribution of species. The correlation between the order of the sites and the temperature variable is 0.64, which is high for life-science data. Intuition is that the number of mammal species increases to some extent with temperature and few species live in cold conditions. Furthermore, of the 54155 total 1s in the dataset, 70% of them are accumulated within the borders of the band retrieved by `Alternating` algorithm (see Figure 8(b)). This means we found a dense structure. The matrix needs 23997 bidirectional flips to become a fully banded matrix.

As a comparison, Figure 8(c) depicts the band obtained by the `Barycentric` method. The bidirectional flips needed to make this matrix fully banded is 42041 flips. Nonetheless, the row ordering has high correlation with certain variables




(a) Mammals data by `Alternating` (b) Mammals data band by `Alternating` (c) Mammals data by `Barycentric`

**Fig. 8.** Reordered Mammals data. Left part (a) shows the data reordered by the `Alternating` algorithm, using equal-cost as a weighting scheme. Middle part (b) shows the borders of the band, that is all the 1s in original Mammals data that belong to the structure. In total the band includes 70% of all the 1s. Right part (c) shows the data as reordered by the `Barycentric` algorithm. The order of locations has strong correlations with latitude coordinates (0.92) and average temperatures (−0.85) of the locations.

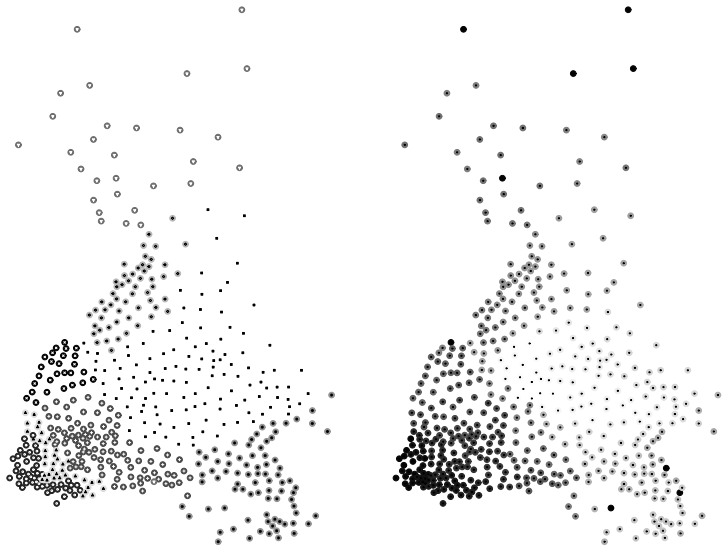
associated with the location, such as latitude coordinate (0.92) and average temperature (−0.85). It seems that both `Alternating` and `Barycentric` produce reasonable, yet visually different results here: `Alternating` prefers large dense clusters of 1s, whereas `Barycentric` assumes that all rows and columns belong to the banded pattern. Which shape is preferable depends on application. Note that assigning different weights for flips causes the `Alternating` method to change the shape of the band.

### 7.3 Dialect Data

The Dialect dataset [14], originally published in 1940, contains data about the usage of dialectical features in spoken Finnish language. The data is in binary form and represents 1334 phonological features and their usage in 506 municipalities.

The basic division of Finnish dialects has long remained static among linguistics. The division into two dialects results in Western and Eastern dialects; further divisions bring out more detail inside these two main dialects. The known division into eight dialect areas is shown in Figure 9(a).

We used both the `Alternating` algorithm and the `Barycentric` algorithm on the Dialect data to uncover a band structure. The best ordering found by the `Alternating` method after 1000 iterations is shown in Figure 10(a). It visually indicates a band where 65% of the original 1s belong to the banded pattern, despite the noise in the data. For comparison purposes, Figure 10(b) shows



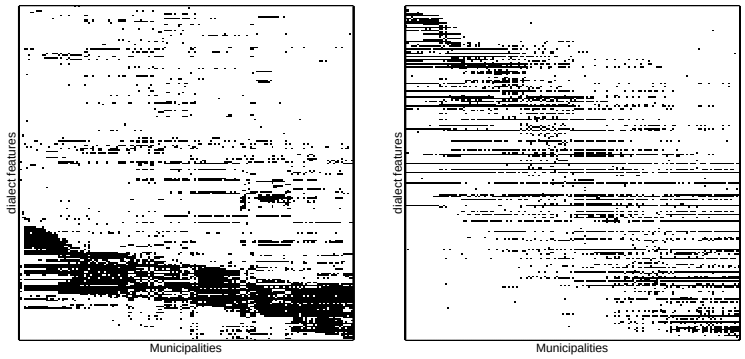
(a) Traditional division into 8 dialect areas

(b) Municipalities in band-order

**Fig. 9.** Left part (a) shows the traditional division of Finnish municipalities into eight areas by their dialectical features. Right part (b) visualizes the ordering of municipalities interpolated from the band discovered by the `Alternating` algorithm. The position of a municipality in band-order is depicted by its color, ranging from black to white. The ordering of the band has captured the main variation between Western and Eastern dialects.

the band of the `Barycentric` method. The two bands are very different: while `Alternating` aims at creating dense bands of 1s, the `Barycentric` method aims at creating wider banded patterns.

Interestingly, the order obtained for municipalities in the band from the `Alternating` algorithm can be interpreted as a crude estimate of the language spoken in a municipality: the order captures the variation from Western dialects to Eastern dialects. Indeed, the municipalities interpolated from the ordering of the band are plotted in Figure 9(b). We can see that this interpolation is very similar to the known division shown aside in Figure 9(a), despite some outliers. The outliers are explained by the small number of data points (1s) for those municipalities. Of course the dialectical variation cannot be fully described by a single ordering: some municipalities do not really fit anywhere in the linear order, for example those in Karelia. Note that the results show many geographically coherent dialect areas, although the algorithm did not have access to spatial information about the municipalities.




(a) `Alternating` band (b) `Barycentric` band

**Fig. 10.** Left part (a) shows the best band found by the `Alternating` algorithm on the Dialect data. This band is 54461 flips away from being fully banded. The right part (b) shows the band found by the `Barycentric` algorithm. This band is 105410 flips away from being fully banded.

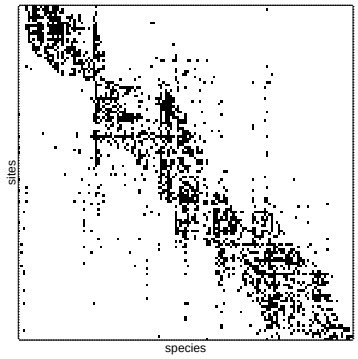



**Fig. 11.** Paleontological data reordered by `Alternating` method that assumes a weighting scheme where the cost of a 1-to-0 flip is four times that of a 0-to-1 flip.

### 7.4 Paleontological Data

The Paleontological data contains information of fossils genera in Europe [15][1]. Columns correspond to genera and rows correspond to sites. There are 124 sites and 139 genera. The banded structure found in the Paleontological dataset is shown in Figure 11. The result is given by the `Alternating` method.

[1] From August 31, 2007, the version of the dataset selects all columns with at least 10 occurrences and all rows with at least 10 genera present.

As is common with data gathering in real life, the paleontological data has more missing 1s than false 1s. Therefore 0-to-1 flips should be preferred over 1-to-0 flips. In general it is difficult to decide for a real life application which are the best weights, and many criteria for "correct" band-shape can be given. On this data, we decided to give to 1-to-0 flips a weight that is four times larger that that of a 0-to-1 flip, as it matches with our estimation of error rates.

### 7.5 DNA Amplification Data

The DNA amplification data is available upon request from the authors in [26]. It contains information on the DNA copy number amplifications recorded in 4590 cases (rows) and 393 band specific chromosomal locations (columns). A 0 denotes no DNA copy number amplification in the corresponding chromosomal location, and a 1 denotes a finding in the DNA copy number amplification in the location. More than this we also have available the neoplasm labels (cancer label) coupled to the cases of the matrix. The goal is to investigate DNA amplifications in different neoplasms types. This justifies a solution where different subsets of columns and rows can be evaluated separately, so we will run our `Iterative row-column elimination` algorithm. We use the performance measures of accuracy and recall as a deletion criteria of rows and columns, while allowing a maximum of 100 flips in the retrieved submatrices.

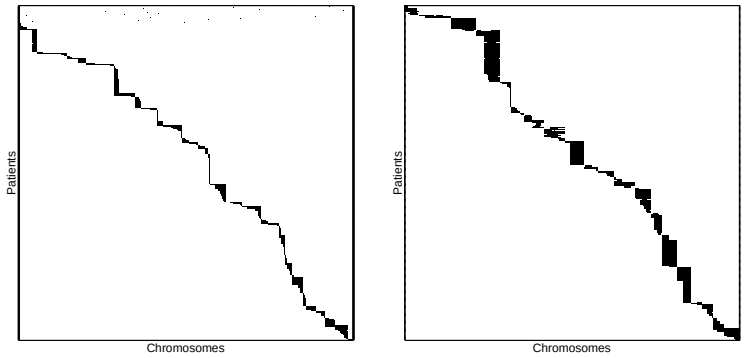
Figures 12(a) and 12(b) show the two retrieved submatrices (from the original data) found by accuracy and recall. For both submatrices we observe immediately that they exhibit a very clean banded structure (almost zero noise outside the band). They both preserve nicely the column permutation of the chromosomes, and moreover, each one of these submatrices identifies a different reduced collection of cancer types.

## 8 Related Work

The property of consecutive ones in binary matrices is particularly important in presence/absence data of paleontological applications [29]. The question of determining how far away a given matrix is from satisfying the consecutive-ones property corresponds to counting Lazarus events. This is also known as the seriation problem, to which Spectral ordering has been shown to produce good approximations [3].

Another important ecological concept related to the structure of binary matrices is nestedness [23]. A dataset is nested if for all pairs of rows one row is either a superset or subset of the other. From this definition it is direct to see that all nested matrices are banded, although the reverse does not hold. Bandedness is therefore a generalization of these ecological concepts.

A fully banded binary matrix is also known in some contexts as the monotone consecutive arrangement of ones [32]. Then, we can establish a connection between banded matrices and zero-partitionable matrices from [20]. We say that




(a) (b)

**Fig. 12.** Two submatrices from the DNA data. On the left (a), the submatrix with 1402 rows and 282 columns, using accuracy as the elimination criteria. On the right (b), the submatrix of 291 rows and 244 columns, using recall as the elimination criteria. Permutations on the columns were fixed by the spectral ordering with dot product similarity. The number of flips to obtain a full band is at most 100.

a binary matrix is zero-partitionable if the rows and columns can be independently permuted such that each 0 can be labeled with R or C in such a way that every position to the right of an R is an R and every position below a C is a C.

Therefore, any matrix satisfying the consecutive ones property will be always zero-partitionable, we only need to list rows in nondecreasing order of the position of their left most ones. The inverse statement is not true. This establishes the following hierarchy between classes of matrices,

$$\mathrm{N} \subset \mathrm{B} \subset \mathrm{SC1P} \subset \mathrm{C1P} \subset \mathrm{Z}$$

where N are nested matrices, B are banded matrices, SC1P are simultaneously consecutive-ones on row and columns, C1P are consecutive-ones matrices on rows, and Z are zero-partitionable matrices.

We cannot ignore the relation of banded matrices with the field of numerical analysis [4, 13, 31]. The solutions there also find good permutations to confine non-zero entries to the main diagonal band. Noisy data is, however, a problem for them, as the algorithms require that all 1s must be close to the main diagonal. The approach in this paper allows banded structures outside the main diagonal.

In data mining, bands are closely related to reorderable matrices, see e.g. [8, 22]: this is a simple visualization method to explore multivariate or multidimensional data; the basic principle is to transform a multidimensional data set into a 2D interactive graphic. The graphical presentation of a data set closely resembles the underlying matrix in that it contains rows and columns which can be permuted, allowing different views of the data set. The problem of reorderable matrices can be applied to real-valued data and to binary data. The property of

bandedness can be seen as a subcase of reorderable matrices where we only have binary values and the visualization must satisfy the band constraints. Therefore, our contribution offers new algorithms and theoretical properties for a special case of reorderable matrices.

Finding a band structure is closely related to biclustering problems [21] and subspace clustering [25, 28]: the goal is to identify groups of columns and rows that exhibit similar value patterns (for example, similar expression patterns in the case of microarray data). Many combinatorial problems have been studied around biclustering and the associated bipartite graph. The most related one to the one we study here is the edge editing problem, consisting in adding or removing the fewest edges from a bipartite graph so that it becomes a vertex disjoint union of complete bipartite graphs. However, the definitions are slightly different, as in our case the different biclusters (groups of columns and rows with ones) do not require to be vertex disjoint.

Finally, our approach can be see as a cluster ranking approach, as in [7]. Rather than simply partitioning a network into clusters, a cluster ranking algorithm also orders the clusters by their strength. In our problem, the different "clusters" or groups of rows are mapped into a one-dimensional rank provided by the columns of the same matrix. An important difference with those approaches is that we do not depend on parameters such as the number of clusters, which is a well studied problem in clustering categorical data [11].

## 9 Conclusions

We introduce the new concept of banded structures for binary data and characterize exactly its combinatorial properties. We first establish how row and column permutations affect the structure of the band and derive a polynomial test for bandedness. Based on the derived properties, we propose principled algorithms to approach two important combinatorial problems related to bands: minimum banded augmentation and maximum banded submatrix.

A first set of algorithms relies on fixing a column permutation beforehand. We then lift this column permutation requirement and propose two new algorithms that search for both column and row permutations at the same time. We show in synthetically generated data that our proposals are able to find banded structures efficiently.

Experiments on real data show that bands potentially occur in many applications, and here we concentrate on datasets from life sciences and a word dialect data. For two of the datasets, we discover previously unknown bands that have natural interpretations in the final order of rows and columns. Our results suggest that a hierarchy of mammals exists in Mammals occurrence data. For the word dialect data we have discovered that the band order has captured the main variation between Eastern and Western dialects of the Finnish language. In bandedness real life binary data is given a combinatorial treatment so as to uncover hidden structure in the data.

## References


1. R. Agrawal, T. Imieliński, and A.Swami. Mining association rules between sets of items in large databases. In *SIGMOD'93*, pages 207–216, 1993.
2. F. Alizadeh, R. M. Karp, L. A. Newberg, and D. K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Algorithmica*, 13(1/2):52–76, 1995.
3. J. Atkins, E. Boman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM J. Comput.*, 28(1):297–310, 1999.
4. C. Aykanat, A. Pinar, and U. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25(6):1860–1879, 2004.
5. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
6. A. Banerjee, C. Krumpelman, J. Ghosh, S. Basu, and R. Mooney. Model-based overlapping clustering. In *KDD'05*, pages 532–537, 2005.
7. Z. Bar-Yossef, I. Guy, R. Lempel, Y. S. Maarek, and V. Soroka. Cluster ranking with an application to mining mailbox networks. *Knowl. Inf. Syst.*, 14(1):101–139, 2008.
8. J. Bertin. Graphics and graphic information processing. pages 62–65, 1999.
9. K. S. Booth. *PQ-tree algorithms*. PhD thesis, 1975.
10. P. Burzyn, F. Bonomo, and G. Durán. NP-completeness results for edge modification problems. *Disc. Appl. Math.*, 154(13):1824–1844, 2006.
11. K. Chen and L. Liu. "Best k": critical clustering structures in categorical datasets. *Knowl. Inf. Syst.*, 20(1):1–33, 2009.
12. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill, 1990.
13. E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th ACM national conference*, pages 157–172, 1969.
14. S. Embleton and E. Wheeler. Computerized dialect atlas of Finnish: Dealing with ambiguity. *Journal of Quantitative Linguistics*, 7(3):227–231, 2000.
15. M. Fortelius. Neogene of the old world database of fossil mammals (NOW). http://www.helsinki.fi/science/now/, 2008.
16. G. Garriga, E. Junttila, and H. Mannila. Banded structure in binary matrices. In *KDD'08*, pages 292–300, 2008.
17. M. Girvan and M. Newman. Community structure in social and biological networks. *Proc. National Academy of Sciences USA*, 99(12):7821–7826, 2002.
18. S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
19. D. Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. ACM, New York, NY, USA, 1993.
20. I.-J. Lin and D. B. West. Interval digraphs that are indifference digraphs. In *Graph theory, Combinatorics, and Algorithms*, pages 751–765, 1992.
21. S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.
22. E. Mäkinen and H. Siirtola. Reordering the reorderable matrix as an algorithmic problem. In *Diagrams '00: Proceedings of the First International Conference on Theory and Application of Diagrams*, pages 453–467. Springer-Verlag, 2000.

23. H. Mannila and E. Terzi. Nestedness and segmented nestedness. In *KDD'07*, pages 480–489, 2007.
24. R. M. McConnell. A certifying algorithm for the consecutive-ones property. In *SODA'04*, pages 768–777, 2004.
25. G. Moise, A. Zimek, P. Kröger, H. Kriegel, and J. Sander. Subspace and projected clustering: experimental evaluation and analysis. *Knowl. Inf. Syst.*, 21(3):299–326, 2009.
26. S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25:7324–7332, 2006.
27. C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16:263–270, 1976.
28. L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, 2004.
29. K. Puolamäki, M. Fortelius, and H. Mannila. Seriation in paleontological data using Markov chain Monte Carlo methods. *PLoS Computational Biology*, 2, 2006.
30. F. S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory*, pages 139–146, 1969.
31. R. Rosen. Matrix bandwidth minimization. In *ACM national conference*, pages 585–595, 1968.
32. M. Sen and B. K. Sanyal. Indifference digraphs: A generalization of indifference graphs and semiorders. *SIAM Journal on Discrete Mathematics*, 7(2):157–165, 1994.
33. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.
34. P. Tian, J. Ma, and D. Zhang. Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism. *European Journal of Operational Research*, 118(1):81–94, 1999.
35. A. Tucker. A structure theorem for the consecutive 1's property. *Journal of Combinatorial Theory, Series B*, 12(2):153–162, 1972.