Department of Computer Science
Series of Publications A
Report A-2011-2

Patterns in permuted binary matrices

Esa Junttila

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium 13, University Main Building, on August 10th, 2011, at noon.

University of Helsinki
Finland

Supervisors

Prof. Heikki Mannila, Aalto University, Finland
Dr. Petteri Kaski, Aalto University & HIIT, Finland

Pre-examiners

Prof. Toon Calders, Eindhoven University of Technology, The Netherlands
Prof. Dimitrios Gunopulos, National and Kapodistrian University of Athens, Greece

Opponent

Prof. Matti Nykänen, University of Eastern Finland

Custos

Prof. Hannu Toivonen, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: postmaster@cs.helsinki.fi
URL: http://www.cs.Helsinki.fi/
Telephone: +358 9 1911, telefax: +358 9 191 51120

ISSN 1238-8645
ISBN 978-952-10-7116-4 (paperback)
ISBN 978-952-10-7117-1 (PDF)
Computing Reviews (1998) Classification: F.2.2, G.2.1, I.2.6, I.5.2, I.5.1
Helsinki 2011
Helsinki University Print

Patterns in permuted binary matrices

Esa Junttila

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
Esa.Junttila@alumni.helsinki.fi
http://www.cs.helsinki.fi/esa.junttila/

PhD Thesis, Series of Publications A, Report A-2011-2
Helsinki, July 2011, 155 pages
ISSN 1238-8645
ISBN 978-952-10-7116-4 (paperback)
ISBN 978-952-10-7117-1 (PDF)

Abstract

Reorganizing a dataset so that its hidden structure can be observed is useful in any data analysis task. For example, detecting a regularity in a dataset helps us to interpret the data, compress the data, and explain the processes behind the data. We study datasets that come in the form of binary matrices (tables with 0s and 1s). Our goal is to develop automatic methods that bring out certain patterns by permuting the rows and columns.

We concentrate on the following patterns in binary matrices: consecutive-ones (C1P), simultaneous consecutive-ones (SC1P), nestedness, k-nestedness, and bandedness. These patterns reflect specific types of interplay and variation between the rows and columns, such as continuity and hierarchies. Furthermore, their combinatorial properties are interlinked, which helps us to develop the theory of binary matrices and efficient algorithms. Indeed, we can detect all these patterns in a binary matrix efficiently, that is, in polynomial time in the size of the matrix.

Since real-world datasets often contain noise and errors, we rarely witness perfect patterns. Therefore we also need to assess how far an input matrix is from a pattern: we count the number of flips (from 0s to 1s or vice versa) needed to bring out the perfect pattern in the matrix. Unfortunately, for most patterns it is an NP-complete problem to find the minimum distance to a matrix that has the perfect pattern, which means that the existence of a polynomial-time algorithm is unlikely.

To find patterns in datasets with noise, we need methods that are noise-tolerant and work in practical time with large datasets. The theory of binary matrices gives rise to robust heuristics that have good performance with synthetic data and discover easily interpretable structures in real-world datasets: dialectical variation in the spoken Finnish language, division of European locations by the hierarchies found in mammal occurrences, and co-occuring groups in network data.

In addition to determining the distance from a dataset to a pattern, we need to determine whether the pattern is significant or a mere occurrence of a random chance. To this end, we use significance testing: we deem a dataset significant if it appears exceptional when compared to datasets generated from a certain null hypothesis. After detecting a significant pattern in a dataset, it is up to domain experts to interpret the results in the terms of the application.

Computing Reviews (1998) Categories and Subject Descriptors:
F.2.2 Analysis of algorithms and problem complexity: Pattern matching
G.2.1 Discrete mathematics: Combinatorial algorithms, Permutations and combinations
I.2.6 Artificial intelligence: Backtracking, Dynamic programming, Heuristic methods
I.5.2 Pattern recognition: Pattern analysis
I.5.1 Pattern recognition: Statistical and Structural Models

General Terms:
algorithm, theory, experimentation

Additional Key Words and Phrases:
binary matrix, permutation, nestedness, bandedness, consecutive ones, significance testing

Acknowledgements

In my past, people around me figured that I would start pursuing an academic degree. My short-term plans for life, however, did not allow me to think about the future too much at that time. In hindsight, it must have been seeking, gaining, and sharing knowledge that attracted me.

Pursuing a doctoral thesis turned out to be like nothing I had ever done before. In a nutshell, the job involves several years' worth of thinking, reading, writing, editorial work, presentations, graphical designs, slow-paced planning, fast-paced deadlines, traveling, frustration, and thinking it all over again. Being a doctoral student is the most versatile job I can imagine.

In the following I break down the factors that contributed to my work.

Advisors. When I expressed my interest towards research, Dr. Marko Salmenkivi recruited me as a research assistant in 2005. He introduced me to the job and arranged a doctoral student position under the supervision of Prof. Heikki Mannila in 2007. In late 2009, Dr. Petteri Kaski took charge of my supervision and helped me to finish this thesis. Every advisor has unique strenghts and weaknesses, and I was glad to have three advisors to learn from. Watching their example taught me the way of working in this scientific world.

Colleagues. I did not do all the work myself. In addition to the advisors, the credit also goes to my co-authors Gemma C. Garriga and Evimaria Terzi, who also taught me a great deal about research. As far as finishing the text goes, I got useful comments from Marina Kurtén, Pauli Miettinen, Pekka Parviainen, Teppo Niinimäki, Jarkko Toivonen, Jefrey Lijffijt, Katja Junttila, Matti Järvisalo, and Markus Ojala. Thanks to Pauli and Pekka, in particular, for taking their time to help me during all these years.

Funding. I received funding for my doctoral studies from the Department of Computer Science, Helsinki Graduate School in Computer Science and

Engineering (Hecse), Helsinki Institute for Information Technology (HIIT), and Algoritmic Data Analysis Centre-of-Excellence (Algodan), to name a few. I got everything a doctoral student could hope for: the Finnish political climate promoted doctoral studies, Hecse granted me a four-year position, the department provided money for traveling, top-class researchers around supported me, and the competent staff at the department ensured a stable working environment. An additional office at the Aalto University did not hurt either.

Atmosphere. To make a working day meaningful, I felt that discussing random topics with people was necessary. Thanks to the lunch group, members past and present, for providing unrelated topics for discussion and allowing me to think aloud. I also want to emphasize the roles of Jaana Wessman, Niina Haiminen, and Pauli as prīmī motoris for arranging Perjantai events.

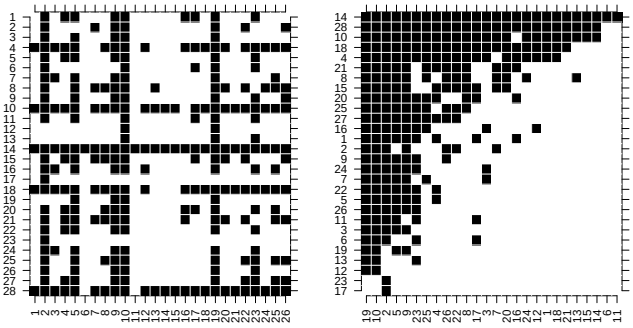
Helsinki, June 15th, 2011

Contents

Chapter 1

Introduction

Reorganizing a dataset so that its hidden structure can be observed is useful in any data analysis task. Consider for example the dataset in Figure 1.1(a). It has 28 locations in the Rocky Mountains (rows) and 26 mammals (columns), with presence/absence of a mammal at a location indicated by black/white. We observe that the dataset appears to have some structure, but its precise nature is perhaps not immediate. However, by permuting the rows and columns of the dataset, we observe in Figure 1.1(b) a slightly noisy pattern that describes a hierarchy of the occurrences of the mammals.



(a) Original dataset (b) Reordered dataset

Figure 1.1: Original dataset (a) and with reordered rows and columns (b).

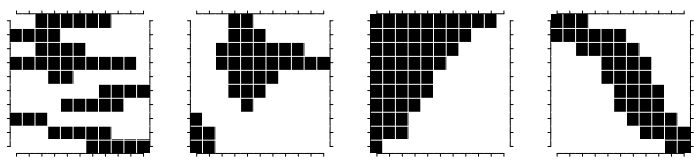
By using automatic reorganization to discover a specific structure, we can increase the interpretability of the data, compress the data, and better understand the processes that generated the data. The challenge is to identify meaningful structures that occur in datasets and develop algorithms that reveal these patterns automatically by reorganizing the datasets—also if the data contains noise and errors.

In this thesis we concentrate on datasets that come in the form of binary matrices; that is, 2-dimensional tables that contain only 0s and 1s. We use black squares in illustrations to indicate 1s. This type of data occurs frequently in, for example, ecology, paleontology, graph theory, interaction and social networks as well as in market basket analysis of retail companies. Binary data is versatile enough for complex patterns to emerge as well as simple enough to allow a combinatorial view that yields efficient algorithms.

We study patterns that arise by permuting the rows and the columns of a binary matrix. Note that a permutation of a data matrix does not change the contents of the data, but simply reorganizes it. Detecting these reorderable patterns in the matrices is foremost a type of permutation problem.

Patterns come in two varieties, global and local: global patterns span the whole dataset, whereas local patterns can be observed only in some parts of the data. We address both types of pattern, but the emphasis is on global patterns.

As depicted in Figure 1.2, we concentrate on the following patterns and their automatic detection in binary matrices: a *consecutive-ones (C1P)* matrix (a) describes data in which a total order on the columns establishes contiguous patterns; in a *simultaneous consecutive-ones (SC1P)* matrix (b) the contiguous patterns occur on both rows and columns; in a *nested* matrix (c) the rows and columns form a hierarchy while a *k-nested* matrix consists of several independent nested patterns; in a *banded* matrix (d) the attributes (rows or columns) show overlapping variation. These patterns have interlinked combinatorial properties and they allow a view on the general class of reorderable patterns.



(a) C1P (b) SC1P (c) Nested (d) Banded

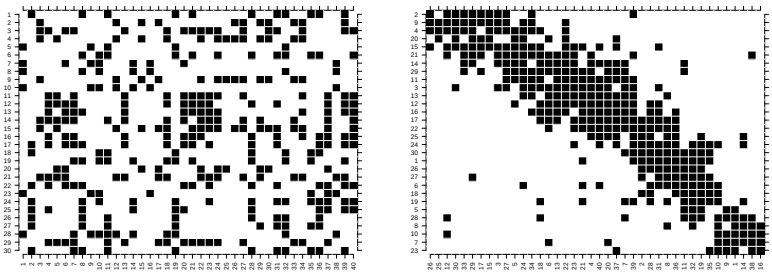
Figure 1.2: Example patterns in binary matrices

Developing an automatic process for finding useful hidden patterns in large datasets falls within the realm of *data mining*. In data mining we combine both statistics and computer science to ensure that extracted patterns are relevant and that we can handle large datasets in practical time. The need for new data mining methods is warranted, as the number of large datasets that are electronically available keeps increasing. The concept of

reorderable patterns has so far been used in data mining mainly as a simple visualization method to explore multivariate or multidimensional data. The basic principle is to transform a data set into a 2D interactive graphic [Ber99, MS00], which allows a different view on the dataset.

The history of reorderable matrices in data analysis can be traced back at least to the 19th century and the work of Petrie, who applied reordering techniques to study archeological data [Lii10]. Since then, several reordering methods have been introduced to study particular patterns in real-world data matrices.

Real-world datasets often contain errors, and then the goal is to detect whether the data is relatively close to a pattern. For example, in Figure 1.3(a) the matrix seems random, but after reordering its rows and columns, we observe on the right (b) a pattern that is *almost* banded.



(a) Original dataset (b) Reordered dataset

Figure 1.3: Original dataset (a) and with reordered rows and columns (b). We observe an almost banded pattern and noise.

When a dataset contains noise, the computational task of finding patterns becomes harder, which means increased demands of time or memory. The purpose of computational complexity analysis is to evaluate how much time and memory an algorithm needs and how the algorithm scales with larger problem instances. Roughly, an algorithm with a polynomial time complexity is useful in practice, whereas exponential time is a hindrance. Unfortunately, permutation problems are typically hard, and we need to settle for heuristics and approximations to manage large datasets. Nevertheless, before we can develop practical algorithms we need to study the combinatorial properties of the patterns.

For the majority of the patterns presented in this thesis, we can detect a perfect reorderable pattern in polynomial time in the size of the input matrix. If the data contains errors, then the problem becomes that of finding a closest matrix that has a perfect pattern, which can be solved

in polynomial time for some patterns. In many cases, however, finding a closest matrix is an NP-hard problem, which means that the existence of a polynomial-time algorithm is unlikely.

In summary, this thesis is a combination of data mining, theory on structures in binary matrices, algorithm design and analysis, statistical significance testing, and real-world applications. We study patterns in binary matrices, formulate the patterns exactly, and conduct a theoretical study on the combinatorial properties of the patterns. The combinatorial results are then used to develop algorithms—both exact and heuristic—that are able to find these patterns in real-world datasets automatically. We need new statistical tools to assess whether these patterns are statistically significant, such as testing the data against a specific null model. If the methods find a significant and strong pattern in a real-world dataset, it is up to domain experts to interpret the result and its meaning so that they can better understand the data and the underlying processes that generated it.

The experiments in the further chapters demonstrate that these patterns exist in a wide range of real-world datasets. For example, we used the heuristic algorithms introduced in this thesis to detect variation in the dialects of the spoken Finnish language, multi-layered hierarchies of European mammals, an ordering for excavation sites in a paleontological dataset by age, and co-appearing groups of characters in the novel Les Misérables. The experiments on synthetic data verify that the new data mining algorithms extract patterns from synthetic binary matrices reliably. The methods tolerate high levels of noise and they handle matrices with thousands of rows and columns in practical time.

1.1 Organization

We present the basic terminology and background of reorderable patterns and related problems in Chapter 2, and introduce the error models and statistical methods used throughout this text. Starting from the simplest pattern, we introduce the concept of consecutiveness and its algorithms in Chapter 3. Nested patterns, their properties and algorithms will be reviewed in Chapter 4, and the statistical significance of nested patterns is discussed in Chapter 5. Chapter 6 generalizes nestedness into a more general pattern, segmented nestedness, and contains algorithms for finding a good partition of data into k nested patterns. The pattern of bandedness, its applications and several dedicated algorithms are introduced in Chapter 7 in the context of data mining. Finally, the connections between these patterns and further topics are gathered into Chapter 8.

1.2 Contributions

The content of this thesis is based on published articles [GJM08, GJM11, JK11], unpublished manuscript [JTM10], as well as other unpublished materials. None of the new results in these articles has been included in a PhD thesis so far. In addition, a lot of related work by others has been included in this thesis for completeness. We refer to the following people (me and the co-authors) by their family name: Esa Junttila, Heikki Mannila, Gemma C. Garriga, Petteri Kaski, and Evimaria Terzi. In the following we break down the main contribution of each article and then review the new unpublished results chapter by chapter.

Contributions in the articles. The articles on bandedness [GJM08, GJM11] introduced the concept of banded patterns in binary matrices. New combinatorial results led to a polynomial-time test for bandedness and to the first set of heuristic algorithms for detecting almost banded patterns in real-world datasets. Experimental results showed that the algorithms are fast enough in practice, and almost banded patterns occur in a wide range of real-world applications. Garriga, Junttila, and Mannila conceived the main ideas together. Junttila conducted the experiments.

The article on segmented nestedness [JK11] built on the concept of nestedness, and generalized it into k-nestedness: several nested patterns in data. We extended the theoretical foundation of the concept and gave a polynomial-time algorithm for recognizing k-nestedness. We showed that the problem of finding a closest k-nested matrix is NP-hard, and gave several heuristic algorithms for the problem. In addition, an MDL-based method was developed to choose k automatically. Experimental results indicated that the methods are noise-tolerant, in particular the one with SVD, and they find k-nested patterns reliably. The results on real-world datasets showed k-nested patterns that were easy to interpret. Junttila and Kaski developed all new results together and Junttila conducted the experiments.

The unpublished manuscript [JTM10] analyzed the performance of nestedness measures in terms of significance testing and noise-tolerance. We studied the concept of nestedness from the viewpoint of ecological processes and datasets. We introduced a new noise-tolerant method for measuring nestedness in ecological datasets and conducted experiments that revealed that popular nestedness methods have severe shortcomings in how they handle noise. Mannila and Terzi conceived the idea of comparing nestedness methods, Junttila and Mannila studied the topic further and Junttila conducted the experiments.

New contributions in this thesis. The development of the knowledge discovery process presented in this thesis is not a part of any particular section, but a composite of general techniques for extracting interesting patterns from data. Therefore describing the process counts as a contribution: from identifying an interesting pattern to using specific algorithms to detect the pattern in data.

Chapter 2 provides the common theoretical background to the subsequent chapters. It has a general framework for reorderable patterns, an error model for reliably testing the performance of data mining algorithms on synthetic data, the relations between patterns in matrices and graphs, and a general-purpose algorithm for finding submatrices that have a given pattern. Junttila, Mannila, and Garriga conceived these ideas together.

Chapter 3 is essentially a survey of previous work on consecutiveness. New material includes an algorithm for finding a closest matrix that has direct C1P and two new algorithms that solve the minimum distance problem exactly. Junttila developed the new materials.

Chapter 4 presents a survey of previous work on nestedness and new unpublished results. It brings together the main results from both matrix and graph theory that are related to nestedness. The new material consists of polynomial-time algorithms for detecting nestedness, in particular a new sublinear recognition algorithm and a new polynomial-time algorithm for finding a closest directly nested matrix. For the problem of finding a closest nested matrix this chapter includes new approximation and lower bound algorithms, a minor generalization of an existing algorithm, and two new exact algorithms. It also has a groundwork on the problem of finding almost nested submatrices, including a correction of an NP-completeness proof. Junttila developed the new results; the idea for Algorithm 10 is from Terzi.

Chapter 5 is based on the unpublished manuscript [JTM10] on nestedness analysis, with only minor changes to the original text.

Chapter 6 is essentially the published article [JK11] on segmented nestedness, with only minor changes to the original text.

Chapter 7 is based on the articles [GJM08, GJM11] on bandedness. In addition the following unpublished material has been included: connections from bandedness to other patterns in binary matrices; extended related work and different characterizations of bandedness in terms of graph-theoretical concepts; many proofs have been rewritten and extended for clarity; a new NP-hardness result (Theorem 7.17) has been written down; a new heuristic algorithm (Section 7.5.3) has been developed; Section 7.9 gives new exact algorithms for finding a closest banded matrix. Junttila developed the new results; the idea for the proof of Theorem 7.17 is from Garriga.

Chapter 2

Preliminaries

In the following we give the basic terminology used throughout this thesis. We then introduce the concept of a reorderable pattern, and present the problems related to recognizing the pattern and finding how far a matrix is from the pattern. We examine the connection between patterns in graphs and their counterparts in matrices. We provide tools for evaluating how good algorithms are in finding patterns in data and how to assess the significance of these patterns. Finally, we address the problem of finding from data such submatrices that have a small distance to a reorderable pattern.

2.1 Binary matrices

We first introduce the notation and basic concepts used in this thesis. For future reference, see the index and the table on pages 153–155.

We denote an n-dimensional vector by $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, and by the *zero vector* (*zero matrix*) we refer to a vector (matrix) that is full of 0s. An $m \times n$ matrix A has m rows and n columns, and the entry on the ith row and jth column is denoted $a_{i,j}$. A *submatrix* of A only includes subsets of the rows and columns in A and may use a different ordering for included rows and columns. We write $\{1, 2, \ldots, k\}$ as $[k]$.

By a *binary* value, we mean that the value is either 0 or 1. If a vector or a matrix is binary, we mean that it contains only binary values. We illustrate (large) binary matrices by drawing the 1-entries as black and 0-entries as white as in Figure 1.1.

In a binary dataset, the values can be interpreted in many ways. For example, a binary value 1 can be interpreted as truth value true, as a directed edge in a graph, as an indicator whether a mammal species occurs in a location, or just as a plain number.

Matrices with binary values occur in many different applications. A typical example is market-basket data gathered by retail companies [AIS93]. Also interactions and social networks can be represented as binary matrices. Furthermore, binary matrices abound in a large variety of fields ranging from information retrieval (documents and occurrences of words) [BYRN99], to bioinformatics and computational biology (genes and probes) [AKNW95, MHB+06], or ecology and paleontology (sites and occurrences of species) [ABH98, PFM06].

Detecting patterns in a data matrix is essential for gaining an understanding of the underlying processes that generated the data. For binary matrices in particular, it is the nature of relations between rows and columns that must be understood. Detecting patterns in an error-free data is computationally easier than searching in erroneous data, in which patterns are incomplete.

It is often convenient to convert a binary matrix problem into a graph problem, or vice versa. Given a directed graph $G = (V, E)$ that is unweighted and has no duplicate edges, we can describe G by a $|V| \times |V|$ binary adjacency matrix A as follows: $a_{i,j} = 1$ if and only if $(i, j) \in E$. For an undirected bipartite graph $G = (R \cup C, E)$, where $R \cap C = \emptyset$ and $E \subseteq R \times C$, we can use a more compact representation for the adjacency matrix A that has $|R|$ rows and $|C|$ columns: for vertices $r \in R$ and $c \in C$, we have $a_{r,c} = 1$ if and only if $\{r, c\} \in E$.

In later sections, we use frequently the *set interpretation* of the rows and columns in a binary matrix A. In these cases, we denote by C_j the set of row indices where the column j has 1s, that is, $C_j = \{i \mid 1 \leqslant i \leqslant m \text{ and } a_{i,j} = 1\}$. We use a similar notation for rows: R_i is the set of column indices where row i has 1s.

For an exact definition of permutations of vectors and matrices, consider the following. A *permutation* π on $[k]$ is a bijection $\pi\colon [k] \to [k]$. When permuting a vector $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ with π, we map the entry at position i to position $\pi(i)$ for all $i \in [k]$. For an $m \times n$ matrix, a row permutation σ is a permutation on $[m]$, and a column permutation τ is a permutation on $[n]$. When permuting a matrix with σ and τ, we map the entry at position (i, j) to the position $(\sigma(i), \tau(j))$ for all entries.

2.2 Matrix distance

Real-world matrices are not expected to be error-free—consider for example the noisy matrices in Figure 1.3. To be able to compare matrices in the presence of noise, we need to define a distance measure. We are interested

in (edit) distances between two binary matrices A and B that have the same dimensions. A *flip* refers to changing the value of one entry: either from 0 to 1 (a 0-to-1 flip) or from 1 to 0 (a 1-to-0 flip). We want to know the minimum number of flips needed to make A identical to B.

With the *augmentation distance* we refer to the minimum number of 0-to-1 flips so that A becomes B. Likewise, the *deletion distance* is the minimum number of 1-to-0 flips needed. Note that sometimes there is no way to convert A into B with only one type of flips. The *Hamming distance* is a combination of the two: the minimum number of flips when both 0-to-1 and 1-to-0 flips are allowed. To transform a matrix A into a matrix B, we need to flip all the entries that are nonzero in $A - B$; the Hamming distance is the number of such nonzero entries. All matrices have a finite Hamming distance (at most mn) to all other binary matrices with the same dimensions.

We use a *weighted distance* to handle a variety of distance measures. Assume that both matrices A and B have equal dimensions, and that a nonnegative weight $w_{i,j}$ is associated with each entry $a_{i,j}$. In general, the weight matrix W defines the distance measure used.

Definition 2.1 (Distance) *Let A and B be binary matrices and let W be a nonnegative matrix of weights for the entries of A. The* distance $d_W(A, B)$ *from A to B is $\sum_{i,j} w_{i,j} \cdot |a_{i,j} - b_{i,j}|$.*

For weighted distances, the distance is no longer the number of some type of flips, but the sum of weights on those flips. For example, the Hamming distance is captured by the weighted distance with a weight matrix full of 1s. On the other hand, the augmentation distance is captured by setting weight 1 for 0-entries in A, while 1-entries have a weight larger than mn. If such a distance from A to B is more than mn, then 0-to-1 flips alone are insufficient for the conversion. We denote the augmentation distance by d_{A}, the deletion distance by d_{D}, and the Hamming distance by d_{H}.

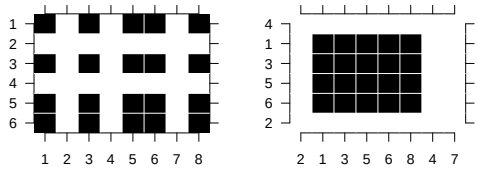
A weight matrix W with arbitrary weights can be used to define a more detailed entry-wise weighting scheme. For example, if a domain expert knows that a real-world dataset contains entries that have uncertain values, low weights should be set on these entries. The triangle inequality holds for arbitrary nonnegative weights W, but the augmentation and deletion distances are not symmetric.

The distance measure d_W used in this thesis is an instance of a more general edit distance: minimum number of elementary operations needed. Similar edit distances have been used earlier in the context of graphs and approximate string matching. In particular, many related edit problems for graphs are NP-complete [BBD06].

2.3 Reorderable patterns

Pattern recognition in binary matrices involves finding certain combinations of 0s and 1s from data. In this thesis we concentrate on reorderable patterns, which are patterns that may not necessarily be observable in the data matrix, but reordering the rows and columns can reveal them. This can be seen as a seriation problem [Lii10], which involves reordering data points in an order that satisfies certain meaningful patterns.

We use rectangular patterns to illustrate the concept of reorderable patterns. An $m \times n$ binary matrix A is *directly rectangular* if A is the zero matrix or there exist indices $i, k \in [m]$ and $j, l \in [n]$ such that $a_{r,c} = 1$ if and only if $r \in \{i, i+1, \ldots, k\}$ and $c \in \{j, j+1, \ldots, l\}$. In other words, the 1s in A form a rectangle. To illustrate this, the matrix in Figure 2.1(a) is not directly rectangular but 2.1(b) is.



(a) (b)

Figure 2.1: Original matrix (a) and with reordered rows and columns (b). The matrix (a) is not directly rectangular but (b) is.

We next describe the general framework for reorderable patterns in matrices. Consider a pattern $\mathcal{P}$ that describes a property that a binary matrix may have. Given the family $\mathcal{M}$ of all binary matrices, a *pattern* is formally a subset $\mathcal{P} \subseteq \mathcal{M}$, that is, a set of matrices that have the property. We say that a matrix A has the pattern $\mathcal{P}$ if $A \in \mathcal{P}$.

We concentrate on patterns that may emerge by reordering the rows, columns, or both in a matrix. Given a pattern $\mathcal{P}$, its associated *reorderable pattern* $\mathcal{R}(\mathcal{P})$ consists of all the matrices obtained from the matrices in $\mathcal{P}$ by permutation of the rows and columns. We say that a matrix A has the reorderable pattern $\mathcal{R}(\mathcal{P})$ if $A \in \mathcal{R}(\mathcal{P})$. The framework of reorderable patterns can also be applied to real-valued data. In the remainder of this thesis, whenever we talk about the distance from a matrix A to a pattern $\mathcal{P}$, we mean the distance from A to a closest matrix $B \in \mathcal{P}$.

As an example, consider the reorderable pattern that arises from directly rectangular matrices. A binary matrix A is *rectangular* if there exist permutations of the rows and columns such that the permuted A is directly

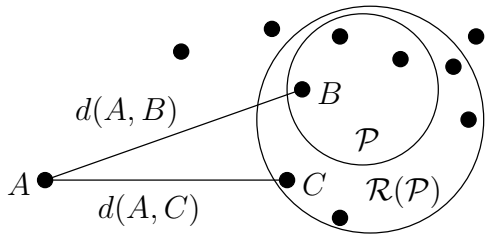


Figure 2.2: Illustration of Problems 2.2, 2.3 and 2.4. Each dot represents a matrix. From the perspective of A, a closest matrix that has pattern $\mathcal{P}$ (inner circle) is B, while a closest matrix that can be permuted into a matrix that has pattern $\mathcal{P}$ (outer circle) is C. In the RECOGNITION problem, we ask whether a given matrix A is in $\mathcal{R}(\mathcal{P})$, that is, inside the outer circle.

rectangular. We observe that both matrices in Figure 2.1 are rectangular—their only difference comes from different permutations.

We study three types of problems on a matrix A, a pattern $\mathcal{P}$, and the associated reorderable pattern $\mathcal{R}(\mathcal{P})$. They ask whether A has a reorderable pattern and what the minimum distance from A to the pattern is, with or without reordering. The problems are illustrated in Figure 2.2.

Problem 2.2 (RECOGNITION OF PATTERN $\mathcal{P}$) *Given a binary matrix A determine whether $A \in \mathcal{R}(\mathcal{P})$.*

Given our example pattern of directly rectangular matrices, we observe that a matrix is rectangular if and only if it is a zero matrix or there exist integers $r \in [m]$ and $c \in [n]$ such that (a) r rows have sum c, (b) c columns have sum r, and (c) all other row and column sums are 0s. In the language of bipartite graphs, the RECOGNITION problem asks us to determine whether a given graph is isomorphic to a graph that has pattern $\mathcal{P}$.

Problem 2.3 (CLOSEST MATRIX WITH DIRECT PATTERN $\mathcal{P}$) *Given a binary matrix A and a nonnegative weight matrix W, find a matrix $B \in \mathcal{P}$ that minimizes the distance $d_W(A, B)$.*

The word DIRECT in Problem 2.3 means that the pattern can be witnessed in matrix B without reordering the rows or the columns. Observe that B need not be unique, and if A already has $\mathcal{P}$, we have $A = B$ and $d_W(A, B) = 0$. As an example, consider the matrix in Figure 2.3(a); the matrix in Figure 2.3(b) is the d_{H}-closest matrix that is directly rectangular.

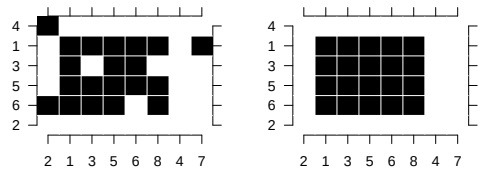


Figure 2.3: The matrix (a) is not directly rectangular, but (b) is. The matrix (b) is the d_{H}-closest such matrix to (a), with d_{H}-distance 6.

Problem 2.4 (Closest Matrix With Reorderable Pattern $\mathcal{P}$**)** *Given a binary matrix A and a nonnegative weight matrix W, find a matrix $B \in \mathcal{R}(\mathcal{P})$ that minimizes the distance $d_W(A, B)$.*

Problem 2.4 extends Problem 2.3 so that all permutations of the rows and columns are allowed to witness the pattern $\mathcal{P}$. It is Problem 2.4 that is more important from the viewpoint of data mining: ability to discover patterns that cannot be easily seen. As an example, consider the matrix in Figure 2.4(a); the matrix in Figure 2.4(b) is the d_{H}-closest matrix that is rectangular. When we use a specific distance measure, say W, we refer to the problems as d_W-Closest and likewise for d_{A}, d_{D}, and d_{H}.

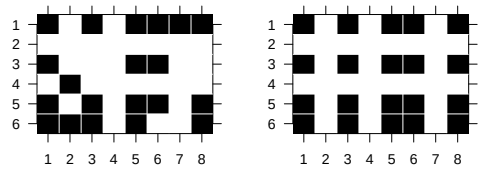


Figure 2.4: The matrix (a) is not rectangular, but (b) is. The matrix (b) is the d_{H}-closest such matrix to (a), with d_{H}-distance 6.

By identifying a reorderable pattern in a data matrix, we can reveal this structure in data by reordering the rows and columns, which only changes the organization of the data, but does not change the values in any way. Most patterns benefit from this reordering, as the pattern in question becomes easier to see by the human eye, and the permutation for the rows (columns) may give a meaningful interpretation on the dependencies between the rows (columns). This is also related to data visualization: reordering the data gives a way to describe the data from the viewpoint of the pattern in question.

Why are we interested in finding the distance from A to a closest matrix that has a pattern $\mathcal{P}$? The actual distance offers a way to estimate how strong the underlying pattern $\mathcal{P}$ is in A. In addition, if the phenomenon behind the data has the pattern $\mathcal{P}$, we can identify entries that are likely errors in the data A by checking the entries where A and B differ, where B is a closest matrix that has $\mathcal{P}$. In some cases, a closest matrix B may be a more accurate representation of reality than the erroneous data A is.

In reordering problems we need to find permutations under which the data matrix exhibits a pattern $\mathcal{P}$. If both rows and columns can be reordered, the number of permutations is $m!n!$. This makes brute-force algorithms useless in practice except for the smallest of matrices. To deal with larger matrices, we must develop an algorithm that exploits the structure found in matrices that have $\mathcal{P}$. Failing to do that, we settle for heuristics and approximations that *almost* solve the problem: there is no guarantee for optimality.

Considering Problems 2.3 and 2.4, the former is generally an easier problem, as it does not require reordering of the data. Given an algorithm `Algo` for Problem 2.3, we can solve Problem 2.4 heuristically, for example by using local search. Indeed, if we interpret the minimum distances obtained from `Algo` as energy values, we can use simulated annealing to find a permutation that (almost) minimizes the energy, which solves Problem 2.4. This approach is discussed in Section 7.7.3. If the reorderable pattern has a particularly easy characterization, Problem 2.4 may be easier than Problem 2.3. Consider a pattern $\mathcal{P}$ where the columns occur in nondecreasing order by their column sums: all matrices have $\mathcal{R}(\mathcal{P})$, but not all have $\mathcal{P}$.

For some patterns $\mathcal{P}$ there exists a *forbidden submatrix characterization*: a matrix A has a pattern $\mathcal{P}$ if and only if none of the submatrices of A belongs to the collection of forbidden submatrices. It follows that a pattern has such a characterization if and only if the pattern is closed under deletion of rows and columns (for every matrix in $\mathcal{P}$ also its submatrices are in $\mathcal{P}$), but the collection may be infinite. For example, in Section 4 we state that a matrix is nested if and only if none of its submatrices are switch boxes as in (4.2) on page 39. In the setting of bipartite graphs, this closure property is called *hereditary* and we will use it in the context of binary matrices. If the forbidden submatrices have bounded sizes, we can use them to develop recognition algorithms or heuristic algorithms for distance problems. Alas, in many cases the number of forbidden submatrices increases with the input size, or such a characterization does not exist in the first place.

In addition to searching for a global pattern that spans the whole dataset, we can also seek local patterns that consist only of a subset of the rows or

columns. For example, frequent itemsets [Goe03] refer to submatrices that form rectangles full of 1s, overlapping submatrices [JXFD08] approximate a dataset by a union of patterns, and maximally banded patterns [ABJ10] identify banded submatrices in a dataset.

2.4 Patterns in graphs

Many patterns in graphs can be described as patterns in binary matrices and vice versa. This allows combining results from both graph theory and matrix theory, which is useful for analytical studies and for designing efficient algorithms. We next show how the matrix patterns studied in this thesis translate into graph patterns, and how well-known graph patterns can be described conveniently as matrix patterns. The details of the matrix patterns (C1P, nestedness, bandedness) follow in the later chapters.

For example, the rectangular matrix in Figure 2.1 corresponds to the bipartite graph (bigraph) in Figure 2.5. Indeed, rectangles translate into bicliques; that is, complete bipartite subgraphs in bipartite graphs.

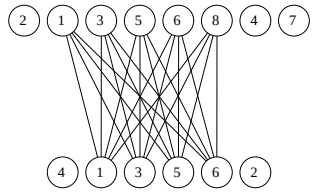


Figure 2.5: Rectangular matrices form bicliques in bipartite graphs.

Next we consider a well-known pattern in graphs, the recognition of which is an NP-complete problem [GJ79, Prob. GT39].

Problem 2.5 (Hamiltonian Path) *Given an undirected graph $G = (V, E)$, is there a path in which each vertex in V appears exactly once?*

We can study the Hamiltonian Path problem also as a matrix reordering problem. Given an undirected graph $G = (V, E)$, we construct a binary *incidence matrix* A, where the vertices correspond to the rows and the edges to the columns. We have $a_{v,e} = 1$ if and only if the vertex v is incident to the edge e. Each column in A contains exactly two 1s (the number of vertices incident to an edge), and a path in the graph translates into a *path pattern* of 1s, as seen below in (2.1). Now the graph G has a Hamiltonian path if and only if there exist permutations of the rows and

columns such that in the permuted A the first $|V|-1$ columns (edges) form a submatrix that has the path pattern.

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & & 0 \\ 0 & 1 & 1 & & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & & 1 \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.1}$$

An *interval graph* [LSW97] has a collection of intervals on the real line as its vertices; two vertices are joined by an edge if and only if the intervals intersect. Given a graph G, it is convenient to use matrices to recognize whether G is an interval graph [FG65]: we construct an incidence matrix of vertices and dominant cliques, and check whether this matrix has the C1P pattern, as described in Section 3.2.

Next we consider the pattern of zero-partitionable matrices [LSW97] and state its connection to directed interval graphs.

Definition 2.6 (Zero-partitionable) *A binary matrix A is said to be* zero-partitionable *if there exist permutations for the rows and columns such that each 0-entry in the permuted matrix can be labeled with R or C so that (a) every position to the right of an R is a 0-entry labeled with R, and (b) every position below a C is a 0-entry labeled with C.*

A *directed interval graph* is a generalization of interval graphs to directed graphs. Namely, a directed graph is a *directed interval graph* if and only if its adjacency matrix is zero-partitionable [Wes98].

In *unit interval graphs* all the intervals have the same length on the real line, and in *proper interval graphs* none of the intervals properly contains another. The adjacency matrices of these graphs have the *monotone consecutive arrangement* property [Wes98]. In fact, this graph-theoretical concept is identical to the definition of bandedness in Chapter 7. The following result establishes the connection between bandedness and bipartite interval graphs. Given a matrix A, the following three claims are equivalent [Wes98]: (a) A is the bipartite adjacency matrix of a unit interval bigraph, (b) A is the bipartite adjacency matrix of a proper interval bigraph, (c) A is banded. Furthermore, symmetric banded matrices have an interpretation as proper interval graphs [Rob69], and general banded matrices as proper interval digraphs [SS94].

Given a partially ordered set, a *chain* is a subset that describes a total order. A bipartite graph $G = (R \cup C, E)$ is a *chain graph* if there exists a permutation π of vertices in C such that for all vertices $r \in R$ the following holds: if r is adjacent to a vertex $c \in C$, then r is also adjacent to all vertices in C that succeed c in π. In other words, the graph describes a total order for its vertices. Nested matrices (Chapter 4) are equivalent to chain graphs [MT07, Yan81a].

The problems that involve distances on matrices have a natural interpretation on bipartite graphs. The Hamming distance on graphs is the minimum number of added and/or removed edges. Analogously, the augmentation (deletion) distance refers to the number of added (removed) edges, and a weighted distance has weights on all possible edges. For example, edge augmentation, deletion, and edit (Hamming) problems on interval graphs are NP-complete [BBD06].

2.5 Error models

We next describe the error models that are used in synthetic experiments that appear later in this thesis.

Since real-world datasets are rarely perfect, all algorithms developed to seek patterns in such datasets have to take into account the possibility of noise and errors that come from inaccuracies in measurements and human mistakes. To measure how reliably these algorithms find a pattern $\mathcal{P}$ from erroneous data, we create synthetic data generators that produce ground-truth matrices B, all of which have the pattern $\mathcal{P}$. We then obtain a data matrix A by adding errors to B according to an error model, and randomizing the row and column permutations. We conduct a set of experiments where the algorithms get the erroneous data A as input and they produce a matrix $\widehat{B}$ that approximates a closest matrix that has the reorderable pattern $\mathcal{P}$. Now we can compare $\widehat{B}$ to the ground-truth matrix B and evaluate how well the algorithms are able to retrieve the underlying pattern $\mathcal{P}$.

Real-world binary datasets often have two types of errors: *missing data* and *misclassifications*. In the former, some true 1s appear in data as 0s because of sampling bias, corrupted data, or insufficient research efforts, for example. In the latter, some true 0s appear in data as 1s, which is caused by, for example, faulty or inaccurate equipment and observational human errors. We simulate these errors by parameters Pr(1-to-0) and Pr(0-to-1) in the data generator, and obtain a dataset A by independently flipping the entries in the ground-truth matrix B according to these probabilities.

It is common that ecological presence/absence datasets have lots of miss-

ing data [NB07], yet misclassifications are rare. To imitate this in synthetic datasets, we use *asymmetric* noise, in which Pr(0-to-1) $\neq$ Pr(1-to-0). An alternative is to use *symmetric* noise, in which Pr(0-to-1) = Pr(1-to-0). Also more specific tests are possible by fine-tuning the entry-based probabilities according to additional data.

In later chapters we generate perfect data that has a pattern $\mathcal{P}$. In synthetic experiments we can use a benchmark method that uses the (original) permutations under which the ground-truth matrix B and its directly observable pattern were generated. Heuristic algorithms, of course, have no such information available. When we add increasing levels of noise to the data, however, the pattern vanishes. If this is the case, the original permutations are no longer the best with respect to the distance to $\mathcal{P}$, and other algorithms may find permutations that produce smaller distances (they fit better with the noisy data).

2.6 Significance testing

In *significance testing* the idea is to deem a value either significant or a mere occurrence of random chance. To do so, we need a process that, in our opinion, produces data that is realistic but non-significant with respect to the phenomenon under study. From the process we obtain a distribution of values against which we compare the observed value.

Consider an example of coin tossing where our null hypothesis asserts that the coin is fair. Assuming that the hypothesis is true, we obtain a probability distribution on flipping results. Given a result, such as 1 heads out of 5 flips, we may view the position of the result in the distribution, and consider the result ordinary. A less probable result, however, such as 100 heads out of 500 flips, suggests that the null hypothesis is likely incorrect.

We next describe the (one-sided) significance testing procedure in more detail. Given a null hypothesis H_0, it defines a probability distribution on the values under consideration; let X be a random variable that has this distribution. Given a value d, the (one-sided) *p-value*, denoted p, is the probability $\Pr(X \leqslant d)$. Assume we have selected a significance level α, for example $\alpha = 0.05$. If $p < \alpha$, we *reject* H_0 and say that the value d is *statistically significant* at level α under H_0.

We want to know whether a pattern $\mathcal{P}$ is significant in a matrix A. Let d be the distance from A to $\mathcal{P}$ and let H_0 be a null hypothesis that assumes that $\mathcal{P}$ does not exist in matrices. Then from H_0 we obtain a distribution of matrices and their distances to $\mathcal{P}$. By comparing d to the distribution of distances, we obtain a p-value, and we can assess the significance of $\mathcal{P}$ in A.

Unfortunately the probability distribution in the null hypothesis H_0 can be too complex to be described explicitly. Instead, we may employ a *null model*, that is, a method that produces random matrices for a given dataset under H_0—there is no need to describe the distribution explicitly. We lose exact p-values, but we can approximate them with *empirical p-values*: we sample random matrices R from the null model and count the proportion of the matrices R that have distance at most d.

In significance testing two types of error may take place. Given a null hypothesis H_0, a *Type I* error occurs when H_0 is rejected despite being true, whereas a *Type II* error occurs when H_0 is not rejected despite it being false. Typical null models are *conservative* in that they avoid Type I errors at the expense of having more Type II errors, which makes their statistical power weaker. A *lenient* method does the opposite: the hypothesis H_0 is rejected often, which risks detecting patterns that do not exist. The choice of a null hypothesis (or model) is critical and requires a deep understanding of the pattern and the application.

2.7 Patterns in submatrices

Sometimes a data matrix A does not have a pattern $\mathcal{P}$, but a submatrix of A does. Indeed, there may be several such local patterns in the data that are interesting in their own right. In this case it makes sense to look at large submatrices where the distance to the pattern $\mathcal{P}$ is small in some sense. We first introduce the problem where we search a submatrix of maximum size that has a perfect pattern, and then the problem that prefers large submatrices at the expense of weaker patterns.

Problem 2.7 (Maximum-Size Submatrix With $\mathcal{P}$) *Given a binary matrix A, find in A a submatrix that has pattern $\mathcal{P}$ and maximizes $a+b$, where a is the number of rows in the submatrix and b that of columns.*

In Problem 2.7 we want to remove from A as few rows and/or columns as possible so that the resulting matrix has the pattern $\mathcal{P}$. The measure $a + b$ used for size is not perfect: a submatrix is not necessarily interesting if it contains only a few rows, no matter how many columns it has.

The following result of Yannakakis addresses the computational complexity of this class of submatrix problems, and we will use it in later sections to establish NP-hardness of submatrix problems on C1P, nestedness, and bandedness. The result is based on the concept of hereditary graphs, which was used to prove NP-hardness for a large family of node-deletion

problems on graphs [LY80]. We say that a collection $\mathcal{M}$ of binary matrices is *nontrivial* if $\mathcal{M}$ has infinite size and there are infinitely many binary matrices not in $\mathcal{M}$.

Theorem 2.8 (Submatrix problem complexity [Yan81b]) *Let $\mathcal{M}$ be a nontrivial collection of binary matrices closed under permutation and deletion of rows and columns. Denote by S an $a \times b$ submatrix of a given matrix A. Then finding a submatrix $S \in \mathcal{M}$ that maximizes $a + b$ is solvable in polynomial time if the matrices of $\mathcal{M}$ have bounded rank, and NP-hard otherwise.*

We can characterize many local patterns as submatrices. For example, frequent itemsets are submatrices that form rectangles full of 1s. On the other hand, finding maximal tiles [Mie05, Ch. 4.4] is close to Problem 2.7. When we apply Theorem 2.8 on a reorderable pattern $\mathcal{P}$, the collection $\mathcal{M}$ consists of all the matrices in $\mathcal{P}$ and their submatrices.

If we search submatrices in real-world matrices and require the submatrix to have a perfect pattern $\mathcal{P}$, we usually find only small matrices. It may be more useful to find larger matrices that almost have $\mathcal{P}$. For example, given a fixed distance d, we accept submatrices that have distance at most d to a matrix that has $\mathcal{P}$. Assuming we have a *utility function* f that measures the utility of submatrices with respect to $\mathcal{P}$, finding a submatrix can be formulated as follows.

Problem 2.9 (Submatrix With Pattern $\mathcal{P}$) *Given a binary matrix A, a nonnegative weight matrix W, and a fixed distance d, find a submatrix S of A that has the distance to $\mathcal{P}$ at most d and maximizes the utility $f(S)$.*

What constitutes a *good* submatrix depends on the application. For example, a submatrix that has many rows and columns but is very sparse is not necessarily as interesting as a small matrix with half of its values 1s. We included a utility function f in Problem 2.9 to take into account various qualities the rows and columns in submatrices may have. We obtain a simple heuristic method (Algorithm 1) that removes rows and/or columns one by one until the distance to the pattern $\mathcal{P}$ is within the threshold d.

The exact definition of the utility function f depends, of course, on the pattern $\mathcal{P}$. Sometimes the most interesting matrices are those that have the same number of 1s and 0s, and taking into account the information-theoretical entropy helps to produce submatrices with this property. In general, different utility functions produce different-looking matrices, and it depends on the application which of them is the most useful. We return to the generic algorithm and utility measures in later sections.

Algorithm 1 `FindSubmatrix`

Input: binary matrix A, nonnegative weights W, maximum distance d, pattern $\mathcal{P}$, and utility measure f for submatrices
Output: submatrix S of A such that the d_W-distance from S to $\mathcal{P}$ is at most d and the utility $f(S)$ is close to maximum.

$S \leftarrow A$
$B \leftarrow \operatorname{argmin}_{X \in \mathcal{P}} d_W(S, X)$ // closest matrix with $\mathcal{P}$ (size as S)
while $d_W(S, B) > d$ **do**
  for all rows/columns i in S **do**
    remove row/column i from matrix S
    $f_i \leftarrow f(S)$ // compute utility value
    insert row/column i back to S // cancel the latest removal
  end for
  $r \leftarrow \operatorname{argmax}_i f_i$
  remove row/column r from S
  recompute closest matrix B
end while
return S

One general-purpose utility function is as follows. Given a submatrix S of the data, let B be a closest matrix to S that has a pattern $\mathcal{P}$. We consider B to be a ground-truth matrix and count the type of differences between B and S: the number of *false positives* fp is the number of 0s in B that are 1s in S (0-to-1 flips). Likewise, *false negatives* fn is the number of 1-to-0 flips. For completeness, *true positives* tp counts the number of shared 1s (that is, 1-to-1 entries) and *true negatives* tn the number of shared 0s (0-to-0). The following metrics are frequently used in information retrieval to measure correctness of pattern recognition [BYRN99].

$$\begin{aligned} \mathsf{recall} &= \mathsf{tp}/(\mathsf{tp} + \mathsf{fn}) \\ \mathsf{precision} &= \mathsf{tp}/(\mathsf{tp} + \mathsf{fp}) \\ \mathsf{accuracy} &= (\mathsf{tp} + \mathsf{tn})/(\mathsf{tp} + \mathsf{tn} + \mathsf{fp} + \mathsf{fn}) \end{aligned} \tag{2.2}$$

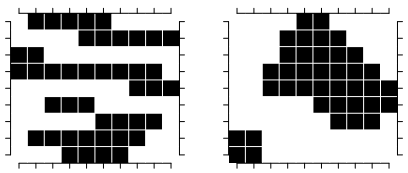
In terms of distance to a pattern $\mathcal{P}$, recall prefers a submatrix that needs few 0-to-1 flips, $\mathsf{precision}$ avoids 1-to-0 flips, and $\mathsf{accuracy}$ seeks small Hamming distance. In later sections, we use Algorithm 1 with one of the three metrics to assess the quality of the submatrices. In more precise terms, the utility $f(S)$ of a matrix S is the minimum recall (or $\mathsf{precision}$ or $\mathsf{accuracy}$) of its rows and columns, each of which is computed on that row or column separately. As a result, the worst row/column is removed in each step.

Chapter 3

Consecutive ones

The concept of consecutivity occurs naturally in some applications. For example, consider a paleontological dataset that has binary records on species (rows) and points in time (columns). Value 1 indicates that a species is alive at a point in time. Now a lifespan of a species can be described as a pattern of consecutive 1s on the corresponding row, such as in the matrices in Figure 3.1.

In general, consecutive 1s reflect certain dependencies in data, such as order and continuation of time. We will next provide the basic concepts related to consecutive 1s in binary matrices, and review the complexity results for the most common consecutive patterns. We concentrate on two reorderable patterns, consecutive-ones and simultaneous consecutive-ones. For example, the matrix in Figure 3.1(a) has consecutive 1s on rows, while the matrix in (b) has consecutive 1s on both the rows and columns.



(a) (b)

Figure 3.1: An example of a consecutive-ones matrix (a) and a simultaneous consecutive-ones matrix (b).

3.1 Consecutivity

We start with the basic definitions and results on a consecutive pattern in vectors. Given a binary vector, we show how to find a closest consecutive vector for any given vector efficiently; this will be useful in later sections that deal with consecutive patterns in matrices.

Definition 3.1 (Consecutive vector) *A binary vector* $\mathbf{a}$ *is* consecutive *if all 1s in* $\mathbf{a}$ *occur at consecutive indices.*

Notation $\langle s, e\rangle$ represents a consecutive vector that has 1s on indices $s, s+1, \ldots, e-1$ and 0s elsewhere. For example, we denote $(0, 1, 1, 1, 0, 0)$ by $\langle 2, 5\rangle$. We say that the index i is *included* in the vector $\langle s, e\rangle$, if $s \leqslant i \leqslant e-1$. In particular, the zero vector $\langle s, s\rangle$ has no 1s, and therefore includes no indices. We write down the exact length of the vector $\langle s, e\rangle$ only when the length is not clear from the context.

Problem 3.2 (Closest Directly Consecutive Vector**)** *Given a binary vector* $\mathbf{a}$ *and a nonnegative weight vector* $\mathbf{w}$*, find a binary vector* $\mathbf{b}$ *that is consecutive and minimizes the distance* $d_{\mathbf{w}}(\mathbf{a}, \mathbf{b})$.

Next, we consider another problem and give an algorithm that will be used as a subroutine in solving Problem 3.2.

Problem 3.3 (Maximum Subvector**)** *Given a vector* $\mathbf{a}$*, find a subvector* $(a_s, a_{s+1}, \ldots, a_{e-1})$ *that maximizes the sum of the values in the subvector.*

Problem 3.3 can be solved by applying Kadane's algorithm [Ben84], which uses dynamic programming to achieve time complexity $\mathcal{O}(n)$. Algorithm 2 gives the details of this `MaximumSubvector` method.

We can now solve Problem 3.2 in linear time in the length of the vector. As described in Algorithm 3, the idea is to reduce Problem 3.2 to the Maximum Subvector problem, and use the algorithm `MaximumSubvector` as a subroutine.

In `FindConsecutive` (Algorithm 3) we seek a consecutive binary vector to which the binary vector $\mathbf{a}$ has the smallest distance. Given a consecutive vector $\mathbf{b} = \langle s, e\rangle$, two types of differences contribute to the distance: 0-entries in $\mathbf{a}$ that are included (1s) in $\mathbf{b}$, and 1-entries in $\mathbf{a}$ that are not included (0s) in $\mathbf{b}$. Specifically, let Z be the distance from $\mathbf{a}$ to the zero vector, that is, the sum of weights on the entries where $\mathbf{a}$ has 1s. Taking the zero vector as a starting point, each included 1-entry in $\mathbf{b}$ decreases the distance, whereas each included 0-entry increases it.

Algorithm 2 `MaximumSubvector`

Input: vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$
Output: (max, s, e), such that subvector $(a_s, a_{s+1}, \ldots, a_{e-1})$ has maximum sum, max

```
1: sum ← 0; s ← 1; e ← 1
2: start ← 1; curr ← 0
3: for all end = 1, 2, ..., n do
4:    curr ← curr + a_end      // include position end to current subvector
5:    if curr < 0 then
6:       curr ← 0; start ← end + 1
7:    else if curr > sum then
8:       sum ← curr; s ← start; e ← end + 1      // update best subvector
9:    end if
10: end for
11: return (sum, s, e)
```

Algorithm 3 `FindConsecutive`

Input: binary vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and nonnegative weight vector $\mathbf{w}$
Output: $(d,\ \mathbf{b})$, where $\mathbf{b}$ is a consecutive binary vector that minimizes $d_{\mathbf{w}}(\mathbf{a}, \mathbf{b}) = d$

1: $N \leftarrow \{i \mid a_i = 1 \text{ and } 1 \leqslant i \leqslant n\}$ // positions of 1s
2: $Z \leftarrow \sum_{i \in N} w_i$ // distance from $\mathbf{a}$ to the zero vector
3: Construct n-dimensional vector $\mathbf{c}$ as $c_i = \begin{cases} +w_i, & \text{if } a_i = 1, \\ -w_i, & \text{if } a_i = 0. \end{cases}$
4: $(max, s, e) \leftarrow$ `MaximumSubvector`$(\mathbf{c})$
5: **return** $(Z - max, \langle s, e \rangle)$

We construct a *distance vector* $\mathbf{c}$ from the binary values $\mathbf{a}$ and weights $\mathbf{w}$ as on Line 3. Given a consecutive vector $\mathbf{b} = \langle s, e \rangle$, each value c_i refers to the decrease in the distance should the index i be included. The distance from $\mathbf{a}$ to $\mathbf{b}$ is therefore $Z - (c_s + c_{s+1} + \cdots + c_{e-1})$. Minimizing the distance is now the same as maximizing $\sum_{i=s}^{e-1} c_i$, which is an instance of Maximum Subvector. Running `MaximumSubvector` on the distance vector $\mathbf{c}$ solves the problem. The returned maximum subvector, represented by the indices from s to $e-1$, is then interpreted as the consecutive binary vector $\langle s, e \rangle$. The time complexity for `FindConsecutive` is $\mathcal{O}(n)$.

3.2 Consecutive matrices

We generalize the concept of consecutiveness on vectors to binary matrices, namely consecutive-ones matrices and simultaneous consecutive-ones matrices. Next, we define the consecutive-ones property (C1P) exactly.

Definition 3.4 (Direct C1P) *A binary matrix A has* direct C1P *if each row vector is consecutive.*

Definition 3.5 (Consecutive-ones property, C1P) *A binary matrix A has the* consecutive-ones property on rows*,* C1P*, if there exists a permutation of the columns such that the permuted matrix has direct C1P.*

The difference between Definitions 3.4 and 3.5 is that in a direct C1P matrix the rows have consecutive 1s, while in a C1P matrix we only require the existence of such a column permutation. There may exist several such permutations, some of which may lead to identical matrices. We consider C1P to be more important a property than direct C1P: the underlying structure of the data matters more than any initial permutation of the data. For example on page 21, both matrices in Figure 3.1 have direct C1P. Also, consider the three matrices below in (3.1). Starting from the matrix on the left, we can permute its columns to establish consecutive 1s on each of the rows. Furthermore, by also permuting the rows we obtain the matrix on the right that has consecutive 1s on both rows and columns.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{array}\right] \end{array} \rightsquigarrow \begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} d & e & a & c & b \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \rightsquigarrow \begin{array}{c} \\ C \\ A \\ D \\ B \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} d & e & a & c & b \end{array} \\ \left[\begin{array}{ccccc} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \tag{3.1}$$

An alternative characterization of C1P can be given by means of forbidden submatrices: a binary matrix has C1P if and only if none of its submatrices belong to the collection of forbidden submatrices, as described by Tucker [Tuc72]. This collection contains an infinite number of matrices, and is not described here. In the literature the C1P pattern is sometimes defined on the columns rather than on the rows.

All C1P matrices are also zero-partitionable (Definition 2.6). To see this, consider a directly C1P matrix obtained from a C1P matrix by permuting the columns. Order the consecutive row vectors $\mathbf{r} = \langle s, e \rangle$ in ascending order of s. Now each 0 that occurs before the 1s on a row can be labeled with Cs and other 0s with Rs.

Since the introduction of the concept of consecutiveness, many types of matrix data have been studied for that property: archeological, DNA, and interaction data, for example. Problems on consecutiveness belong to a family of problems commonly known as the seriation problems [Lii10].

Detecting C1P in binary matrices is particularly important in presence/absence data of paleontological applications [PFM06], where the data contains binary indication of fossil occurrence in excavation sites. It is expected that reordering the data by the site age produces consecutive patterns that show the lifespans of species. The question of determining the distance from a given matrix to a C1P matrix corresponds to counting Lazarus events, that is, occurrences of a species after its assumed extinction and before its resurrection. Also file organization benefits from consecutive patterns [DG99]: given a query, it is faster to read all relevant records if they occur consecutively on a hard disk. In the field of bioinformatics, C1P patterns are encountered in the physical mapping problem [GGKS95], in which DNA fragments need to be reordered to obtain their relative positions in the original DNA sequence.

Next we consider a pattern where consecutiveness holds for both row vectors and column vectors simultaneously.

Definition 3.6 (Direct SC1P) *A binary matrix A has* direct SC1P *if both A and A^T have direct C1P.*

Definition 3.7 (Simultaneous consecutive-ones property, SC1P) *A binary matrix A has* simultaneous consecutive-ones property, SC1P, *if there exist permutations of the rows and columns such that the permuted matrix has direct SC1P.*

A directly SC1P matrix may consist of several blocks of 1s, which can be exploited to identify block matrices, to find overlapping clusters in data, as well as to solve elementary scheduling tasks [Osw03]. For example, the matrix in Figure 3.1(a) does not have SC1P but 3.1(b) has direct SC1P; in (3.1) all three matrices have SC1P but only the last one has direct SC1P. The SC1P pattern and its theoretical results also help to develop theory and methods for other reorderable patterns, such as bandedness.

Another characterization of SC1P comes again from forbidden submatrices [Tuc72], but the collection of such matrices again has infinite size.

Also other concepts that involve consecutiveness exist. In *circular-ones* [BL76] the consecutive 1s are allowed to wrap: the first and the last column of a matrix are considered adjacent. In *k-consecutive* matrices [GGKS95] each row has at most k consecutive streaks of 1s.

3.3 Recognizing consecutive matrices

This section studies how the C1P and SC1P patterns can be recognized in a matrix regardless of its permutations. We consider the patterns that occur without errors; the corresponding distance problems that involve errors are studied in later sections.

Problem 3.8 (C1P Recognition) *Given a binary matrix A, determine whether it has C1P.*

Several algorithms [BL76, Hsu02] exist for solving the problem in linear time $\mathcal{O}(mn)$, and we will refer to these methods later as `TestC1P`. It would be cumbersome to use the Tucker's forbidden submatrices in developing efficient algorithms for C1P, because the number of forbidden submatrices increases with the size of the matrix. Matrices that do not have C1P give rise to another problem: identifying the forbidden submatrices of minimum size [BRV10]. Finding these submatrices has applications in, for example, ancestral genome reconstruction [CHSY10].

Problem 3.9 (Recognizing SC1P) *Given a binary matrix A, determine whether it has SC1P.*

It is easy to solve Problem 3.9 in linear time: it is sufficient to test with `TestC1P` that both A and A^T have C1P, which can be done in linear time. As was the case with C1P, the number of forbidden submatrices for SC1P is not bounded and the characterization does not lead to efficient algorithms.

As stated in Section 2.4, the recognition algorithms for C1P can be used to check whether a graph is an *interval graph* [FG65]. A C1P recognition algorithm can also be used to determine in polynomial time [BL76] whether a matrix has circular-ones. It is, however, an NP-complete problem to determine whether the columns of a matrix can be permuted so that the matrix is k-consecutive, for all fixed $k \geqslant 2$ [GGKS95].

3.4 Distance to directly consecutive patterns

It may be the case that a data matrix contains errors which prevent a perfect pattern to occur. We next study the problems of finding a closest directly C1P or directly SC1P matrix for a given matrix A. It turns out that for C1P the problem is computationally easy, whereas for SC1P it is an NP-hard task.

Problem 3.10 (Closest Direct C1P) *Given a binary matrix A and a nonnegative weight matrix W, find a binary matrix B that has direct C1P and minimizes the distance $d_W(A, B)$.*

We give a linear-time method for Problem 3.10 in Algorithm 4. Since the distance associated with each row in the matrix is independent from the other rows, we can compute the distances row by row and add the distances together. As a subroutine, we use Algorithm 3 for each row in A. The overall time complexity is then linear $\mathcal{O}(mn)$.

Algorithm 4 `FindDirectC1P`

Input: $m \times n$ binary matrix A, nonnegative weights W
Output: $(dist, B)$, where B is a directly C1P matrix that minimizes distance $d_W(A, B) = dist$

1: $dist \leftarrow 0$
2: $B \leftarrow m \times n$ zero matrix
3: **for all** $i = 1, 2, \ldots, m$ **do**
4: $\quad \mathbf{r}, \mathbf{w} \leftarrow i$th row vectors from A and W
5: $\quad (d, \mathbf{b}) \leftarrow$ `FindConsecutive`$(\mathbf{r}, \mathbf{w})$
6: $\quad dist \leftarrow dist + d$
7: $\quad$ set $\mathbf{b}$ as the ith row in B
8: **end for**
9: **return** $(dist, B)$

Problem 3.11 (Closest Direct SC1P) *Given a binary matrix A and a nonnegative weight matrix W, find a matrix B that has direct SC1P and minimizes the distance $d_W(A, B)$.*

It is usually the permutation part that makes minimum distance problems hard on reorderable patterns, but for the SC1P pattern difficulty arises even when no permutations are allowed. Indeed, the d_{H}-Closest Direct SC1P problem is NP-hard [OR09], and consequently also Closest Direct SC1P is NP-hard, even though no reordering of rows or columns is done. For other patterns in this thesis (C1P, nested, banded) the corresponding problem can be solved in polynomial time.

3.5 Distance to consecutive patterns

It is common that the underlying C1P or SC1P pattern in data cannot be seen because of errors and unsuitable permutations. We next define the

problems that ask for closest C1P and SC1P matrices, regardless of initial permutations.

Problem 3.12 (Closest C1P) *Given a binary matrix A and a nonnegative weight matrix W, find a matrix B that has C1P and minimizes the distance $d_W(A, B)$.*

It is an NP-hard task to solve d_A-Closest C1P [Boo75, Pap76, Osw03], which means that it is unlikely that a polynomial-time algorithm exists. Moreover, approximation of d_A-Closest C1P is hard [Vel85]. We will give two exponential-time methods in Section 3.7. A heuristic approach is thus needed to solve the problem in practice; we give some well-known methods for Closest C1P in the next section. In the special case where the number of the rows or columns in a matrix is a constant, however, Closest C1P requires only linear time [OR03].

Problem 3.13 (Closest SC1P) *Given a binary matrix A and a nonnegative weight matrix W, find a matrix B that has SC1P and minimizes the distance $d_W(A, B)$.*

The d_D-Closest SC1P problem is NP-hard [OR09] and therefore Closest SC1P is as well. Both exact (branch and bound) and heuristic algorithms are known for the problem, and if the number of rows or columns is bounded, then solving Closest SC1P requires only polynomial time [Osw03].

As a related computational complexity result, finding a closest circular-ones matrix for a given matrix is NP-complete, even when restricting to the distance d_A [GJ79, Prob. SR16].

3.6 Heuristic algorithms for closest C1P

In the following we describe two polynomial-time heuristic algorithms for Closest C1P (Problem 3.12). Given a binary matrix A, both algorithms produce a column permutation under which A is close to a directly C1P matrix. We can then invoke `FindDirectC1P` (Algorithm 4) on the permuted A, which finds a directly C1P matrix. This C1P matrix is close to the original A, apart from different column permutations.

Intuition says that a good permutation tends to put similar columns close to each other in the order. To find a good permutation for the columns, we consider a symmetric *similarity/distance graph*, which is a complete undirected graph whose vertices are the columns of the input matrix A.

The weight of an edge $\{c, d\}$ is the similarity/distance value between the columns c and d, which is defined later.

The first algorithm is based on a similarity graph on columns, spectral graph theory, and eigenvectors, whereas the second algorithm finds a good permutation for columns by approximating a minimum weight Hamiltonian path in the distance graph. The two algorithms do not solve the problem exactly, but the results in Sections 7.10 and 7.11 suggest they give fair results nonetheless.

There also exists a heuristic [OR00] that is based on linear programming. The idea is to formulate Closest C1P (Problem 3.12) as an integer program (IP) by using the forbidden submatrix characterization and then use a non-integral linear program (LP) relaxation as a heuristic. We do not describe the algorithm in detail here, though.

3.6.1 Spectral ordering

Spectral ordering is based on spectral analysis [vL07] on a similarity graph over the columns of a matrix, and it orders columns in such a way that similar columns are close to each other in the order. Given a symmetric similarity matrix, the key is to construct an unnormalized Laplacian matrix L and find its Fiedler vector, that is, an eigenvector associated with the second-smallest eigenvalue of L. Because a symmetric real matrix has eigenvectors with real values, we can sort the values in the Fiedler vector, which produces a permutation on the columns. This well-known `SpectralOrdering` method is given as Algorithm 5.

`SpectralOrdering` has been popular in finding a permutation that makes a binary matrix almost directly C1P. Furthermore, the method produces

Algorithm 5 `SpectralOrdering`

Input: $m \times n$ binary matrix A, symmetric similarity measure f on columns
Output: column permutation τ that establishes almost direct C1P

1: Construct $n \times n$ similarity matrix S:
   $s_{c,d} \leftarrow f(\mathbf{c}, \mathbf{d})$, where $\mathbf{c}$ and $\mathbf{d}$ are the cth and dth column vectors of A
2: Construct a diagonal matrix D with $d_{i,i} = \sum_{j=1}^{n} s_{i,j}$
3: Let $L \leftarrow D - S$ // an $n \times n$ Laplacian matrix
4: Compute the eigenvectors and eigenvalues of L, and let $\mathbf{e}_2$ be an eigenvector associated with the second-smallest eigenvalue (Fiedler vector).
5: Sort the values in $\mathbf{e}_2$ in nondecreasing order (break ties arbitrarily) and let τ be the corresponding permutation of columns.
6: **return** τ

good approximations to seriation problems [ABH98]. A recent study [Vuo10] shows that `SpectralOrdering` minimizes a certain function that is related to direct C1P, which helps to understand why the method produces results that are optimal for some instances [ABH98].

Note that if there is a ground-truth ordering, the column permutation from `SpectralOrdering` may have it in reversed order. Indeed, the C1P distance would remain unaffected if the sorting on Line 5 was nonincreasing, but the permutation would be different. Therefore, if it makes any difference in the application, both permutations should be considered.

Next, we define three similarity measures for comparing columns. Given two columns c and d and their binary column vectors $\mathbf{c}$ and $\mathbf{d}$, their *dot product* is the number of positions where both vectors have 1s. The *correlation similarity* on the other hand is $(1 + \rho(\mathbf{c}, \mathbf{d}))/2$, where $\rho(\mathbf{c}, \mathbf{d})$ is the well-known Pearson correlation (corr) coefficient between the columns. We see that the correlation similarity values range from 1 (identical columns) to 0 (anticorrelated columns). As an alternative we will use the overlapping measure computed by the *Jaccard coefficient* $|C \cap D|/|C \cup D|$, where C and D are the set interpretations of columns c and d; in case C or D is empty, the value is 1 (does not contradict overlapping sets). Again the value is 1 for identical columns, but this time it is 0 only for non-intersecting columns.

3.6.2 Hamiltonian ordering

Given a weighted undirected graph, in the Travelling Salesman problem we are asked to find a cycle that visits all vertices exactly once and the sum of weights on included edges is minimum. This well-known optimization problem is NP-hard, but there exists a 2-approximation algorithm [RSL77] [CLRS01, `Approx-Tsp-Tour`] for graphs that satisfy the triangle inequality. The algorithm is based on finding a minimum spanning tree and using it to obtain a Hamiltonian cycle.

We use the 2-approximation algorithm to find a Hamiltonian path that has low total weight. Consider the distance graph G on the columns of an input matrix A. We first use the algorithm to find a cycle from G, and then we remove the edge that has the largest weight. The rest of the edges form a Hamiltonian path, which represents a permutation for the columns (or the reverse permutation). We call this method `HamiltonianOrdering`.

We use two distance measures that both satisfy the triangle inequality and are thus eligible in the approximation algorithm. The *Hamming distance* between two column vectors is the number of positions where their values differ. The *Jaccard distance* on the other hand is $1 - J$, where J is the Jaccard similarity coefficient presented in Section 3.6.1.

Whenever `HamiltonianOrdering` is used in experiments, as in Section 7.10, we will indicate which of the distance measures is used to construct the distance graph on the columns. In experiments we will always use the 2-approximation algorithm we just described; we could also use the Christofides algorithm that produces a 3/2-approximation [Chr76] but major improvement is unlikely.

3.7 Exact algorithms for closest C1P

Polynomial-time algorithms for solving Closest C1P (Problem 3.12) are unlikely to exist, as that would imply P=NP. We propose two new exponential-time algorithms for solving the problem exactly. The first reduces the problem to Maximum Satisfiability, whereas the second employs a branch and bound method that uses a lower bound for distances. An integer-programming algorithm for the problem has also been developed [OR00], but we skip the details. Because of their exponential time requirements, most exact algorithms have difficulties with matrices, say, larger than 20×20, which makes heuristic methods more appealing for practical use.

3.7.1 MAX-SAT algorithm

Recall that an *assignment* on a propositional logic formula assigns truth values 1 (true) or 0 (false) to each variable. The formula is *satisfiable* if the formula evaluates to 1 under some assignment.

A *complete satisfiability solver* (SAT-solver) [GKSS08] recognizes whether a given propositional logic formula is satisfiable and returns a satisfying assignment if one exists. Since deciding on satisfiability is an NP-complete problem [GJ79, Ch. 2.6], all current solvers use exponential time in the worst case, but recent developments [GKSS08] and regular competitions [Sat11, Max11] suggest that these solvers can be useful in solving various hard problems that can be encoded in propositional logic. Next, we show how SAT-solvers can solve Problem 3.12.

SAT-solvers usually assume that the propositional logic formulae are in *conjunctive normal form* (CNF), which is a conjunction of *clauses*, each of which consists of a disjunction of variables and their negations. In order to assess the size of a formula, we count the number of variables and clauses used. Given an $m \times n$ binary matrix A, we construct a CNF formula that is satisfiable if and only if A has C1P. The formula consists of variables and clauses as follows.

- *Entry variables* $e_{i,j}$ for each matrix entry $a_{i,j}$. Truth values on the entry variables represent matrix values 1 and 0. Contributes mn variables.

- *Order variables* $Z_{a<b}$ for each ordered pair of distinct columns (a, b) in A. An assignment true on variable $Z_{a<b}$ indicates that a preceeds b in order. Contributes $n(n-1)$ variables.

- *Entry clauses* $(e_{i,j})$ for each entry $a_{i,j} = 1$, and $(\neg e_{i,j})$ for each entry $a_{i,j} = 0$. The entry clauses reflect the values in A, and each unsatisfied entry clause corresponds to a flip in A. Contributes mn clauses.

- *Antisymmetry clauses* for each unordered pair of distinct columns $\{a, b\}$ in A.

$$\text{Antisymmetry: } \neg(Z_{a<b} \leftrightarrow Z_{b<a})$$
$$\text{Same clauses in CNF: } (Z_{a<b} \vee Z_{b<a}) \wedge (\neg Z_{a<b} \vee \neg Z_{b<a})$$

  This ensures that either a preceeds b or vice versa. As a result, $\neg Z_{a<b}$ and $Z_{b<a}$ are equivalent. Contributes $n(n-1)$ clauses.

- *Transitivity clauses* for each ordered triplet of distinct columns (a, b, c).

$$\text{Transitivity: } (Z_{a<b} \wedge Z_{b<c} \rightarrow Z_{a<c})$$
$$\text{Same clause in CNF: } (\neg Z_{a<b} \vee \neg Z_{b<c} \vee Z_{a<c})$$

  Given an assignment, the order variables form a total order if and only if all antisymmetry and transitivity clauses are satisfied. Contributes $n(n-1)(n-2)$ clauses.

- *Consecutivity clauses* for each row i and ordered triplet of distinct columns (a, b, c).

$$\text{Between two 1s is a 1: } (Z_{a<b} \wedge Z_{b<c} \wedge e_{i,a} \wedge e_{i,c} \rightarrow e_{i,b})$$
$$\text{Same clause in CNF: } (\neg Z_{a<b} \vee \neg Z_{b<c} \vee \neg e_{i,a} \vee \neg e_{i,c} \vee e_{i,b})$$

  The consecutivity clauses ensure that on each row no 0s occur between two 1s. Given an assignment, a total order is established and all consecutivity clauses are satisfied if and only if the corresponding matrix has direct C1P. Contributes $mn(n-1)(n-2)$ clauses.

In cases where A contains errors and does not have C1P, we can still find the Hamming distance to a closest such matrix. Indeed, based on the

construction above, we can produce a C1P matrix B by using an assignment that satisfies all the antisymmetry, transitivity, and consecutivity clauses, but leaves some entry clauses unsatisfied. The unsatisfied entry clauses then correspond to the entries where A and B differ.

Finding the Hamming distance is an instance of the Partial Max-Sat problem [HLO08]. Given a propositional logic formula and a subset P of its clauses, the goal is to find a truth assignment on the variables that satisfies all clauses in P, and the number of other satisfied clauses is maximum. We include to P all antisymmetry, transitivity, and consecutivity clauses, and only entry clauses are allowed to be unsatisfied. Solving this instance with a partial MAX-SAT solver produces an assignment that corresponds to a C1P matrix that is closest to A, and the Hamming distance is then the number of unsatisfied entry clauses for this assignment.

The general d_W-weighted version of finding a closest C1P matrix can be solved similarly, when a weighted MAX-SAT solver [HLO08] is available. We assign to each entry clause a weight that is the same weight as the corresponding matrix entry $a_{i,j}$ has. All antisymmetry, transitivity, and consecutivity clauses have infinite weights (any weight larger than the sum of all weights in W), since they must always be satisfied. A weighted solver maximizes the sum of weights on satisfied clauses; therefore the sum of weights on the unsatisfied entry clauses gives the weighted distance from A to a closest C1P matrix.

A more compact way to construct the CNF-formula above is to use only $n(n-1)/2$ order variables and treat $\neg Z_{a<b}$ as $Z_{b<a}$. As a result, antisymmetry clauses can be removed, which reduces the size of the formula.

3.7.2 Branch and bound algorithm

To solve Closest C1P (Problem 3.12) exactly, we take a simple brute-force algorithm `BF` as a starting point and convert it into a branch and bound method that computes lower bounds (distances) in certain matrices as the algorithm runs, which reduces its running-time.

Every C1P matrix has direct C1P under some column permutation. Actually, the solutions for Closest Direct C1P and Closest C1P (Problems 3.10 and 3.12) are identical, if the columns of the input matrix are permuted appropriately. Therefore we can solve Closest C1P by trying all $n!$ permutations on columns and solving Closest Direct C1P for all permuted matrices A'. The solution is then the minimum distance to direct C1P among the permuted matrices.

The brute-force algorithm `BF` creates column-orders by adding columns one by one into a chain $\mathcal{C}$ (a total order on a subset of columns), backtrack-

ing, and continuing until all total orders have been formed. Every time `BF` forms a total order, we obtain a permuted matrix A', and `BF` invokes `FindDirectC1P` (Algorithm 4) to compute the distance from A' to a closest directly C1P matrix. Finally, by keeping track of the distances encountered, `BF` returns the minimum distance and the associated permutation.

We modify `BF` so that it can avoid iterating through all permutations. Assume that during its execution, `BF` has already established a chain $\mathcal{C}$ that starts with columns $c_1 < c_2 < \cdots < c_i$, but $c_{i+1}, c_{i+2}, \ldots, c_n$ have not been included in $\mathcal{C}$ yet. Consider all total orders whose first i columns are as in $\mathcal{C}$: each of the total orders induces a permuted matrix A'. We give a lower bound for the distance from any A' to a directly C1P matrix.

We obtain a lower bound for the matrix as a sum of minimum distances computed separately for each row. We next show how to compute the distance for a single row vector $\mathbf{r}$, given a chain $\mathcal{C}$ of i columns from A. Without loss of generality, we assume that when $\mathbf{r}$ is permuted as in $\mathcal{C}$, its first i entries are $(r_1, r_2, \ldots, r_i)$, and the ordering of the remaining entries $r_{i+1}, r_{i+2}, \ldots, r_n$ has not been fixed yet. We need to know the minimum distance from the permuted $\mathbf{r}$ to a consecutive vector, no matter what the permutation of the remaining $n - i$ entries is. We observe that placing all remaining 1-entries before the remaining 0s minimizes the distance. Assuming that the number of 1-entries among remaining entries is k, we then arrive at the following permuted row vector $\mathbf{s}$.

$$\mathbf{s} = \begin{cases} s_p = r_p & \text{for } p = 1, 2, \ldots, i \\ s_p = 1 & \text{for } p = i+1, i+2, \ldots, i+k \\ s_p = 0 & \text{for } p = i+k+1, i+k+2, \ldots, n \end{cases}$$

Then solving Maximum Subvector (Problem 3.3) on $\mathbf{s}$ and $\mathbf{w}$ gives the minimum distance for this row, where $\mathbf{w}$ is the associated weight vector in W and is permuted as $\mathbf{s}$. As the total order used in $\mathbf{s}$ is found for each row separately, this cannot always be applied to all rows simultaneously. Therefore the sum of the distances from the row vectors gives a lower bound for the matrix, given a chain $\mathcal{C}$.

For example, consider a row vector $\mathbf{r} = (1, 1, 1, 0, 0, 1, 1, 0, 1)$ and a chain $\mathcal{C}$ of columns that has $c_2 < c_5 < c_6 < c_1$. Therefore the first four entries of the permuted $\mathbf{r}$ are (1,0,1,1). We establish a total order that starts with $\mathcal{C}$ and produces a permuted vector $\mathbf{s}$ = (1,0,1,1,1,1,1,0,0). Given the chain $\mathcal{C}$, the produced total order minimizes the distance from $\mathbf{s}$ to a consecutive vector, regardless of weights.

To conclude, we add the following check to the brute-force algorithm `BF`. Every time the algorithm produces a new chain $\mathcal{C}$, it computes a lower

bound for the distance from the matrix, assuming chain $\mathcal{C}$. Should the lower bound be equal to or larger than a distance produced by a total order this far, the algorithm backtracks. In this case, it never considers another total order that starts with $\mathcal{C}$.

3.8 C1P submatrices

Even when a matrix does not have C1P, it is still interesting to know which of its submatrices have such a property. After all, the forbidden submatrix characterization ensures that some submatrices indeed have C1P (small submatrices cannot contain a forbidden submatrix). In the physical mapping problem, for example, the C1P pattern does not occur perfectly if the dataset on DNA fragments contains errors. C1P submatrices can then be used to obtain the relative positions of the fragments [TZ07].

Problem 3.14 (Maximum-Size C1P Submatrix) *Given a binary matrix A, find in A a submatrix that has C1P and maximizes $a + b$, where a is the number of rows in the submatrix and b that of columns.*

Problem 3.14 is a special case of Problem 2.7, but restricted on C1P. We show that this problem is hard by using Theorem 2.8 (Yannakakis) and two types of binary matrices. An *upper triangular* matrix U has entry $u_{i,j}$ equal to 1 if and only if $i \leqslant j$. A *zero diagonal* matrix D has entry $d_{i,j}$ equal to 1 if and only if $i \neq j$. Both matrix families have unbounded size and rank.

Theorem 3.15 Maximum-Size C1P Submatrix *is NP-hard.*

Proof. Let $\mathcal{M}$ be the collection of all C1P matrices. We observe that $\mathcal{M}$ is nontrivial: upper triangular matrices have C1P, but zero diagonal matrices of size at least 3×3 do not have C1P. We note that $\mathcal{M}$ is closed under permutation of rows and columns by definition. Because C1P has a forbidden submatrix characterization, $\mathcal{M}$ is also closed under deletion of the rows and columns. The result follows from Theorem 2.8, because $\mathcal{M}$ contains upper triangular matrices that have unbounded rank. □

We can also restrict Problem 3.14 to either the rows or columns alone: we remove as few rows (or columns) as possible to reach a submatrix that has C1P. Still, both problems remain NP-hard [HG02].

Many matrices, such as the incidence matrices of graphs, contain only a few 1s on each column. Focusing on sparse matrices where the number of 1s is bounded on each row and column may produce more efficient algorithms. Indeed, if each row and column contains at most two 1s, then

C1P submatrices can be found in polynomial time. Unfortunately, if either the rows or columns contain three 1s, the problems already become NP-complete [TZ07]. The literature knows several negative approximability results for the submatrix problems, but also polynomial-time approximation algorithms that assume bounded row and column sums [DGN10].

Chapter 4

Nestedness

The concept of nestedness describes hierarchies and subset relations in data. This type of pattern appears in contexts where one is expected to witness a progressive development of attributes, such as students advancing through a curriculum from introductory to specialized courses. Another example involves species distributions: it is common that species found in barren areas also occur in abundant areas, which demonstrates a hierarchy of the species. Nested patterns occur frequently in ecological datasets, as well as in interaction networks; these aspects will be discussed later in Chapter 5. The figure below is an example of a nested matrix.

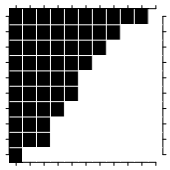


Figure 4.1: An example of a directly nested matrix.

4.1 Theoretical results

The theoretical background of the ecological concept of nestedness was not fully developed until Mannila and Terzi [MT07] introduced the concept to the data mining community. In this section we give the exact definition of nestedness and several alternative characterizations. We also review theoretical results on nestedness that are needed in algorithms that are introduced in the following sections.

Definition 4.1 (Directly nested) *A binary matrix A is* directly nested *if for each 1-entry $a_{i,j} = 1$ we have that $a_{r,c} = 1$ for all $r \in [i]$ and $c \in [j]$.*

Definition 4.2 (Nested) *A binary matrix A is* nested *if there exist permutations of the rows and columns such that the permuted matrix is directly nested.*

For example the matrices in (4.1) are identical, apart from different permutations. Both are nested, but only the latter is directly nested.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \quad \rightsquigarrow \quad \begin{array}{c} \\ D \\ E \\ A \\ B \\ C \end{array}\!\!\begin{array}{c} \begin{array}{cccccc} e & a & d & b & c & f \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \tag{4.1}$$

We observe that nestedness is invariant under transposition: A is nested if and only if A^T is nested. This also holds for directly nested matrices.

We describe three characterizations for direct nestedness. First, an $m \times n$ binary matrix A is directly nested if and only if $R_1 \supseteq R_2 \supseteq \cdots \supseteq R_m$ and $C_1 \supseteq C_2 \supseteq \cdots \supseteq C_n$, where R_i and C_j are the set interpretations of the row i and column j. In other words, both the rows and columns form a *chain* under set inclusion.

Second, consider a visual representation of a matrix A (as in Figure 4.1) and a grid between the entries of the matrix. We can imagine a path along the grid that starts from the bottom-left corner, moves only up or right, and ends at the top-right corner. Such a *staircase path* separates the entries in A into two parts: a matrix is directly nested if and only if there exists a staircase path that separates all 1s from 0s [Wes98]. The number of directly nested matrices is thus equal to that of staircase paths. There are exactly $m + n$ moves until the path reaches the top-right corner, m of which are up moves. The number of staircase paths—and that of directly nested matrices—is therefore $\binom{m+n}{m}$.

The third characterization comes from number theory. An *integer partition* of a positive integer q is a vector $\mathbf{c} = (c_1, c_2, \ldots, c_k)$ of positive integers such that $c_1 \geqslant c_2 \geqslant \cdots \geqslant c_k > 0$ and $\sum_{i=1}^{k} c_i = q$. A graphical representation of an integer partition $\mathbf{c}$, called a *Ferrers diagram*, is a binary matrix F that has c_i topmost entries on the column i as 1s. For example, given an integer partition $(5, 4, 4, 2, 1, 1)$, its Ferrers diagram is the second matrix in (4.1). We observe that Ferrers diagrams and directly nested matrices are

equivalent (apart from zero vectors on the rows and columns). By definition, $\mathbf{c}$ contains the column sums of the Ferrers diagram F. The *conjugate* of $\mathbf{c}$ is the vector $\mathbf{s}$ whose values represent the row sums of F. We observe that $\mathbf{s}$ is unique and also an integer partition of q.

If a matrix has a nested pattern, we need to find suitable permutations of the rows and columns to observe the directly nested pattern. We give two characterizations that describe the structure of nested matrices.

First, a matrix A is nested if and only if for all columns i, j we have $C_i \subseteq C_j$ or $C_j \subseteq C_i$. By the properties of subset inclusion, this forms a chain on the column vectors in A. In fact, the rows form a chain if and only if the columns form a chain.

Second, nestedness can be also characterized by forbidden submatrices. The matrices in (4.2) are called *switch boxes*.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.2}$$

A binary matrix A is nested if and only if none of its submatrices is a switch box. Contrary to the C1P and SC1P patterns, where the collections of forbidden submatrices have unbounded size, the switch boxes have size 2×2 and we can use them in developing efficient algorithms.

Next we show that nested patterns are unique given the row sums and the column sums. A *Ryser class* [Rys60] is a collection of binary matrices that have the same dimensions and identical row sums $\mathbf{r}$ and column sums $\mathbf{c}$. Ryser classes have been studied extensively [Bru80, WZ98]. Suppose a binary matrix A contains a switch box as a submatrix. A *switch* represents flipping the four entries in this submatrix, which produces another type of switch box, but preserves the row sums and column sums.

Theorem 4.3 (Ryser class reachability [Rys57]) *All binary matrices included in a Ryser class are reachable from one another by a series of switches on their switch boxes.*

We can now state the result that establishes the uniqueness of nested matrices. This result will be used to develop an algorithm for nestedness recognition in the next section.

Theorem 4.4 (Nested matrices and Ryser classes) *A binary matrix A is nested if and only if A is the only matrix within its Ryser class.*

Proof. By Theorem 4.3 a binary matrix A is the only matrix within its Ryser class if and only if A does not contain any switch boxes as submatrices. A matrix with no switch boxes is equivalent to being nested. □

4.2 Recognition of nestedness

In this section we give three polynomial-time algorithms that recognize whether a matrix is nested. The challenge is to detect whether there exist suitable permutations of the rows and columns such that a directly nested pattern can be observed. Because of the equivalence of chain graphs and nestedness, as established in Section 2.4, an algorithm that recognizes nestedness also recognizes chain graphs.

Problem 4.5 (NESTEDNESS RECOGNITION**)** *Given a binary matrix A, determine whether it is nested.*

The first algorithm for NESTEDNESS RECOGNITION arises from the forbidden submatrix characterization. An $m \times n$ matrix A has $\binom{m}{2}\binom{n}{2}$ submatrices of size 2×2. Checking that none of them is a switch box takes time $\mathcal{O}((mn)^2)$.

The second algorithm comes from reordering the rows and columns and then checking for direct nestedness. We reorder the rows (and columns) in nonincreasing order by their row sums (column sums). Because of the subset relations of the rows and columns in A, the permuted A is now directly nested if and only if A is nested. Reordering and checking direct nestedness takes time $\mathcal{O}(m \log m + n \log n + mn)$.

The third algorithm solves Problem 4.5 in sublinear time $\mathcal{O}(m+n)$, given the row sums and column sums. For example the degrees of vertices in a bipartite graph are often precomputed, and can be interpreted as row sums and column sums. The algorithm combines the theory of nested matrices, integer partitions, and `CountingSort` method [CLRS01, Ch. 8] that sorts integers in linear time. For technical convenience, we can assume that all row sums and column sums are positive, because zero vectors do not affect nestedness. We first describe the subroutine `Conjugate` and then the recognition algorithm `TestNested` (Algorithms 6 and 7).

The `Conjugate` method (Algorithm 6) takes an integer partition $\mathbf{c}$ as input and returns the unique conjugate partition of $\mathbf{c}$. Let F be a directly nested matrix that represents the Ferrers diagram of $\mathbf{c}$. To find the conjugate partition $\mathbf{s}$ of $\mathbf{c}$, `Conjugate` constructs a staircase path implicitly along the matrix grid of F from the bottom-left corner to the top-right corner, and keeps track of the column and row positions with variables x and y. We obtain the conjugate partition by examining the positions in the path. Looking at the loop conditions reveals that the time complexity for `Conjugate` is $\mathcal{O}(m + n)$.

Next we describe the recognition algorithm `TestNested`. Given an $m \times n$ binary matrix A as input, we use only its row sums $\mathbf{r}$ and column sums $\mathbf{c}$.

Algorithm 6 `Conjugate`

Input: integer partition $\mathbf{c} = (c_1, c_2, \ldots, c_n)$
Output: conjugate integer partition $\mathbf{s} = (s_1, s_2, \ldots, s_m)$

```
1: m ← max(c)                                  // length of conjugate partition s
2: x ← 1                                       // horizontal position: start from the left
3: y ← m                                       // vertical position: start from the bottom
4: while y ⩾ 1 do                              // compute all values in s
5:    if x > n or y > c_x then                 // do we know the value of s_y?
6:       s_y ← x − 1
7:       y ← y − 1                             // start finding the next value in s
8:    else
9:       x ← x + 1      // use the next value in c to deduce the values in s
10:   end if
11: end while
12: return s
```

Algorithm 7 `TestNested`

Input: positive row sums $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ and positive column sums $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ of a binary matrix A
Output: is A nested?

```
1: s ← Conjugate(CountingSort(c) in nonincreasing order)
2: if s is identical to CountingSort(r) in nonincreasing order then
3:    return yes
4: end if
5: return no
```

Since permuting the matrix does not affect nestedness, we can assume that $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ and $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ are in nonincreasing order; if not, we can reorder them in linear time with `CountingSort`. Thus, $\mathbf{r}$ and $\mathbf{c}$ represent a directly nested matrix if and only if A is nested.

Consider a directly nested matrix F that has column sums as $\mathbf{c}$. By using the equivalence of Ferrers diagrams and directly nested matrices, we obtain the conjugate $\mathbf{s}$ of integer partition $\mathbf{c}$ from the `Conjugate` method. Therefore, by using only the column sums $\mathbf{c}$, we can deduce the unique row sums $\mathbf{s}$ in the directly nested F.

We now compare the row sums $\mathbf{s}$ and $\mathbf{r}$ and determine whether A is nested. If $\mathbf{s}$ and $\mathbf{r}$ are different, then A is not nested, because $\mathbf{s}$ contains the unique row sums for a nested matrix that has column sums $\mathbf{c}$. Now suppose the contrary: $\mathbf{s}$ and $\mathbf{r}$ are identical. Now A and F belong to the same Ryser class, since they have the same row sums and column sums. Because F is

nested, Theorem 4.4 implies $F = A$, and A is nested.

The time complexity of Algorithm 7 is sublinear in the size of an input matrix, $\mathcal{O}(m + n)$, which comes from both sorting the integer vectors and computing `Conjugate`. If the row and column sums are not available in constant time, we need $\mathcal{O}(mn)$ time for preprocessing.

4.3 Distance to direct nestedness

If a matrix A is not directly nested, the reason may be that it contains noise. We are interested in finding a closest matrix that is directly nested. This way we can identify the entries in A that break the direct nestedness pattern. We give a polynomial-time algorithm for the problem.

Problem 4.6 (CLOSEST DIRECTLY NESTED) *Given a binary matrix A and a nonnegative weight matrix W, find a binary matrix B that is directly nested and minimizes the distance $d_W(A, B)$.*

Algorithm `FindDirectNested` (Algorithm 8) uses dynamic programming to solve Problem 4.6 in linear time $\mathcal{O}(mn)$. For technical convenience, we decompose the weights in W into two matrices: $W = U + V$, where U contains the weights of 1-entries and V those of 0-entries.

The goal is to construct a directly nested matrix B to which matrix A has minimum distance. We use staircase paths to represent directly nested matrices efficiently. Recall that a staircase path starts from the bottom-left corner of the matrix grid, and moves `right` and `up` until it reaches the top-right corner. On each row, we keep track of the position where the path goes `up`—there are $n + 1$ such positions. The first and last positions are special cases that indicate that the path is on the outer border of the matrix grid.

Denote by A^i the submatrix of A that includes rows $i, i + 1, \ldots, m$. We use dynamic programming to compute distances from the submatrices A^i to direct nestedness for $i = m, m - 1, \ldots, 1$, and use the distances from A^i to deduce the distances for A^{i-1}.

We compute an $m \times (n+1)$ distance matrix C, where entry $c_{i,j}$ represents the path position j on the row i. We have the following invariant: $c_{i,j}$ is the minimum distance from A^i to a directly nested matrix that has $j - 1$ entries as 1s on the (topmost) row i. Once C has been computed, the minimum distance from A to a directly nested matrix is then the minimum on the first row in C.

The algorithm computes the entries in C one by one—assume that $c_{i,j}$ is computed next. We observe that the staircase path may reach (i, j) by

Algorithm 8 `FindDirectNested`

Input: $m \times n$ binary matrix A, nonnegative weights W

Output: minimum distance from A to a directly nested matrix

1: $U \leftarrow \begin{cases} u_{i,j} = w_{i,j} & \text{if } a_{i,j} = 1 \\ u_{i,j} = 0 & \text{if } a_{i,j} = 0 \end{cases}$ $\quad V \leftarrow \begin{cases} v_{i,j} = 0 & \text{if } a_{i,j} = 1 \\ v_{i,j} = w_{i,j} & \text{if } a_{i,j} = 0 \end{cases}$

2: Compute $m \times (n+1)$ matrix C using the recurrence

$$
\begin{aligned}
c_{m,1} &\leftarrow \sum_{k=1}^{n} u_{m,k} \\
c_{m,j} &\leftarrow \texttt{Right}(m,j) \text{ for } j = 2,3,\ldots,n+1 \\
c_{i,1} &\leftarrow \texttt{Up}(i,1) \text{ for } i = m-1, m-2, \ldots, 1 \\
c_{i,j} &\leftarrow \min\{\texttt{Right}(i,j), \texttt{Up}(i,j)\} \\
&\qquad \text{for } i = m-1, m-2, \ldots, 1 \text{ and } j = 2,3,\ldots,n+1 \\
\texttt{Right}(i,j) &= c_{i,j-1} + v_{i,j-1} - u_{i,j-1} \\
\texttt{Up}(i,j) &= c_{i+1,j} + \sum_{k=1}^{j-1} v_{i,k} + \sum_{k=j}^{n} u_{i,k}
\end{aligned}
$$

3: **return** minimum value on the first row vector in C

moving right or up. The operator `Right` assumes that the staircase path reached (i,j) by moving right. In that case the distance $c_{i,j}$ is the same as $c_{i,j-1}$ but with the added/subtracted weight of the 0/1-entry $a_{i,j-1}$. On the other hand, the operator `Up` assumes the last move was up. In that case the distance $c_{i,j}$ is the same as $c_{i+1,j}$ but with the added weights of the entries on the row i that are on the wrong side of the staircase path position j. Because the only way for a path to reach (i,j) is moving either right or up, we choose the distance $c_{i,j}$ to be the minimum of those from operators `Right` and `Up`.

In addition to the minimum distance, the algorithm can also produce a closest directly nested matrix B: we obtain a staircase path by tracing the operators (`Right` and `Up`) used along the path that produced the minimum distance; this path defines a directly nested matrix B.

Algorithm 8 has time complexity $\mathcal{O}(mn)$ if we make use of the following improvement. Instead of using matrices U and V as is, we compute two cumulative weight sum matrices on the rows of U and V, which takes time $\mathcal{O}(mn)$. With these new matrices all the sums in Algorithm 8 can be evaluated in constant time; thus computing a single entry in C takes constant time.

The memory requirement of Algorithm 8 is $\mathcal{O}(mn)$, which comes from the matrix C, the weight matrices U and V, and the cumulative sum matrices. If we only need the minimum distance but not a closest matrix, then $\mathcal{O}(n)$ space suffices. The idea is to implement C, U, V, and the cumulative weight matrices as vectors. In particular, C is an $(n+1)$-dimensional vector and its values are recomputed on each row, which replaces the values computed for the previous row. The weight vectors are generated for each row separately. The minimum distance is obtained as before, but we cannot reconstruct the associated staircase path or a closest matrix.

4.4 Distance to nestedness

In this section we address the problem of finding a closest nested matrix for a given matrix. In contrast to the previous section, this problem involves taking into account different permutations. We review the computational complexity results for the problem restricted on the d_{A}, d_{D}, d_{H}, and d_W distances (augmentation, deletion, Hamming, and weighted). We also give polynomial-time heuristics restricted on these distance measures.

Problem 4.7 (Closest Nested) *Given a binary matrix A and a nonnegative weight matrix W, find a binary matrix B that is nested and minimizes the distance $d_W(A, B)$.*

Problem 4.7 is NP-hard, because the d_{A}-Closest Nested problem (only 0-to-1 flips) has been shown [Yan81a, MT07] to be NP-hard via chain graphs. Also d_{D}-Closest Nested is NP-hard: we can solve d_{A}-Closest Nested by computing d_{D}-Closest Nested on a matrix where 0s and 1s have been interchanged. The computational complexity of d_{H}-Closest Nested, however, is an open question.

We present three heuristic algorithms for Problem 4.7 restricted on certain distance measures. The first algorithm computes a lower bound on d_{A} and d_{H}, the second is an approximation algorithm that gives an upper bound for d_{A}, and the third method gives upper bounds for weighted distances d_W. For more algorithms, a recent study [FMT09] gives approximation algorithms for the d_{A}-Closest Nested problem via minimum cuts and vertex degrees (row sums and column sums).

4.4.1 Lower bound algorithm

Algorithm 9 produces a lower bound for d_{A}-Closest Nested (0-to-1 flips). Denote by p the number of 0-entries in an input matrix A, and by $s(i, j)$ the number of switch boxes in which the entry $a_{i,j}$ is included.

Algorithm 9 `LowerBoundAugNested`

Input: $m \times n$ binary matrix A
Output: lower bound for $d_{\mathrm{A}}(A, B)$, where B is a d_{A}-closest nested matrix

1: $L \leftarrow$ empty list
2: **for all** (i, j) such that $a_{i,j} = 0$ **do**
3: compute $s(i, j)$ and add it to the list L
4: **end for**
5: $(l_1, l_2, \ldots, l_p) \leftarrow \mathtt{sort}(L)$ in nonincreasing order
6: $t \leftarrow 1/2 \cdot \sum_{k=1}^{p} l_k$ // the number of switch boxes in the matrix
7: **return** $\min\{e \in \mathbb{N}_{\geqslant 0} \mid \sum_{k=1}^{e} l_k \geqslant t\}$

Theorem 4.8 (Lower Bound) *Algorithm 9 produces a lower bound for* d_{A}-Closest Nested.

Proof. We count a lower bound for the number of 0-to-1 flips needed to make A nested. To do so, we seek to eliminate all switch boxes, but we decide to ignore new switch boxes that are generated by flipping entries.

For each entry (i, j) such that $a_{i,j} = 0$ denote by $s(i, j)$ the number of switch boxes in which the entry is included. Flipping just one entry, say $a_{i,j}$, eliminates exactly $s(i, j)$ switch boxes. After the first flip, some switch boxes have already been eliminated, and flipping a further entry $a_{u,v}$ eliminates at most $s(u, v)$ switch boxes that were originally in A.

Denote by p the number of 0-entries in A, and let $(l_1, l_2, \ldots, l_p)$ be their switch box counts in nonincreasing order. Then flipping e entries eliminates at most $l_1 + l_2 + \cdots + l_e$ switch boxes in A, which corresponds to flipping the e entries that have the highest participation in switch boxes.

Since each switch box includes two 0s, the number of switch boxes in the matrix is $t = 1/2 \cdot (l_1 + l_2 + \cdots + l_p)$. Let e be the minimum number such that $l_1 + l_2 + \cdots + l_e \geqslant t$. Because the number of eliminated switch boxes is an upper bound, $e - 1$ flips are not enough to eliminate all of them. Because all solutions need to eliminate from A all switch boxes, e is a lower bound for the minimum distance. □

Flipping the entries in the matrix as indicated by Algorithm 9 does not necessarily produce a valid solution, since flipping may generate new switch boxes that are not eliminated. Computing the number of switch boxes for each entry takes time $\mathcal{O}((mn)^2)$, which dominates the time complexity of the algorithm.

Algorithm 9 can be modified to handle the Hamming or deletion distances. For the Hamming distance, Line 2 should include all entries in the matrix, Line 6 should have 1/4 instead of 1/2 (each switch box is counted

four times), and Lines 5 and 6 should have mn instead of p. In case of the deletion distance, Line 2 should include only 1-entries instead of 0-entries, and Lines 5 and 6 should have $mn - p$ instead of p.

Unfortunately, an adaption of the algorithm for the weighted distance is not straightforward. We could order the entries by their eliminating-efficiencies s_i/w_i, where w represents the weights, but this may lead to flipping an entry with high efficiency, when an entry with low cost would suffice. In other words, the result is not necessarily a lower bound.

4.4.2 Approximation algorithm

Another simple algorithm produces an upper bound for d_{A}-Closest Nested and does that with a provable approximation factor. The idea is to identify all 0s that participate in switch boxes and to flip them to 1s; the method is described in Algorithm 10. It is easy to modify the algorithm to solve the deletion distance: flip 1s to 0s. We proceed by giving proofs for the correctness and the approximation factor for the algorithm.

Algorithm 10 `ApproxAugNested`

Input: $m \times n$ binary matrix A
Output: $(d, \widehat{B})$, where d is an upper bound for $d_{\mathrm{A}}(A, B)$; $\widehat{B}$ is close to a d_{A}-closest nested matrix B

1: $\widehat{B} \leftarrow A$
2: $S \leftarrow$ for each entry in A, the number of switch boxes it belongs to
3: **for all** entries $a_{i,j}$ in A **do**
4: **if** $a_{i,j} = 0$ and $s_{i,j} > 0$ **then**
5: $\widehat{b}_{i,j} \leftarrow 1$
6: **end if**
7: **end for**
8: $d \leftarrow \sum_{i,j} |\widehat{b}_{i,j} - a_{i,j}|$
9: **return** $(d, \widehat{B})$

Theorem 4.9 (Upper Bound) *Algorithm 10 produces an upper bound for* d_{A}-Closest Nested.

Proof. Since all 0s in switch boxes are flipped to 1s, it is clear that all switch boxes in the original matrix are eliminated. It remains to show that the flips do not generate any new switch boxes.

If the original matrix is already nested, there are no switch boxes, and flips are not needed. Otherwise the matrix contains at least one switch box.

Without loss of generality, let the rows r_1, r_2 and the columns c_1, c_2 contain one of these switch boxes. In what follows, we assume that we have flipped all 0s that participate in switch boxes; in particular, 0-entries (r_1, c_1) and (r_2, c_2) have been flipped (in bold).

	c_1	c_2
r_1	0	1
r_2	1	0

	c_1	c_2	c_3
r_1	**1**	1	?
r_2	1	**1**	0
r_3	?	0	1

Assume against the claim that the matrix still contains a switch box. Since flipping the 0-entries eliminated all original switch boxes, the remaining switch boxes are generated by these flips. Assume without loss of generality that flipping the 0-entry (r_2, c_2) generated such a switch box on the rows r_2, r_3 and columns c_2, c_3. We next examine the value of the unknown entry (r_1, c_3). If (r_1, c_3) has value 0, the rows r_1, r_3 and columns c_2, c_3 form a switch box that exists before flipping. On the other hand, if (r_1, c_3) has value 1, the rows r_1, r_2 and columns c_1, c_3 formed a switch box before flipping. Thus, either (r_2, c_3) or (r_3, c_2) participated in a switch box and one of them should have been flipped, which is a contradiction with the assumption that there exists a switch box after flipping the 0s. □

Theorem 4.10 (Approximation Factor) *The approximation factor for Algorithm 10 is* 2α*, where* $\alpha = \max\{s(i,j) \mid a_{i,j} = 0\}$.

Proof. Suppose the matrix has k switch boxes. Since each switch box includes two 0s, the algorithm flips at most $2k$ entries, with equality if all 0s in the matrix participate in at most one switch box.

On the other hand, compute for each 0-entry the number of switch boxes it participates in and denote by $\alpha = \max\{s(i,j) \mid a_{i,j} = 0\}$ the maximum. To eliminate all switch boxes, at least one entry must be flipped from each of them. Then flipping one entry eliminates at most α switch boxes, and the minimum number of flips needed is at least k/α.

Combined, the approximation factor for the algorithm is 2α. Diagonal square matrices are tight examples. □

4.4.3 Greedy upper bound algorithm

The third method, `GreedyNested` [MT07], is a heuristic algorithm designed to produce an upper bound to d_{A}-Closest Nested and d_{H}-Closest Nested. Although no approximation factor has been shown for the algorithm, it provides fair results in practice [MT07]. In what follows is a

minor generalization (Algorithm 11) of the algorithm, which accepts arbitrary positive weights in the Closest Nested problem. The idea behind the algorithm is to flip entries one by one until the matrix is nested. The selection of flipped entries is based on the number of switch boxes they belong to.

Algorithm 11 `GreedyNested`

Input: binary matrix A, positive weights W
Output: $(d, \widehat{B})$, where d is an upper bound for $d_W(A, B)$; $\widehat{B}$ is close to a d_W-closest nested matrix B

1: $d \leftarrow 0$
2: $\widehat{B} \leftarrow A$
3: $S \leftarrow$ for each entry in $\widehat{B}$, the number of switch boxes it belongs to
4: **while** $\widehat{B}$ is not nested **do** // flip entries one by one until nested
5: $(r, c) \leftarrow \operatorname{argmax}_{(i,j)}\{s_{i,j}/w_{i,j}\}$ // best elimination efficiency
6: flip entry $\widehat{b}_{r,c}$
7: $d \leftarrow d + w_{r,c}$
8: update switch box counts in S
9: **end while**
10: **return** $(d, \widehat{B})$

On Line 4, nestedness can be checked by keeping track of the total count of switch boxes in $\widehat{B}$, and one evaluation takes constant time. On Line 8, the counts of switch boxes must be updated: for all eliminated switch boxes certain counts are decreased, and for all generated switch boxes certain counts are increased. The details of updating are the same as in the original algorithm [MT07].

In its original form [MT07], Algorithm 11 permits at most one flip per entry—this is supposed to avoid situations where the algorithm flips the same entries repeatedly and never stops. This may, however, lead to trouble: suppose that flipping an entry produces a switch box and the three other entries in this switch box have been flipped earlier. This switch box cannot be eliminated and the algorithm fails to produce a solution. Although unlikely, this phenomenon takes place occasionally with random 100×100 matrices. We can fix this by invoking Algorithm 10 whenever the situation occurs, which makes the matrix nested. Another fix would be to allow several flips per entry, but disallowing consecutive flips (and flip cycles).

4.5 Exact algorithms for closest nested

Although solving Closest Nested (Problem 4.7) is NP-hard, we can still solve instances of size up to, say, 20×20. We present here two new algorithms that use MAX-SAT solvers and a branch and bound technique.

Recall the SAT-formulae in Section 3.7.1. Given an $m \times n$ binary matrix A, we construct the following propositional logic formula in CNF that is satisfiable if and only if A is nested.

- *Entry variables* $e_{i,j}$ for each matrix entry $a_{i,j}$. Truth values on the entry variables represent matrix values 1 and 0. Contributes mn variables.

- *Entry clauses* $(e_{i,j})$ for each entry $a_{i,j} = 1$, and $(\neg e_{i,j})$ for each entry $a_{i,j} = 0$. The entry clauses reflect the values in A, and each unsatisfied entry clause corresponds to a flip in A. Contributes mn clauses.

- *Nested clauses* for each ordered pair of distinct rows (i, k) and columns (j, l) in A.

  $$\text{No switch box: } \neg(e_{i,j} \wedge e_{k,l} \wedge \neg e_{i,l} \wedge \neg e_{k,j})$$
  $$\text{Same clause in CNF: } (\neg e_{i,j} \vee \neg e_{k,l} \vee e_{i,l} \vee e_{k,j})$$

  Each nested clause prevents one type of switch box on specific rows and columns. Given an assignment, the corresponding matrix is nested if and only if all nested clauses are satisfied. Contributes $n(n-1)m(m-1)$ clauses.

The Hamming distance from A to a closest nested matrix can be computed by a partial MAX-SAT solver—just as in Section 3.7.1 to solve C1P instances. Using the above formula for A, we search for an assignment that satisfies all nested clauses and the number of satisfied entry clauses is maximum. This assignment corresponds to a nested matrix that is d_{H}-closest to A, and their Hamming distance is the number of unsatisfied entry clauses.

Solving the problem on the weighted distance d_W follows the same guidelines as in the C1P solution. Each entry clause has the same weight as the corresponding entry in A, and all nested clauses have infinite weights (larger than the sum of all weights in W). A weighted SAT solver then produces the weighted distance from A to a closest nested matrix: the sum of weights on unsatisfied entry clauses.

An alternative way to construct the formula is to establish total orders on both the rows and columns, which needs pairwise order variables and both antisymmetry and transitivity clauses as in the C1P formula. Instead

of using nested clauses, we now ensure that the total orders establish direct nestedness: for each 1-entry all entries above it and to the left from it are 1s; for this we need $mn(m-1) + mn(n-1)$ clauses. Compared to the formula above, this formula has more variables but the number of clauses is asymptotically smaller.

Another exact algorithm for Closest Nested (Problem 4.7) uses a branch and bound technique much like it was used to solve C1P instances in Section 3.7.2. Recall that Algorithm 9 produces lower bounds for distances to nested matrices. The branch and bound algorithm flips entries in the matrix and continues flipping one by one until it has *completed* that branch of flips, after which the algorithm cancels the latest flip and continues flipping other entries. Completing a branch happens in two ways. First, if the encountered matrix is nested, that branch is completed and the best distance so far is updated. Second, at some point the estimated distance, that is the distance from the flips so far plus the lower bound, is larger than the best distance found so far. In that case the branch is completed, as it has only worse solutions. In the end the algorithm has completed all branches, and it has found a closest nested matrix.

In an implementation of the branch and bound algorithm two optimizations should be included. First, the algorithm should start flipping from the entries that are included in many switch boxes. Second, we can complete many early branches quickly, if we run a heuristic algorithm beforehand, such as `GreedyNested` (Algorithm 11), and at the start of the algorithm we treat this upper bound as the best solution found so far.

4.6 Nested submatrices

Instead of finding nested matrices that are close to a binary matrix A, we can seek submatrices of A that are (almost) nested. For example, in a dataset that combines data from many sources, perhaps only a part of the attributes conforms to nestedness—the challenge is to identify this pattern. We study three types of submatrix problems. In the first the goal is to find a maximum set of columns that induces a nested submatrix. In the second the goal is to remove as few rows or columns as possible so that the matrix becomes nested. In the third problem we study how to find almost nested submatrices that have a high utility value.

Problem 4.11 (Maximum Columns Submatrix**)** *Given a binary matrix A and the set of its columns C, find a subset $D \subseteq C$ such that the submatrix induced by the columns in D is nested and $|D|$ is maximum.*

Problem 4.11 can be seen as removing as few columns from A as possible so that A becomes nested. Let us view the problem from the perspective of graphs. An *inclusion graph* is a directed graph $G = (V, E)$, where vertices in V represent columns in A and a directed edge $(i, j) \in E$ indicates that the column j is included in i, which means $C_j \subseteq C_i$ in terms of set interpretation of columns. Given an $m \times n$ matrix, we have $|V| = n$ and $|E| \leqslant n^2$.

Now any path in the graph G forms a chain of columns, and therefore the included columns form a nested submatrix. Finding the largest set of columns is the same as solving the Longest Path problem on G. Although this problem is NP-hard for general graphs, a well-known algorithm—via topological sorting and dynamic programming—finds a longest path in time $\mathcal{O}(|V| + |E|)$ if the graph is acyclic.

We observe that G is almost acyclic: the only problem is that identical columns have directed edges in both directions. We modify the inclusion graph G above to produce an *acyclic inclusion graph* $G' = (V, E')$ as follows. We assume that the vertices have been labeled $V = \{v_1, v_2, \ldots, v_n\}$, and we include in E' a directed edge (i, j) if and only if $C_j \subset C_i$ or both $C_j = C_i$ and $j < i$. This only restricts the order in which identical columns can appear in a path, and in particular it does not change the lengths of longest paths where each vertex appears at most once. Constructing the acyclic inclusion graph G' by a simple method takes time $\mathcal{O}(mn^2)$, and Problem 4.11 can be solved in time $\mathcal{O}(mn^2)$.

In the next problem we are allowed to remove rows in addition to columns.

Problem 4.12 (Maximum-Size Nested Submatrix) *Given a binary matrix A, find in A a submatrix that is nested and maximizes $a + b$, where a is the number of rows in the submatrix and b that of columns.*

Problem 4.12 is a special case of Problem 2.7, but with a specific pattern, nestedness. We establish the NP-hardness of this problem by using Theorem 2.8 and two matrix families, upper triangular and zero diagonal (page 35).

Theorem 4.13 Maximum-Size Nested Submatrix *is NP-hard.*

Proof. Let $\mathcal{M}$ be the collection of all nested matrices. We observe that $\mathcal{M}$ is nontrivial: upper triangular matrices are nested, but zero diagonal matrices of size at least 2×2 are not nested. By definition of nestedness, $\mathcal{M}$ is closed under permutation of rows and columns. It is also closed under deletion of the rows and columns, because nestedness has a forbidden submatrix characterization. The result follows from Theorem 2.8, because $\mathcal{M}$ contains upper triangular matrices that have unbounded rank. □

An earlier attempt [MT07] to prove Theorem 4.13 had the correct result but the argumentation was based on NP-completeness results on general graphs [LY80]. This argumentation is not valid, as binary matrices are presented as bipartite graphs, and the time complexity results for general graphs are upper-bound results for bipartite graphs.

A more practical problem is to find almost nested submatrices that have maximum utility with respect to a utility function f.

Problem 4.14 (ALMOST NESTED SUBMATRIX) *Given a binary matrix A, a nonnegative weight matrix W, and a fixed distance d, find a submatrix S of A that has distance to nestedness at most d and maximizes the utility $f(S)$.*

Recall Algorithm 1, `FindSubmatrix`, that works by removing rows and/or columns one by one based on a utility function. We can invoke the algorithm to find large submatrices that are almost nested, but we need to choose the utility function f first. Given a submatrix S of A, we can compute the utility value for each row/column i separately and then define $f(S)$ as the minimum of these values. Examples of utility measures for the row/column i follow.

- The inverse of the number of switch boxes the entries in the row/column i belong to.
- The inverse of the number of entries `GreedyNested` (Algorithm 11) flips on the row/column i. This is the same as $1/(\mathsf{fp} + \mathsf{fn})$ in terms of false positives and negatives.
- An information retrieval measure recall, precision, or accuracy, as presented in (2.2) on page 20. We obtain the ground-truth nested matrix (or its approximation) from `GreedyNested`.

Using precision produces small but dense submatrices, whereas removing the row/column that has the most flips produces a submatrix that is large and sparse. It depends on the application whether dense or large nested submatrices are preferred, and expert consultation is needed to choose a suitable utility function.

Chapter 5

Significance testing for nestedness

In this chapter we assume a less technical view on nestedness that focuses on the interpretation and applications in ecology. We explain the concept of nestedness from an ecological viewpoint and review the null models that are used for statistical significance testing. We provide tools for testing how effective the null models are in detecting an almost nested pattern if it exists, and how prone the models are to suggest almost nestedness when it does not exist. By conducting a series of experiments, we show that many popular nestedness measures and null models have severe shortcomings, and noise-tolerant methods should be preferred.

5.1 Nestedness in ecology

It is common in ecology that the distribution of species and sites is described by a presence/absence matrix, with the rows corresponding to sites (locations) and the columns corresponding to species (or more generally taxa). These biogeographical matrices are binary: a 1 corresponds to the presence of the species at the site, while a 0 indicates absence of the species. By absence we mean that the species is not present or it has not been observed yet. In addition to biogeographical data, species interactions can also be described in matrix form. Bipartite graphs have been used to represent both types of data [LIPJ$^+$06, ANGL07, BJMO03, RJB07].

Understanding the structure of interaction and biogeographical binary matrices is an important task in ecology. The nestedness pattern, as introduced by Patterson and Atmar [PA86] in an ecological context, describes co-occurrences where species compositions are proper subsets of those on sites with greater species richness. In this chapter, we use *nestedness* as a descriptive word, as it is used in ecology, and use the term *fully nested*

whenever we refer to perfect nestedness in the sense of Definition 4.2.

Nestedness has been observed for several taxonomic groups [FL05]. Possible mechanisms leading to nestedness include differential colonization, differential extinction, and hierarchical habitat distributions [HWS06]. More possible causes for nestedness have been found [WPM+98, UANG09], such as sampling bias, habitat fragmentation, distance from a source of colonists, and island area. In addition, conflicting phenotypic traits of plants and animals, such as their physical size, leads to nestedness [RJB07]. In biodiversity conservation management, identifying the nested patterns in data has implications on selecting the areas for conservation [SAP10, FL05, Pat87]. Antinestedness, the concept of which still lacks an exact meaning [ANGL07], has been used to describe data that are less nested than expected by chance.

As nestedness observed in nature is rarely perfect, lots of effort has been made to define measures that evaluate how far a binary matrix is from being fully nested. During the past 15 years, the most widely established nestedness measure has been `Temperature` [AP93, ANGL07, RGS06]. Other measures in this study include, for historic reasons, `NO` [PA86] and `N1` [UG07b], as well as more recently developed measures `Nodf` [ANGG+08], `Discrepancy` [BS99], and Hamming distance (`Hamming`) [MT07]. This set of measures is, in our opinion, a representative sample of all measures developed for nestedness.

A nestedness measure produces a distance, but it does not tell whether this distance is exceptionally small, that is, if data is strongly or significantly nested. In Section 2.6 we suggested using a null model to assess the statistical significance of the results from nestedness measures. A null model generates random matrices that share certain characteristics with the original data. Here we study a wide range of null models, namely `Fill` [WPM+98], `R0` [PA86], `R1` [PA86], `R2` [WPM+98], `Contin` [GD82], `Bascompte` [BJMO03], and `Swaps` [CC03]. To our knowledge, `Bascompte` has not been analyzed against other methods earlier, although it is still in use [SF07].

While the intuition behind nestedness measures and null models is simple, the actual selection of methods for a particular application is not. For example, skewed distributions of species abundances can lead to patterns that can be interpreted as nested, even when the species occurrences are independent [HWS06]. The selection of null models and nestedness measures has been demonstrated to influence the results on parasite datasets [TP07]. The individual contribution of each row and column in the observed matrix can also be addressed [SF07].

Differences between nestedness measures and null models have been

studied earlier [WPM+98], and recent studies [UG07b, UANG09, UG07a] have identified shortcomings in most nestedness methods. The effects of matrix shape, size, and fill have also been considered. Measures `NO`, `N1`, and null models `Fill`, and `RO` have all been found inadequate by earlier studies, but are still included here for the sake of comparison.

The effect of sampling errors on nestedness detection has been studied to some extent [NB07, CNHS00], but the overall effect of errors and noise on nestedness analysis is still unclear. We use the error models described in Section 2.5 that simulate the effects that degrade the quality of data. We study the sensitivity of nestedness methods to noise in the data in order to see how strongly the results depend on the quality of the data. A nestedness measure should be resistant to small amounts of errors or noise in the data. That is, if two binary matrices A and B differ from each other in only a few entries, we would expect the values of nestedness measures for A and B to be close to each other. We do not want a small set of observational errors to completely change our views on the nestedness (or non-nestedness) of the data. It turns out that there are fundamental differences in the way noise affects the measures and null models.

The goal in nestedness analyses is to better understand the processes that produce the data. Before analyzing the reasons for nestedness on a specific dataset, however, it is necessary to ensure that such a pattern exists in the data. In this study we concentrate on recognizing whether a dataset represents a nested pattern regardless of noise, and leave aside the discussion on the reasons for nestedness.

5.2 Methods and datasets

Next we describe the nestedness measures and null models that once were in common use and those we believe to be the most widely used today. We also include a new measure for nestedness, `Hamming`, which has not been used in nestedness analyses before. It is the same as `GreedyNested` (Algorithm 11) with Hamming distance: the minimum number of flips needed to make data fully nested. Then we introduce the statistics and noise models that we use to evaluate the noise-tolerance of the nestedness measures and null models. We also give the details of the datasets that we used in the experiments.

5.2.1 Measures of nestedness

Given an $m \times n$ binary matrix A, a nestedness measure produces a value that describes how far A is from perfect nestedness. In all measures presented here, value 0 refers to perfect nestedness.

`NO`: The measure `NO` counts the number of false absences [PA86]. A 0-entry $a_{r,c}$ is considered *false* if there exists another row s in A that has a smaller row sum than r and $a_{s,c} = 1$. The number of false 0s in A is `NO`.

`N1`: In contrast to `NO`, the measure `N1` counts the number of false presences [UG07b]. A 1-entry $a_{r,c}$ is considered *false* if there exists another row s in A that has a larger row sum than r and $a_{s,c} = 0$. The number of false 1s in A is `N1`.

`Temperature (T)`: In `Temperature` the idea is to measure disorder in a given matrix by assessing the deviation of the matrix from one that has the same rank and fill and is fully nested [AP93]. The computation of `Temperature` is done in three steps [RGS06].

1. Compute an isocline of perfect order. This is a curve that separates 1s from 0s in a matrix of the same size as A that is fully nested and has as many 1s as A.

2. Reorganize the matrix. This is done by permuting the rows and columns in a way that maximizes its directly observable nestedness.

3. Associate with each absence above the isocline and with each presence below it a normalized measure of distance to the isocline. The `Temperature` of the matrix is the sum of these distances, over all entries. The `Temperature` measure is normalized so that it ranges from 0 to 100 (from a fully nested to a maximally non-nested matrix).

For evaluating the `Temperature` of the input matrices we have used the code provided by the authors of [RGS06] with default parameters.

`Discrepancy (D)`: One first sorts the rows and columns of A in nonincreasing order by their row sums and column sums. Denote the matrix obtained by this reordering by $\bar{A}$. The row-discrepancy for a single row r is the number of 0s among the first k positions on the row r, where k is the row sum of r. The `Discrepancy` [BS99] of A is then the sum of all row-discrepancy values in $\bar{A}$. `Discrepancy` captures the following intuition: a binary matrix is nested if the 1s in each row are as far to the left as possible and the 1s in each column are as near the top as possible. The `Discrepancy` of a matrix, however, is implementation-dependent: the value is not necessarily unique if several columns share a common column sum. Therefore the results from using `Discrepancy` may be hard to replicate, and it should be considered an approximative measure, not exact.

`Hamming (H)`, Hamming distance to nestedness: The `Hamming` measure is the minimum number of flips needed to make A fully nested [MT07], which involves solving d_{H}-Closest Nested (Problem 4.7). Since no polynomial-time exact algorithm is known for the problem, we use `GreedyNested` (Algorithm 11) to measure the distance.

`Hamming` resembles `Discrepancy` in the following way: both measures count how many modifications are needed to make a matrix fully nested. The results from `Hamming` are unique: they never change because of row and column ordering, in contrast to `Discrepancy`. The reordering step is also different: `Discrepancy` reorders the matrix before modifications, based on possibly erroneous data, whereas `Hamming` makes the modifications first and reorders the matrix according to the uncovered nested pattern.

A variant of `Hamming` can also take into account the specific characteristics of species and sites. Weights, which represent the degree of certainty that the data is correct, can be assigned to the entries (Section 2.2).

`Nodf (F)`, Nestedness metric based on overlap and decreasing fill: Consider an unordered pair of rows $\{i, j\}$, where i has a strictly larger row sum than j has. The degree of nestedness $DN_{\{i,j\}}$ for the pair is the percentage of 1s on the row j that have 1s on the row i at identical positions. If the rows i and j have the same row sums, then $DN_{\{i,j\}} = 0$. There are $m(m-1)/2$ row pairs in total.

The degree of nestedness for column pairs $\{k, l\}$: transpose the matrix and treat columns as rows $\{i, j\}$ above. There are $n(n-1)/2$ column pairs.

The $\texttt{Nodf}_{\text{orig}}$ value among both rows and columns is then the average degree of nestedness over all $m(m-1)/2 + n(n-1)/2$ unordered pairs [ANGG$^+$08]. We will use values $\texttt{Nodf} = 100 - \texttt{Nodf}_{\text{orig}}$ to achieve consistency with the other measures (`Nodf` is 0 for fully nested matrices).

5.2.2 Null models

Null models are used to compare observed species distributions or interactions to those generated by a random model or process. The choice of model plays a major role: a particular pattern found in data, such as nestedness, is significant only if it appears as exceptional with respect to the model. The background of significance testing can be found in Section 2.6. Here we employ a wide range of null models to test nestedness measures. Given a binary dataset A, the models produce randomized matrices as follows.

`Fill:` This model generates a random dataset that has the same number of 1s as in A. The 1s are uniformly distributed. Also known as `R00` [WPM$^+$98].

`R0`: For each row (site) of the matrix, we draw columns (species) from a uniform probability distribution until the row sum in A (species richness) is reached [PA86]. For each row, we sample columns without replacement, and set all sampled entries to 1. Also known as `RANDOM0`.

`R1`: The generation process resembles that of `R0`. For every row, we sample columns with probabilities proportional to their column sums in A until the number of sampled columns equals the row sum in A [PA86]. For each row we sample columns without replacement, and set sampled entries to 1. Also known as `RANDOM1`.

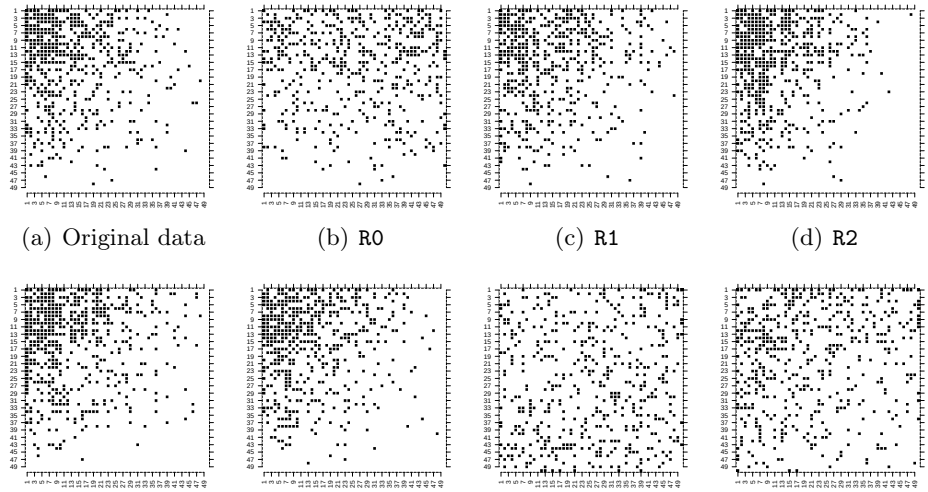
`R2`: This generative model is almost the same as `R1`. The only difference is that in `R2` the probability of sampling a column in a row is proportional to the squared column sum [WPM$^+$98].

`Contin`: In this model [GD82], we associate a value $v(i,j) = r_i c_j / F$ with each matrix entry (i,j), where r_i is the row sum of i, and c_j is the column sum of j, and F is the total number of 1s in A. Entry (i,j) takes value 1 with probability $\min\{v(i,j), 1\}$.

`Bascompte`: This model [BJMO03] uses an empirical probability of occurrence on each row and column, which is the proportion of 1s among all values on that row or column. On matrix entry (i,j) the probability of value 1 is the average of empirical probabilities of the row i and column j.

`Swaps`: We sample random matrices that have the same row sums and column sums as A. This method is known as the fixed-fixed model [Got00] or swap randomization [CC03, MP04], and is based on Ryser's Theorem 4.3. The method seeks randomly two rows r, s and two columns c, d that form a switch box (matrices in (4.2)) in A, and then makes a switch (flips these four entries). This procedure is repeated until the initial matrix A has a negligible effect on the randomized matrix (close to random); in our experiments the number of repetitions was 10 times the number of 1s in A. By Theorem 4.4, `Swaps` is not able to randomize a fully nested matrix.

Figure 5.1 displays an example of a data matrix and randomized matrices generated from it by using each of these null models. The randomized matrices share certain statistical properties with the data matrix A. `Fill` generates matrices with the same proportion of 1s as A; `R0`, `R1`, and `R2` generate matrices with the same row sums as A (and hence with the same proportion of 1s); `Contin` generates matrices with approximately the same



(a) Original data (b) `R0` (c) `R1` (d) `R2`

(e) `Contin` (f) `Swaps` (g) `Fill` (h) `Bascompte`

Figure 5.1: Displayed in (a) is a dataset of size 50×50, and in (b–h) the data has been randomized with different null models.

row sums and column sums as A; `Swaps` preserves the row sums and column sums exactly. In `Fill`, `Contin`, `Bascompte`, and `Swaps` the roles of rows and columns are the same in the generation process, which is desirable, as the nestedness pattern does not distinguish between the two. Nevertheless, taking the specific nature of sites and species into account in the null model adds complexity, and may not have use outside that particular biogeographical context.

In this thesis we assume that the data consists of binary values. If the data, however, had abundance data of species instead of presence/absence values, we could use an ecologically more explicit null model, as suggested by Moore and Swihart [MS07]. It generates random matrices not based on the observed values, but by using expected values of species richness and abundance. This topic has been studied further and a nestedness measure also for non-binary datasets has been proposed [GPI09].

5.2.3 Significance of nestedness

A nestedness measure alone does not give an indication of the relevancy of nestedness. In the following we bring out two performance indicators that determine how strong a nested pattern is and whether it is statistically significant.

Given an input matrix A, a nestedness measure $\mathtt{N} \in \{\mathtt{NO}, \mathtt{N1}, \mathtt{Temperature}, \mathtt{Discrepancy}, \mathtt{Hamming}, \mathtt{Nodf}\}$ and a null model $\mathtt{R} \in \{\mathtt{Fill}, \mathtt{R0}, \mathtt{R1}, \mathtt{R2}, \mathtt{Contin}, \mathtt{Bascompte}, \mathtt{Swaps}\}$, we generate random instances $A_{\mathtt{R}}$ of A by using the null model $\mathtt{R}$, and compare the nestedness values $\mathtt{N}(A_{\mathtt{R}})$ against the value $\mathtt{N}(A)$ of the measure on the original data. Small values of $\mathtt{N}(A)$ indicate strong nestedness; the value 0 stands for perfect nestedness.

To test whether a nested pattern in A is significant, we compute the empirical p-value, denoted by $p(A, \mathtt{N}, \mathtt{R})$ for nestedness measure $\mathtt{N}$ and randomization method $\mathtt{R}$. As in Section 2.6, the value is computed by counting the fraction of the random instances $A_{\mathtt{R}}$ that have $\mathtt{N}(A_{\mathtt{R}}) \leqslant \mathtt{N}(A)$. In other words, $p(A, \mathtt{N}, \mathtt{R})$ indicates how likely it is that an instance generated from the null model is considered at least as nested as the original data. A small p-value means that rejecting the null hypothesis in favor of nestedness poses only a small risk of Type I error.

We also introduce the measure of *nestedness intensity* $I(A, \mathtt{N}, \mathtt{R})$,

$$I(A, \mathtt{N}, \mathtt{R}) = \frac{\mathtt{N}(A)}{E(\mathtt{N}(A_{\mathtt{R}}))}, \tag{5.1}$$

where we estimate expectation E by sample mean. An intensity value close to 0 indicates that A has a strong nested structure, whereas a value close to 1 indicates that A is non-nested. Values greater than 1.0 indicate that the original dataset is even less nested (when measured by $\mathtt{N}$) than random instances produced by $\mathtt{R}$, which is known as anti-nestedness [ANGL07].

These two measures, intensity and p-value, have different roles: nestedness intensity indicates how strongly the data are nested with respect to the null model used, whereas p-value shows whether the pattern is statistically significant. Indeed, a weak pattern may be considered significant if the sample size is large enough.

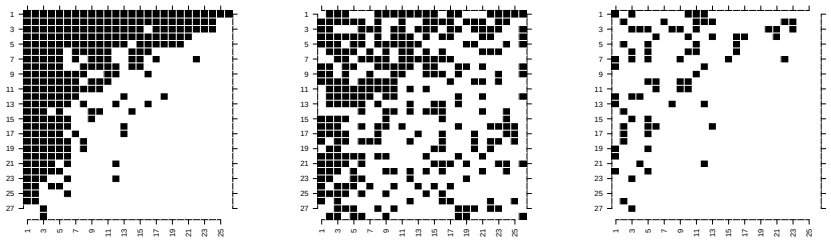
5.2.4 Data and error models

We test all combinations of nestedness measures and null models on a variety of datasets: Rasch-type matrices, real-world data on Rocky Mountain mammals, and synthetic matrices that contain noise and errors. The matrices in the first dataset are non-nested, while the second and third datasets are nested up to some noise level. As in Section 2.5, we use the symmetric error model (both 1-to-0 and 0-to-1 errors) and the asymmetric error model (only 1-to-0 errors). The noise level for symmetric noise is denoted by $\ell = \Pr(\text{0-to-1}) = \Pr(\text{1-to-0})$, whereas $h = \Pr(\text{1-to-0})$ shows the level of asymmetric noise. Next we describe the datasets in detail.

We use a model close to the Rasch model [Ras60, FM95] to generate random matrices that are non-nested. We sample two sets of 100 real numbers from the interval $[0, 1]$, one for rows $r_1, \ldots, r_{100}$ and one for columns $c_1, \ldots, c_{100}$. Then a 100×100 matrix is generated, with entry (i, j) equal to 1 with probability $r_i c_j$. Figure 5.1(a) shows an example of a 50×50 matrix generated by this method, with rows and columns ordered by r and c. Because in Rasch matrices the 1s on rows and columns are conditionally independent, given the values r and c, they bear no structure that is specifically characteristic of nestedness. Therefore Rasch matrices should be identified as non-nested by any reasonable method.

As a particular example of a real-world dataset that is strongly nested we use the data about the mammals on the Rocky Mountains [PA86]. The dataset, consisting of 28 sites and 26 species, corresponds to Boreal and Boreo-Cordilleran species of mammals in the Southern Rocky Mountains. Figure 5.2(a) displays the dataset. Figures 5.2(b,c) show the effect of symmetric noise at the level $\ell = 0.32$ and that of asymmetric noise at the level $h = 0.70$.

To study the effects of noise in a controlled fashion, we use the following procedure to generate synthetic data that is fully nested. Given the number of rows m and columns n, we sample a random number k uniformly from $\{0, 1, \ldots, n\}$ for each row and set the first k entries on that row to 1 and set the rest to 0. We then add symmetric or asymmetric noise at various levels to test how the nestedness methods handle noise. The size of the synthetic matrices is 100×100, which is a typical size for an ecological dataset.



(a) Original data (b) With noise $\ell = 0.32$ (c) With noise $h = 0.70$

Figure 5.2: Examples of noisy datasets on Rocky Mountain mammals. On the left (a) is the original data [PA86]; in the middle (b) is the data with $\ell = 0.32$ symmetric noise; on the right (c) is the data with $h = 0.70$ asymmetric noise. All matrices share the same row and column ordering, which shows how the true underlying nested pattern appears as noisy data.

5.3 Results

The following sections show the experimental results for Rasch, Rocky Mountain mammals, and synthetic datasets. The Rasch experiment tests whether the methods are able to detect non-nestedness, and the others test how well the methods recognize underlying nestedness. There are major differences in the behavior of nestedness measures and null models.

5.3.1 Results on Rasch data

In the Rasch experiment the datasets are assumed non-nested. We generated 100 Rasch matrices and randomized each one of them 100 times with a null model, yielding 10,000 samples. Figure 5.1 displays examples of randomized Rasch matrices, and Table 5.1 shows the results of the Rasch experiment. The generated datasets are considered significantly nested by each measure for the null models `Fill`, `Bascompte` and `R0`, implying that Type I errors are frequent. `Swaps` and `Contin` correctly identify Rasch matrices as non-nested, whereas the matrices are considered less nested than expected by `R2`. Using `R1` gives both nested and non-nested results, depending on the measure used. Of the measures, `Discrepancy`, `Hamming`, and `Nodf` behave in a similar fashion, all having good performance.

	R0		R1		R2		Contin		Swaps		Fill		Basc	
	I	p	I	p	I	p	I	p	I	p	I	p	I	p
`N0`	.90	.00	1.00	.55	1.20	1.00	1.03	.85	1.04	0.90	.74	.00	.78	.00
`N1`	.96	.00	0.98	.03	1.04	0.99	1.03	.88	0.98	0.05	.93	.00	.94	.01
`Disc`	.73	.00	0.93	.00	1.21	1.00	1.03	.92	0.99	0.31	.65	.00	.73	.00
`Hamming`	.78	.00	0.94	.00	1.18	1.00	1.03	.92	0.98	0.04	.72	.00	.75	.00
`Temp`	.50	.00	0.90	.00	1.29	1.00	1.01	.59	1.04	0.97	.34	.00	.47	.00
`Nodf`	.83	.00	0.97	.11	1.18	1.00	1.02	.96	1.02	1.00	.74	.00	.81	.00

Table 5.1: Displayed are the nestedness intensities $I(A, N, R)$ and significance values $p(A, N, R)$ for measures and null models on the Rasch data. The expected intensity is close to 1, which translates into non-nestedness.

5.3.2 Results on Rocky Mountain data

The results of adding symmetric noise to the Rocky Mountain data are shown in Table 5.2. The datasets should be considered nested up to some noise level, and in general, the null models tend to follow the ordering: `Fill`, `Bascompte`, `R0`, `R1`, `Contin`, and `R2`, meaning that the p-value and the nestedness intensity increase when moving from one model to the next in

this ordering. `Fill` is the most lenient, in that it treats some datasets as nested when the other null models do not.

Null model `Swaps` suggests that the Rocky Mountain data is not significantly nested at any symmetric noise level with most methods, which is counter-intuitive. It is known that `Swaps` cannot distinguish a fully random data from a fully nested one, but this experiment shows that the underlying nestedness can be almost completely missed, regardless of noise level. Only with `Hamming` is `Swaps` able to recognize significant nestedness consistently, although nestedness intensity is identified as weak.

At noise level $\ell = 0.32$, as displayed in Figure 5.2(b), the results show large differences between the null models: `Fill` and `R0` consider the data still very nested for `Discrepancy`, `Hamming`, `Nodf`, and `Temperature`, while `R2` considers the data anti-nested. Null model `R1` together with measures `N0` and `N1` yield non-nested results already around $\ell = 0.08$, whereas all other measures require a noise level between $\ell = 0.16$ and $\ell = 0.32$ to reject nestedness, which is more reasonable.

Null model `R2` gives a non-nested result already at low noise levels ℓ, and, as noted, suggests that at noise level $\ell = 0.32$ the Rocky Mountain matrices are less nested than on the average. The `Contin` results lie between those of `R1` and `R2`.

The results for the asymmetric noise model in Table 5.3 follow the same pattern: `Fill`, `Bascompte` and `R0` are the most lenient null models and `R2` is the most conservative. Again, the `Swaps` method stands out against the other null models. The nestedness measures indicate that when asymmetric noise reaches level $h = 0.5$, nestedness has vanished from the data. The exception is `Temperature`: for `R1` and more lenient null models it considers the data nested even at level $h = 0.70$, which is intuitively hard to accept, given Figure 5.2(c).

5.3.3 Results on synthetic data

Figures 5.3(a–g) show the results for the synthetic experiment where symmetric noise was applied to fully nested matrices. We expect that the best methods consider the datasets nested up to some noise level.

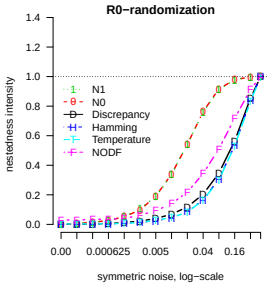
The behavior of the null models `Fill`, `R0`, and `Bascompte` in Figures 5.3(a,f,g) resemble each other under symmetric noise. With these models, the measures `N0` and `N1` fail to recognize nestedness (intensity close to 1) even if the noise level remains fairly low ($\ell < 0.04$). Measures `Discrepancy`, `Hamming` and `Temperature`, too, have only small differences under these null models, but they recognize nestedness correctly even at relatively high noise levels. `Nodf` jumps out as a compromise between the two extremes.

	R0		R1		R2		Contin		Swaps		Fill		Bascompte	
	intensity (intensity std)	*p*-value	intensity (intensity std)	*p*-value	intensity (intensity std)	*p*-value	intensity (intensity std)	*p*-value	intensity (intensity std)	*p*-value	intensity (intensity std)	*p*-value	intensity (intensity std)	*p*-value
$\ell = \mathbf{0.00}$														
NO	0.23 (.00) 0.00	0.27 (.00) 0.00	0.40 (.00) 0.00	0.27 (.00) 0.00	0.78 (.00) 0.03	0.17 (.00) 0.00	0.20 (0.00) 0.00							
N1	0.36 (.00) 0.00	0.43 (.00) 0.00	0.63 (.00) 0.00	0.47 (.00) 0.00	0.95 (.01) 0.40	0.24 (.00) 0.00	0.28 (0.00) 0.00							
Disc	0.30 (.00) 0.00	0.40 (.00) 0.00	0.61 (.00) 0.00	0.44 (.00) 0.00	0.91 (.00) 0.15	0.22 (.00) 0.00	0.28 (0.00) 0.00							
Hamming	0.28 (.00) 0.00	0.36 (.00) 0.00	0.57 (.00) 0.00	0.40 (.00) 0.00	0.95 (.00) 0.27	0.18 (.00) 0.00	0.24 (0.00) 0.00							
Temp	0.10 (.00) 0.00	0.19 (.00) 0.00	0.43 (.01) 0.00	0.19 (.00) 0.00	1.12 (.03) 0.81	0.05 (.00) 0.00	0.09 (0.00) 0.00							
Nodf	0.25 (.00) 0.00	0.34 (.00) 0.00	0.56 (.00) 0.00	0.32 (.00) 0.00	1.00 (.00) 0.94	0.19 (.00) 0.00	0.24 (0.00) 0.00							
$\ell = \mathbf{0.02}$														
NO	0.44 (.08) 0.00	0.51 (.08) 0.00	0.71 (.12) 0.04	0.51 (.09) 0.00	1.00 (.11) 0.51	0.35 (.06) 0.00	0.40 (.07) 0.00							
N1	0.53 (.08) 0.00	0.62 (.09) 0.00	0.87 (.13) 0.22	0.69 (.11) 0.02	0.99 (.12) 0.47	0.37 (.06) 0.00	0.42 (.07) 0.00							
Disc	0.38 (.02) 0.00	0.50 (.03) 0.00	0.74 (.04) 0.00	0.55 (.03) 0.00	0.91 (.04) 0.14	0.28 (.02) 0.00	0.36 (.02) 0.00							
Hamming	0.34 (.02) 0.00	0.43 (.02) 0.00	0.66 (.03) 0.00	0.48 (.03) 0.00	0.85 (.03) 0.02	0.23 (.01) 0.00	0.29 (.02) 0.00							
Temp	0.17 (.03) 0.00	0.31 (.05) 0.00	0.65 (.10) 0.02	0.33 (.06) 0.00	1.06 (.06) 0.74	0.11 (.02) 0.00	0.16 (.03) 0.00							
Nodf	0.36 (.04) 0.00	0.48 (.05) 0.00	0.76 (.08) 0.04	0.45 (.05) 0.00	1.04 (.04) 0.91	0.28 (.03) 0.00	0.35 (.04) 0.00							
$\ell = \mathbf{0.04}$														
NO	0.57 (.08) 0.00	0.65 (.08) 0.00	0.88 (.11) 0.23	0.66 (.08) 0.00	1.05 (.11) 0.64	0.46 (.07) 0.00	0.52 (.07) 0.00							
N1	0.64 (.09) 0.00	0.74 (.10) 0.02	1.01 (.14) 0.54	0.82 (.12) 0.12	1.01 (.12) 0.54	0.47 (.07) 0.00	0.52 (.07) 0.00							
Disc	0.44 (.03) 0.00	0.57 (.03) 0.00	0.83 (.05) 0.05	0.64 (.04) 0.00	0.92 (.04) 0.15	0.34 (.02) 0.00	0.42 (.03) 0.00							
Hamming	0.40 (.02) 0.00	0.50 (.02) 0.00	0.75 (.03) 0.00	0.56 (.03) 0.00	0.85 (.04) 0.01	0.28 (.02) 0.00	0.36 (.02) 0.00							
Temp	0.24 (.03) 0.00	0.41 (.05) 0.00	0.82 (.09) 0.14	0.46 (.06) 0.00	1.03 (.05) 0.69	0.15 (.02) 0.00	0.23 (.03) 0.00							
Nodf	0.45 (.04) 0.00	0.58 (.05) 0.00	0.89 (.06) 0.23	0.56 (.05) 0.00	1.04 (.04) 0.90	0.36 (.03) 0.00	0.43 (.04) 0.00							
$\ell = \mathbf{0.08}$														
NO	0.75 (.07) 0.00	0.82 (.07) 0.04	1.05 (.09) 0.66	0.85 (.08) 0.11	1.08 (.08) 0.77	0.64 (.07) 0.00	0.70 (.08) 0.00							
N1	0.78 (.07) 0.00	0.88 (.08) 0.12	1.15 (.10) 0.88	0.98 (.09) 0.47	1.02 (.08) 0.60	0.61 (.06) 0.00	0.66 (.06) 0.00							
Disc	0.54 (.04) 0.00	0.68 (.04) 0.00	0.95 (.05) 0.35	0.76 (.05) 0.00	0.92 (.04) 0.14	0.44 (.03) 0.00	0.53 (.04) 0.00							
Hamming	0.51 (.03) 0.00	0.62 (.04) 0.00	0.88 (.05) 0.12	0.69 (.04) 0.00	0.87 (.04) 0.01	0.38 (.03) 0.00	0.46 (.03) 0.00							
Temp	0.38 (.05) 0.00	0.58 (.05) 0.00	1.04 (.08) 0.63	0.68 (.07) 0.00	1.01 (.03) 0.58	0.26 (.04) 0.00	0.36 (.05) 0.00							
Nodf	0.60 (.05) 0.00	0.73 (.05) 0.00	1.05 (.06) 0.77	0.74 (.06) 0.00	1.04 (.02) 0.94	0.49 (.05) 0.00	0.58 (.05) 0.00							
$\ell = \mathbf{0.16}$														
NO	0.90 (.05) 0.05	0.94 (.05) 0.24	1.09 (.06) 0.85	0.99 (.06) 0.47	1.03 (.05) 0.69	0.81 (.06) 0.00	0.87 (.06) 0.04							
N1	0.91 (.06) 0.10	0.98 (.06) 0.46	1.18 (.08) 0.96	1.11 (.08) 0.84	1.02 (.06) 0.64	0.78 (.06) 0.00	0.82 (.06) 0.01							
Disc	0.70 (.04) 0.00	0.82 (.04) 0.00	1.08 (.04) 0.87	0.93 (.04) 0.22	0.96 (.03) 0.23	0.62 (.04) 0.00	0.70 (.04) 0.00							
Hamming	0.69 (.04) 0.00	0.78 (.03) 0.00	1.03 (.04) 0.69	0.89 (.04) 0.09	0.91 (.03) 0.03	0.56 (.04) 0.00	0.65 (.04) 0.00							
Temp	0.60 (.05) 0.00	0.78 (.04) 0.00	1.21 (.05) 0.97	0.95 (.06) 0.37	0.99 (.02) 0.41	0.45 (.05) 0.00	0.58 (.05) 0.00							
Nodf	0.77 (.04) 0.00	0.88 (.03) 0.01	1.13 (.02) 0.98	0.94 (.04) 0.27	1.01 (.01) 0.92	0.67 (.04) 0.00	0.76 (.04) 0.00							
$\ell = \mathbf{0.32}$														
NO	0.97 (.02) 0.30	0.99 (.02) 0.45	1.04 (.03) 0.79	1.04 (.03) 0.77	0.99 (.02) 0.49	0.95 (.03) 0.18	0.97 (.03) 0.36							
N1	0.99 (.02) 0.43	1.01 (.02) 0.67	1.08 (.03) 0.95	1.09 (.03) 0.92	1.01 (.02) 0.64	0.95 (.03) 0.17	0.97 (.03) 0.34							
Disc	0.90 (.03) 0.04	0.97 (.03) 0.38	1.14 (.03) 0.99	1.08 (.02) 0.94	0.99 (.02) 0.46	0.87 (.04) 0.01	0.92 (.03) 0.11							
Hamming	0.91 (.03) 0.03	0.97 (.02) 0.33	1.13 (.03) 0.99	1.09 (.02) 0.95	0.98 (.02) 0.40	0.86 (.04) 0.00	0.92 (.03) 0.08							
Temp	0.87 (.04) 0.01	0.98 (.03) 0.38	1.23 (.02) 0.99	1.19 (.03) 0.98	0.99 (.01) 0.39	0.79 (.06) 0.00	0.88 (.05) 0.08							
Nodf	0.93 (.02) 0.05	0.98 (.01) 0.49	1.10 (.01) 0.99	1.08 (.01) 0.96	1.00 (.00) 0.89	0.88 (.03) 0.01	0.93 (.03) 0.17							

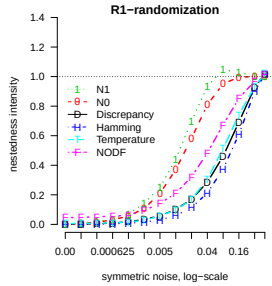
Table 5.2: Rocky Mountain data under symmetric noise. Randomization results for the data under various levels ℓ of symmetric noise (both 0-to-1 and 1-to-0 errors). We generated 100 noisy matrices from the original data, and randomized each of them 100 times. The tables shows the associated nestedness intensity, its standard deviation (std), and p-value.

	R0 intensity (intensity std) *p*-value	R1 intensity (intensity std) *p*-value	R2 intensity (intensity std) *p*-value	Contin intensity (intensity std) *p*-value	Swaps intensity (intensity std) *p*-value	Fill intensity (intensity std) *p*-value	Bascompte intensity (intensity std) *p*-value
h = 0.00							
N0	0.23 (.00) 0.00	0.27 (.00) 0.00	0.40 (.00) 0.00	0.27 (.00) 0.00	0.78 (.00) 0.03	0.17 (.00) 0.00	0.20 (.00) 0.00
N1	0.36 (.00) 0.00	0.43 (.00) 0.00	0.63 (.00) 0.00	0.47 (.00) 0.00	0.95 (.01) 0.40	0.24 (.00) 0.00	0.28 (.00) 0.00
Disc	0.30 (.00) 0.00	0.40 (.00) 0.00	0.61 (.00) 0.00	0.44 (.00) 0.00	0.91 (.00) 0.15	0.22 (.00) 0.00	0.28 (.00) 0.00
Hamming	0.28 (.00) 0.00	0.36 (.00) 0.00	0.57 (.00) 0.00	0.40 (.00) 0.00	0.95 (.00) 0.27	0.18 (.00) 0.00	0.24 (.00) 0.00
Temp	0.10 (.08) 0.00	0.20 (.16) 0.01	0.47 (.37) 0.01	0.20 (.16) 0.01	1.11 (.04) 0.80	0.06 (.05) 0.00	0.09 (.00) 0.00
Nodf	0.25 (.00) 0.00	0.34 (.00) 0.00	0.56 (.00) 0.00	0.32 (.00) 0.00	1.00 (.00) 0.94	0.19 (.00) 0.00	0.24 (.00) 0.00
h = 0.08							
N0	0.31 (.02) 0.00	0.37 (.03) 0.00	0.53 (.04) 0.00	0.38 (.03) 0.00	0.69 (.05) 0.00	0.24 (.02) 0.00	0.28 (.02) 0.00
N1	0.73 (.09) 0.00	0.88 (.10) 0.19	1.24 (.14) 0.92	0.98 (.12) 0.46	1.24 (.10) 0.94	0.54 (.07) 0.00	0.59 (.07) 0.00
Disc	0.40 (.03) 0.00	0.54 (.04) 0.00	0.81 (.05) 0.05	0.61 (.04) 0.00	0.92 (.05) 0.21	0.30 (.02) 0.00	0.38 (.03) 0.00
Hamming	0.37 (.02) 0.00	0.49 (.03) 0.00	0.75 (.05) 0.01	0.54 (.04) 0.00	0.88 (.04) 0.05	0.26 (.02) 0.00	0.33 (.02) 0.00
Temp	0.16 (.02) 0.00	0.30 (.03) 0.00	0.61 (.06) 0.01	0.31 (.04) 0.00	0.91 (.07) 0.21	0.10 (.01) 0.00	0.15 (.02) 0.00
Nodf	0.36 (.03) 0.00	0.48 (.03) 0.00	0.76 (.05) 0.03	0.46 (.04) 0.00	0.95 (.02) 0.40	0.29 (.02) 0.00	0.35 (.03) 0.00
h = 0.16							
N0	0.37 (.03) 0.00	0.44 (.03) 0.00	0.64 (.05) 0.00	0.47 (.04) 0.00	0.66 (.05) 0.00	0.29 (.02) 0.00	0.33 (.02) 0.00
N1	0.88 (.07) 0.06	1.05 (.08) 0.73	1.45 (.12) 0.99	1.19 (.10) 0.90	1.24 (.08) 0.97	0.69 (.07) 0.00	0.74 (.07) 0.00
Disc	0.49 (.03) 0.00	0.66 (.04) 0.00	0.98 (.07) 0.47	0.75 (.06) 0.00	0.95 (.04) 0.29	0.39 (.03) 0.00	0.48 (.04) 0.00
Hamming	0.48 (.04) 0.00	0.61 (.05) 0.00	0.93 (.07) 0.32	0.70 (.06) 0.00	0.90 (.04) 0.10	0.34 (.03) 0.00	0.42 (.04) 0.00
Temp	0.22 (.03) 0.00	0.39 (.04) 0.00	0.75 (.08) 0.08	0.42 (.06) 0.00	0.84 (.06) 0.06	0.14 (.02) 0.00	0.21 (.03) 0.00
Nodf	0.46 (.04) 0.00	0.61 (.05) 0.00	0.93 (.06) 0.34	0.60 (.06) 0.00	0.93 (.02) 0.14	0.38 (.04) 0.00	0.45 (.04) 0.00
h = 0.32							
N0	0.45 (.04) 0.00	0.55 (.04) 0.00	0.79 (.07) 0.08	0.61 (.05) 0.00	0.66 (.05) 0.00	0.35 (.03) 0.00	0.41 (.04) 0.00
N1	0.95 (.04) 0.20	1.10 (.04) 0.93	1.48 (.08) 1.00	1.29 (.07) 0.99	1.15 (.05) 0.96	0.82 (.05) 0.00	0.86 (.05) 0.07
Disc	0.62 (.04) 0.00	0.82 (.04) 0.01	1.19 (.06) 0.97	0.96 (.05) 0.38	0.99 (.04) 0.49	0.52 (.04) 0.00	0.62 (.04) 0.00
Hamming	0.65 (.04) 0.00	0.83 (.05) 0.01	1.22 (.07) 0.97	0.96 (.06) 0.40	0.98 (.04) 0.47	0.51 (.04) 0.00	0.60 (.04) 0.00
Temp	0.30 (.04) 0.00	0.50 (.05) 0.00	0.86 (.10) 0.25	0.55 (.07) 0.00	0.75 (.05) 0.00	0.21 (.03) 0.00	0.30 (.05) 0.00
Nodf	0.62 (.04) 0.00	0.78 (.04) 0.00	1.12 (.04) 0.91	0.81 (.05) 0.02	0.93 (.01) 0.07	0.54 (.04) 0.00	0.61 (.04) 0.00
h = 0.50							
N0	0.54 (.05) 0.00	0.67 (.06) 0.00	0.98 (.09) 0.45	0.78 (.07) 0.06	0.71 (.06) 0.00	0.42 (.04) 0.00	0.50 (.05) 0.00
N1	0.98 (.03) 0.40	1.09 (.03) 0.93	1.39 (.08) 1.00	1.26 (.06) 0.98	1.08 (.04) 0.92	0.89 (.05) 0.07	0.92 (.05) 0.24
Disc	0.75 (.05) 0.00	0.97 (.05) 0.40	1.36 (.06) 0.99	1.12 (.05) 0.88	1.03 (.04) 0.78	0.66 (.05) 0.00	0.75 (.05) 0.00
Hamming	0.81 (.06) 0.00	1.00 (.05) 0.57	1.43 (.07) 0.99	1.17 (.06) 0.92	1.06 (.04) 0.86	0.70 (.06) 0.00	0.78 (.06) 0.02
Temp	0.40 (.07) 0.00	0.57 (.07) 0.00	0.88 (.13) 0.31	0.63 (.10) 0.03	0.71 (.07) 0.00	0.31 (.07) 0.00	0.42 (.09) 0.00
Nodf	0.76 (.04) 0.00	0.90 (.03) 0.05	1.21 (.04) 0.99	0.98 (.04) 0.45	0.95 (.01) 0.08	0.69 (.04) 0.00	0.75 (.04) 0.00
h = 0.70							
N0	0.64 (.06) 0.00	0.83 (.08) 0.11	1.17 (.13) 0.82	1.00 (.10) 0.51	0.79 (.07) 0.03	0.52 (.06) 0.00	0.62 (.07) 0.00
N1	1.00 (.02) 0.58	1.06 (.03) 0.91	1.25 (.09) 0.99	1.17 (.07) 0.89	1.04 (.03) 0.85	0.95 (.05) 0.25	0.96 (.05) 0.43
Disc	0.87 (.06) 0.07	1.07 (.05) 0.82	1.41 (.07) 0.99	1.21 (.06) 0.96	1.05 (.04) 0.82	0.81 (.06) 0.00	0.89 (.06) 0.19
Hamming	0.91 (.05) 0.14	1.07 (.05) 0.84	1.41 (.09) 0.99	1.21 (.06) 0.94	1.04 (.04) 0.80	0.84 (.06) 0.02	0.90 (.06) 0.25
Temp	0.54 (.11) 0.00	0.61 (.10) 0.02	0.80 (.14) 0.22	0.62 (.12) 0.05	0.70 (.10) 0.02	0.47 (.13) 0.00	0.57 (.15) 0.04
Nodf	0.88 (.03) 0.00	0.99 (.01) 0.53	1.22 (.05) 0.99	1.08 (.01) 0.91	0.97 (.01) 0.24	0.84 (.03) 0.00	0.88 (.03) 0.01

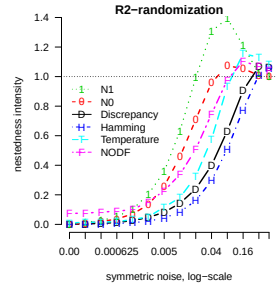
Table 5.3: Rocky Mountain data under asymmetric noise. Randomization results for the data under various levels h of asymmetric noise (1-to-0 errors). We generated 100 noisy matrices from the original data, and randomized each of them 100 times. The tables shows the associated nestedness intensity, its standard deviation (std), and p-value.



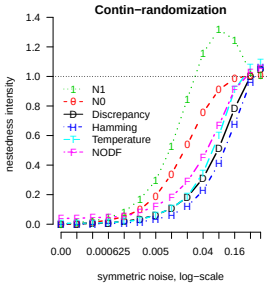
(a) `R0`



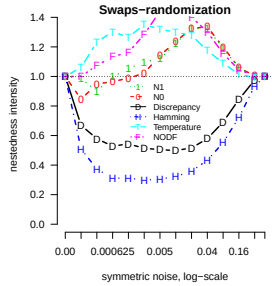
(b) `R1`



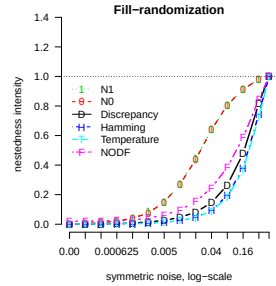
(c) `R2`



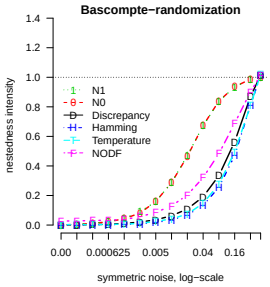
(d) `Contin`



(e) `Swaps`



(f) `Fill`



(g) `Bascompte`

Figure 5.3: Results on synthetic data with symmetric noise (both 0-to-1 and 1-to-0 errors) and different null models. Displayed on the x-axis is the level of symmetric noise ℓ; displayed on the y-axis is nestedness intensity. The plotted values are averages over 10,000 values (100 noisy matrices and 100 randomizations for each).

As the level of symmetric noise increases, the null models R1, Contin, and R2 begin to suggest weak anti-nestedness with many measures. Furthermore, both measures N0 and N1 are prone to Type II errors: the data is seen as non-nested already at low noise levels(> 0.04), which makes these measures unreliable.

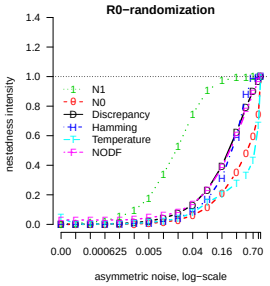
The most distinctive behavior is that of the Swaps model, as seen in Figure 5.3(e). This model has become increasingly popular in nestedness analyses during the past years. The original fully nested matrix is misclassified as non-nested for all measures (nestedness intensity 1.0) because Swaps cannot randomize fully nested matrices. Still, adding low levels of noise helps Hamming and Discrepancy to reveal underlying nestedness to some extent under Swaps. For the popular Temperature and Nodf measures, however, the results incorrectly suggest anti-nestedness on almost all noise levels. The behavior of measures N0 and N1 is erratic, indicating that they should not be used to analyze nestedness when noise is present.

Hamming is more lenient under synthetic noise than other measures. This behavior is indeed desirable, since the underlying original data is known to be fully nested.

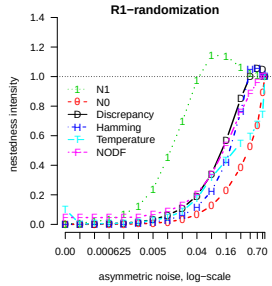
Figures 5.4(a–g) show the results for asymmetric noise (1-to-0 errors) on synthetic matrices that are nested. Many observations from the experiment on symmetric noise hold here, too: similarity of null models Fill, Bascompte, and R0; counter-intuitive results for the Swaps model; the N1 measure is prone to Type II error; and similarity of Hamming and Discrepancy. The main difference is that Nodf follows the behavior of Hamming and Discrepancy fairly closely.

In the experiment with asymmetric noise N0 and Temperature are the most lenient measures. This is expected from N0, considering the way it counts 0s. Low levels of noise keep Temperature close to Hamming and Discrepancy, but as the level of noise increases, Temperature becomes more lenient for nestedness, even overly so. In Figures 5.4(b,d) we observe that Temperature considers the datasets nested even at asymmetric noise level 0.70 when coupled with the most reasonable null models. Thus Temperature is prone to Type I errors, just like in the Rocky Mountain experiment.

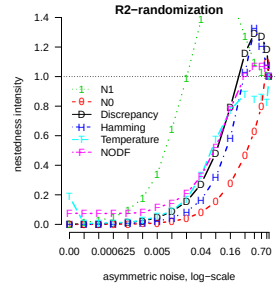
Again in Figure 5.4(e) we see the unexpected results for Temperature under the Swaps model. The underlying data, despite noise, is strongly nested, which is captured by N0, Hamming and Discrepancy. The overall graphic nonetheless shows how unreliable the Swaps model is for nestedness analyses with respect to noise. This compromises the use of Swaps, since practically all real-world datasets contain noise.



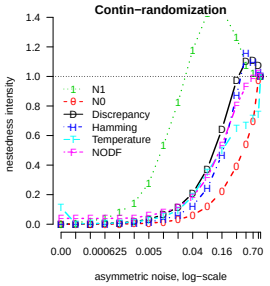
(a) `R0`



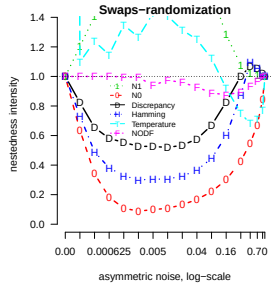
(b) `R1`



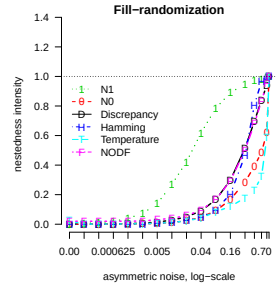
(c) `R2`



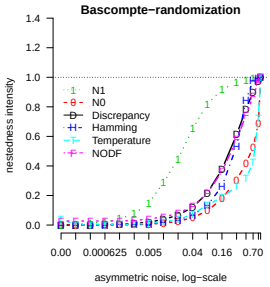
(d) `Contin`



(e) `Swaps`



(f) `Fill`



(g) `Bascompte`

Figure 5.4: Results on synthetic data with asymmetric noise (1-to-0 errors) and different null models. Displayed on the x-axis is the level of asymmetric noise h; displayed on the y-axis is nestedness intensity. The plotted values are averages over 10,000 values (100 noisy matrices and 100 randomizations for each).

5.4 Discussion

There is no differentiation between species and sites in the concept of nestedness: a fully nested matrix remains nested even if the matrix is transposed. Thus, a nestedness measure should not be affected by transposition or exchanging the roles of the rows and columns in the matrix. Of all measures presented here, `Temperature`, `Hamming`, and `Nodf` satisfy this desirable property. Likewise, null models should be ignorant of the nature of rows and columns of the matrix. Of the models described here, `Fill`, `Contin`, `Swaps`, and `Bascompte` indeed have this property. This implies that the nestedness methods that are transposable are also applicable to both biogeographical and interaction data. When analyzing specific datasets, we can develop tailored null models for detecting nested patterns that are generated by specific processes, or to account for different properties of rows and columns [UANG09]. This is likely to improve statistical inference, but requires additional data and experts to develop the methods, and is not applicable to other domains.

What is a nested pattern also depends on the application. No single null model is able to address all these aspects of nestedness. A null model should reflect what we expect to see in datasets without nestedness—a deviation from this model (significance) should indicate a notable discovery in the data. The results from earlier studies [UG07a] show that almost any kind of regularity is enough to make non-nested matrices nested in the eyes of many null models. These null models identify nonrandomness instead of nestedness. In order to distinguish between different nonrandom patterns, specifically designed null models are needed.

The results from the Rasch, Rocky Mountain, and synthetic experiments show that the choice of the measure and null model has a strong effect on the conclusions of nestedness analysis. How should we classify the choices of six measures and seven null models? The measures fall roughly into three categories: the measures `N0` and `N1` have less tolerance for noise than `Discrepancy`, `Hamming`, and `Temperature`; between these groups lies `Nodf`. The results agree with earlier claims that `Temperature` may lead to incorrect conclusions about nestedness [FL02, GC06]. The behavior of `N0` and `N1` is also dependent on the type of noise (symmetric or asymmetric noise): `N1` is fairly sensitive to asymmetric noise. Because of their proneness to Type II error, using `N0` and `N1` for nestedness analyses is not recommended. This agrees with an earlier study [UG07b] that found `N0` and `N1` to be generally inferior to `Discrepancy`.

For the null models, the ordering `Fill`, `Bascompte`, `R0`, `R1`, `Contin`, `R2` characterizes the results quite well: `Fill`, `Bascompte`, and `R0` consider

most of the datasets to be nested, `R1` fewer, and so on, while `R2` views only few datasets as nested. This result is understandable, as the set inclusion relation between the collection of random matrices produced by the null models follows approximately the same ordering. To our knowledge, this study is the first to include the `Bascompte` model in a comparison against other null models. The results show that the results from using this model are as questionable as with `Fill`, which has poor performance [UG07b, Got00]. The experimental results indicate that `Fill`, `Bascompte`, `R0` and `R1` are too lenient (prone to Type I error), while `R2` is too conservative (Type II error).

We showed that the null model `Swaps` is counter-intuitive and erratic in nature. It recognizes fully nested matrices as non-nested, and when used together with the `Temperature` measure it may incorrectly assert non-nestedness or even anti-nestedness. This is in contrast with the current popularity of the method [UANG09], and indicates that the meaningfulness of using `Swaps` should be re-evaluated. If the `Swaps` model is used, only noise-tolerant measures, such as `Hamming` and `Discrepancy` should be employed. Then `Swaps` serves as a conservative model, with poor statistical power, as noted earlier [UG07b]. Since `Swaps` has poor statistical power and its behavior is hard to predict, we recommend using a more robust null model instead, such as `Contin`.

The selection of measure and null model still has a lot of impact on the results. The conclusion of the experiments reported here is that `Temperature` is prone to Type I errors at large levels of asymmetric noise. Also, the behavior of the `Swaps` model is erratic. To assure more reliable data analysis on real-world data, noise-tolerant null models should be used instead. After a consideration of desirable properties of co-occurrence methods [Got00], we conclude that of the methods evaluated here, the use of `Contin` as the null model and `Hamming`, `Discrepancy`, or `Nodf` as the nestedness measure produces the most robust results. The first two measures have good statistical power, while `Nodf` could be used as a conservative less powerful measure.

If a data matrix is nested, its rows and columns can be reordered to reveal the pattern, which helps to understand the reasons for nestedness. Apart from the studies involving `Temperature`, there has not been a lot of discussion about the methods for reordering. In many cases the data matrix is simply ordered by its row sums and column sums. Thus, the errors in the data potentially have an effect on ordering, nestedness results, and further conclusions. Instead of using erroneous data as the basis for reordering, the new measure `Hamming` seeks a nested pattern that is closest to the data, and reorders the matrix according to that pattern. In fact, sum-based

reordering can be arbitrarily bad when compared to a reordering that is based on the underlying nested pattern [MT07]. It is still an open question whether changing reordering methods has an impact on nestedness analysis on real-world data.

There is still disagreement on what counts as a nested pattern. This is exemplified by the `Nodf` measure as it uses a different concept of nestedness than the other measures. A matrix that has 1s in a rectangular shape is considered by `Nodf` even less nested than a fully random matrix, while all other nestedness measures agree on perfect nestedness. In this respect `Nodf` identifies different patterns than the other measures. In general the differences between the concepts can be seen only in the most extreme cases. Nevertheless, a unified view of nestedness is desirable for further development of nestedness methods, or a clear separation of different nested patterns, based on the processes that produce such patterns.

Besides nestedness, there are other well-established types of structure. The concept of anti-nestedness, that is, situations in which the data is less nested than would be assumed by a null model, has stimulated discussion on the ecological processes that lead to such a pattern [ANGL07]. A null model specifically tailored for anti-nestedness is still needed to reliably detect anti-nestedness. There are (roughly) four types of ecological structures [LIPJ$^+$06]: gradients, compartmented, nested, and combined. Modularity patterns have been shown to co-occur with nestedness [FSO$^+$10] in plant-pollinator communities. In segmented nestedness [MT07] a dataset consists of several nested patterns; it remains to be seen whether this type of pattern occurs frequently in ecological data. The theoretical properties of segmented nestedness are considered in the next chapter.

Chapter 6

Segmented nestedness

In this chapter we study the concept of segmented nestedness, in which binary data consists of several disjoint nested patterns. Recall that nestedness describes patterns in which one is expected to witness a hierarchy or a progressive variation of attributes. If the columns of a dataset can be divided into k nested parts, we say that the data has a k-nested pattern. Such a pattern occurs in datasets that have several layers of hierarchical structure, or those that are composites of several smaller data matrices; for example, species may form different nested patterns in different habitats. An example of a k-nested matrix is displayed in Figure 6.1.

In what follows, we develop a theoretical background for k-nestedness and give computational complexity results for finding such structures in binary matrices. By reduction to the Minimum Path Cover problem, we give an algorithm that recognizes the pattern in a noise-free dataset in polynomial time. If noise is present, finding a closest k-nested matrix is an NP-hard problem. We provide heuristic algorithms for studying k-nestedness in noisy data. It turns out that the heuristic that employs both singular value decomposition and k-`means++` methods provides the best results in terms of structure discovery and robustness to noise. Experiments on synthetic data show that the algorithms are noise-tolerant and perform well. In addition, we show how to choose a good number of almost nested blocks k for a data matrix by an MDL model, and how this method gives meaningful results for real-world datasets.

6.1 The concept of segmented nestedness

When Mannila and Terzi [MT07] studied nestedness from the viewpoint of data mining, they also introduced a new concept, segmented nestedness,

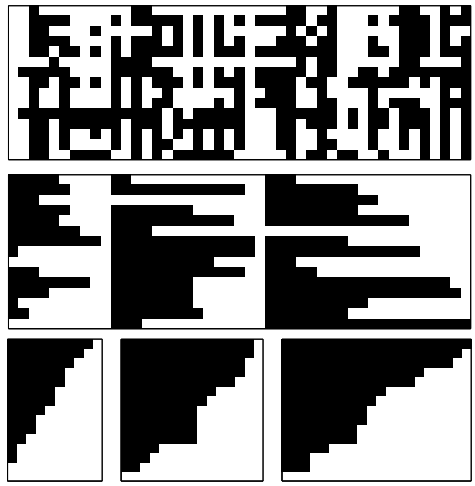


Figure 6.1: A 3-nested binary matrix. We display the matrix (top), with the three nested submatrices grouped into contiguous columns (middle), and each of the three directly nested submatrices (bottom). Observe that the nested components in general cannot be simultaneously presented as directly nested.

which is the main focus in this chapter. The corresponding pattern, k-nestedness, is a reorderable pattern that, unlike other patterns in this thesis, involves partitioning the data.

Segmented nestedness [MT07] is a conceptual extension of nestedness that assumes that a dataset can be partitioned into independently nested parts (Figure 6.1). Consider an $m \times n$ binary matrix A. Given a set of columns S, we denote by $A[S]$ a submatrix of A that is restricted on columns in S.

Definition 6.1 (k-Nested) *Given a binary matrix A and a positive integer k, we say that A is* k-nested *if there exists a partition $\{S_1, S_2, \ldots, S_k\}$ of the columns of A so that submatrix $A[S_j]$ is nested for each $j = 1, 2, \ldots, k$.*

Given a partition $\{S_1, S_2, \ldots, S_k\}$ in Definition 6.1, we say that the partition *splits* A into the nested submatrices $A[S_j]$. We call these nested submatrices *blocks*. From the perspective of set interpretations, a k-nested pattern can be characterized by k chains of subsets.

If a matrix is k-nested, it is also t-nested with all $t > k$. In particular, an $m \times n$ matrix is always n-nested, since a submatrix with only one column is nested (it cannot contain a switch box). All k-nested matrices are also k-consecutive.

A perfectly k-nested dataset is of course rarely witnessed in practice, for example, due to noise or a more rich structure present in the data. In practice one can hope to discover that the dataset is *almost* k-nested, that is, the distance (say, the Hamming distance) to a perfectly k-nested matrix is relatively small. Figure 6.2 illustrates a paleontological dataset partitioned into three blocks that are almost nested.

A related concept of bucket orders [UPGM09] represents a dataset as ordered partitions. This can be seen as a dual problem of k-nestedness, which focuses on partitions of orders. In general the emphasis on matrix reordering has been on single global patterns that fill the whole matrix, but the existence of several independent patterns has received relatively little attention, perhaps partly due to the computational challenges involved in the task. Indeed, already in the case for a single pattern ($k = 1$), finding the

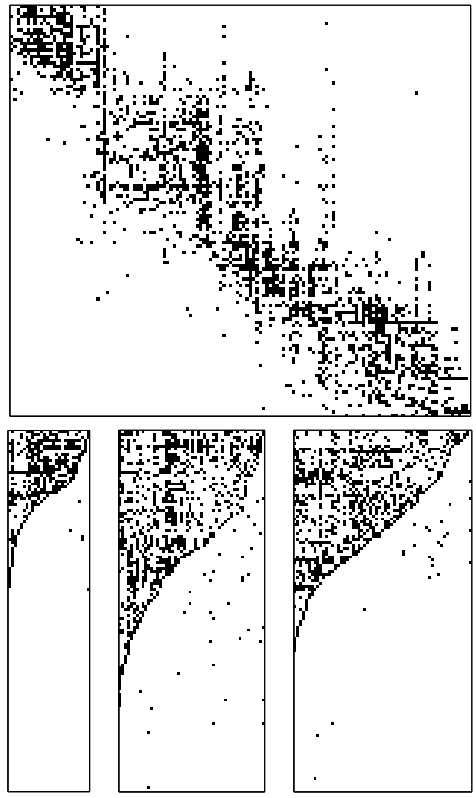


Figure 6.2: A paleontological dataset in its original form (top) and partitioned into 3 almost nested blocks whose rows and columns are permuted individually (bottom). Details of the data are given in Section 6.7.2.

minimum number of flips to reach a target pattern can be computationally challenging, as is the case for nestedness.

Another related problem (in the noise-free case) is the k-Chain Subgraph Cover problem [YCM98], in which the task is to determine if the edge set of a given bipartite graph is the union of the edges in k chain graphs. The difference to k-nestedness is that the partition is for edges, and several paths may be used to cover all 1s on a column.

6.2 Recognition of k-nestedness

In this section we give a polynomial-time algorithm for recognizing a k-nested pattern in a noise-free matrix. The problem is as follows.

Problem 6.2 (k-Nestedness Recognition) *Given a binary matrix A and a positive integer k, find a partition $\{S_1, S_2, \ldots, S_k\}$ that splits A into nested parts, or determine that no such partition exists.*

Given an $m \times n$ binary matrix A as input, we look for the minimum $k \in [n]$ such that A is k-nested. The idea for solving k-Nestedness Recognition in polynomial time is based on a reduction to Minimum Path Cover (Problem 6.5) in directed graphs that are transitive and acyclic. For basic graph-theoretic terminology we refer to [Die10].

For the remainder of this section we assume, without loss of generality, that A has distinct columns. Indeed, we can collapse each class of repeated columns into one representative column, solve for k-nestedness, and expand each representative column in the solution.

We construct an acyclic inclusion graph $D = (V, E)$ for A as in Section 4.6. In short, vertices in V represent the columns of A and a directed edge $(i, j) \in E$ indicates that column C_i is a superset of C_j (when viewing columns as sets). We observe that the graph D is *transitive*, that is, for all $a, b, c \in V$ it holds that if $(a, b) \in E$ and $(b, c) \in E$, then $(a, c) \in E$. Figure 6.3 shows an example matrix and its acyclic inclusion graph. Later in this section, to denote the vertices or edges that belong to a certain structure, we use functions $V(\cdot)$ and $E(\cdot)$.

Lemma 6.3 (Paths and nested submatrices) *A directed path P in D defines a nested submatrix $A[V(P)]$. Conversely, if $A[S]$ is a nested submatrix obtained by restricting A to columns $S \subseteq [n]$, then there exists a directed path P in D with $V(P) = S$.*

Proof. Consider a directed path P in D with vertices $V(P) = \{j_1, j_2, \ldots, j_p\}$ and edges $E(P) = \{(j_t, j_{t+1}) : t = 1, 2, \ldots, p-1\}$. We observe that

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

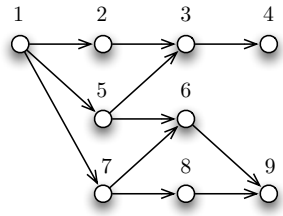


Figure 6.3: A 3-nested matrix with nine columns and its acyclic inclusion graph (with transitive edges omitted). Three paths are needed to cover all the vertices.

$C_{j_1} \supseteq C_{j_2} \supseteq \cdots \supseteq C_{j_p}$ and hence $A[V(P)]$ is nested. Conversely, let $A[S]$ be a nested submatrix of A with $|S| = p$. Because $A[S]$ is nested, there exists a bijection $g\colon [p] \to S$ such that $C_{g(1)} \supseteq C_{g(2)} \supseteq \cdots \supseteq C_{g(p)}$. It follows that $(g(t), g(t+1)) \in E(D)$ for each $t = 1, 2, \ldots, p-1$. In particular, these $p-1$ edges form the edge set of a directed path P with vertex set S in D. □

Lemma 6.4 (Paths and k-nestedness) *The matrix A is k-nested if and only if there exist k directed paths $P_1, P_2, \ldots, P_k$ in D such that $V(D) = \cup_{i=1}^{k} V(P_i)$.*

Proof. It follows immediately from Definition 6.1 and the previous lemma that A is k-nested if and only if there exist k directed paths $P_1, P_2, \ldots, P_k$ in D such that (a) $V(D) = \cup_{i=1}^{k} V(P_i)$ and (b) for all $1 \leq i < j \leq k$ it holds that $V(P_i) \cap V(P_j) = \emptyset$. In particular, it suffices to show that the requirement (b) is redundant. Put otherwise, we must show that if there exists a collection of paths in D that satisfies (a), then there exists a collection of paths in D that satisfies both (a) and (b). It turns out that this follows from the transitivity of D.

Let $P_1, P_2, \ldots, P_k$ be a collection of paths that satisfies the condition (a). For technical convenience, let us allow empty paths; that is, $V(P_i) = \emptyset$ may hold. Now suppose that $V(P_i) \cap V(P_j) \neq \emptyset$ holds for some $1 \leq i \neq j \leq k$. That is, there exists a vertex $v \in V(P_i) \cap V(P_j)$. Let us delete v from P_i as follows. If v either starts or ends the path, we delete v and the only (if any) incident edge with v from P_i. Otherwise v is an internal vertex, with incident edges (u, v) and (v, w) in P_i with some $u, w \in V(D)$. By transitivity of D, also edge (u, w) exists in D. We delete v and its edges from P_i, and add edge (u, w) to the path. After v has been removed, P_i is still a path, but it may be empty. By deleting vertices in this way from the paths, until no such deletions can be made, we obtain a collection of paths $P_1, P_2, \ldots, P_k$ (some of which may be empty) that meet both (a) and (b).

Suppose there are t empty paths. The $(k-t)$ nonempty paths still form a path cover, which implies that A is $(k-t)$-nested, and also k-nested. □

We are ready to restate k-Nestedness Recognition in the language of directed graphs, given an inclusion graph D.

Problem 6.5 (Minimum Path Cover) *Given a directed graph $D = (V, E)$ as input, find directed paths $P_1, P_2, \ldots, P_k$ in D such that $V = \cup_{i=1}^{k} V(P_i)$ and k is minimum.*

Minimum Path Cover in transitive acyclic directed graphs can be transformed [NH79] into a Minimum Flow or a Maximum Matching problem, for which efficient polynomial-time algorithms are known [Gal86]. The idea is to find the maximum independent set of vertices (that is, a maximum *antichain* in the set of columns partially ordered by the subset relation), which, by Dilworth's Theorem [Dil50], has a size equal to that of a minimum path cover. In the context of sets this gives the minimum size chain-partition [Mie05, Ch. 5.2]; in the context of nestedness it is a maximum set of columns where each pair of columns forms a switch box. Since constructing the directed graph D takes $\mathcal{O}(mn^2)$ time, which includes subset checks between all column pairs, we have a polynomial-time algorithm for checking whether a binary matrix is k-nested.

6.3 Distance to k-nestedness

As seen in the paleontological dataset (Figure 6.2), noise leaves no room for a perfect k-nested pattern. We can, however, address the problem of detecting almost k-nested patterns. We next consider the problem of finding a closest k-nested matrix for a given matrix, which turns out to be NP-hard in the general case.

Problem 6.6 (Closest k-Nested) *Given a binary matrix A, a nonnegative weight matrix W, and a positive integer k, find a binary matrix B that is k-nested and minimizes the distance $d_W(A, B)$.*

We will first describe a family of binary matrices where we have control over the distance to a nested matrix. After that we establish NP-hardness on the Closest k-Nested problem by showing that d_{H}-Closest k-Nested is NP-hard, via a reduction from a relaxation of the Vertex Cover problem.

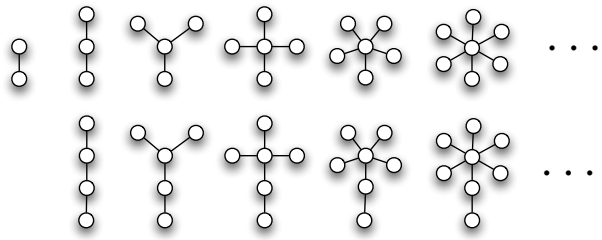


Figure 6.4: Stars (top) and daisies (bottom) that have at least one edge.

Let us start by studying two families of undirected graphs, the *stars* and *daisies* as in Figure 6.4. Throughout this section we assume that all graphs are undirected, loopless (that is, no edge joins a vertex to itself) and simple (that is, no two edges have the same end points).

A *center* of a star or a daisy is a vertex of maximum degree. Note that a center need not be unique if the number of edges is small (at most 1 edge for stars and at most 3 for daisies).

For an undirected graph G with n vertices and s edges, denote by $N(G)$ the *incidence matrix* of G. That is, $N(G)$ is the $n \times s$ binary matrix with vertices as rows and edges as columns such that the entry at row i, column j is 1 if and only if the vertex i and edge j are incident in G. Because of our assumptions (all graphs are simple and loopless), every column of $N(G)$ has exactly two 1s and there are no repeated columns. As an example, we show the incidence matrix of the 3-edge star and the 4-edge daisy:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The following lemmas analyze the Hamming distance of certain incidence matrices to a nested matrix.

Lemma 6.7 (Nested distance from a star) *The incidence matrix of an s-edge star has Hamming distance $s-1$ to a closest nested matrix.*

Proof. We can always permute the rows and columns of the incidence matrix of an s-edge star so that the first row is full of 1s and the rest of the rows $2, \ldots, s+1$ form an $s \times s$ identity matrix.

We need at least $s-1$ flips to reach nestedness: In the identity matrix each pair of columns forms a switch box. If less than $s-1$ flips are made, then at least two columns are left intact, and they still form a switch box.

In fact, $s-1$ flips are sufficient: Flip all 1s in the identity matrix except one. After these $s-1$ flips, switch boxes no longer exist. □

Lemma 6.8 (Nested distance from a daisy) *The incidence matrix of an s-edge daisy has Hamming distance $s-1$ to a closest nested matrix.*

Proof. Consider the $s = 3$ case below.

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

There are two disjoint switch boxes, so one flip is not enough but two flips suffices (the top-right entry and first entry on the second row).

For $s \geq 4$, the incidence matrix of an s-edge daisy consists of the following: an incidence matrix of a $(s-1)$-edge star, a *non-central* column corresponding to the edge not incident to the center vertex, and a row that stands for the vertex not belonging to the star.

We need at least $s-1$ flips to reach nestedness: The $(s-1) \times (s-1)$ identity matrix within the star takes exactly $s-2$ flips. The non-central column and the row corresponding to the center-vertex still form a switch box, so more than $s-2$ flips are needed.

In fact, $s-1$ flips are sufficient: Denote by e the edge that connects the center of the daisy to the vertex with degree 2. Make $s-2$ flips in the identity matrix, namely, flip all 1s but on the column that corresponds to the edge e. This removes all switch boxes within the identity matrix. All remaining switch boxes involve the row that corresponds to the center of the daisy, and we need to flip the only 0 on this row. We observe that for $s-1=3$ we arrive at the following nested matrix, up to permutation of the rows and columns; the bold entries have been flipped.

$$\begin{bmatrix} 1 & 1 & 1 & \mathbf{1} \\ \mathbf{0} & 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

□

Lemma 6.9 (Nested distance from a triangle-free graph) *Let G be a triangle-free loopless simple graph with $s \geq 1$ edges. Then the incidence matrix of G has Hamming distance at least $s-1$ to a closest nested matrix, with equality if and only if G is a daisy or a star.*

Proof. Let G be a triangle-free loopless simple graph with s edges. We proceed by induction on s. The base cases $s = 1, 2$ (1-edge star, 2-edge star, two disjoint edges) are immediate. In particular, observe that a graph with two disjoint edges requires two flips to eliminate the switch boxes. Indeed, up to permutation of the rows and columns, there are three ways F_1, F_2, and F_3 to eliminate the switch boxes with two flips:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \rightsquigarrow \underset{F_1}{\begin{bmatrix} \mathbf{0} & 0 \\ \mathbf{0} & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}, \underset{F_2}{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ \mathbf{1} & 1 \\ \mathbf{1} & 1 \end{bmatrix}}, \underset{F_3}{\begin{bmatrix} \mathbf{0} & 0 \\ 1 & \mathbf{1} \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}.$$

So suppose that $s \geq 3$. Because G is triangle-free it follows that either G is a star (in which case the claim follows by Lemma 6.7) or G must contain two disjoint edges. We thus obtain a partition of the edges of G into two parts, a subgraph D with two disjoint edges and a subgraph H with $s - 2$ edges. By the induction hypothesis we know that at least $s - 3$ flips are required to eliminate all switch boxes in $N(H)$, with equality if and only if H is a daisy or a star. Two more flips are required to eliminate the switch boxes in $N(D)$, so at least $s - 1$ flips are required.

Suppose that exactly $s - 1$ flips are required. We must flip twice in $N(D)$, so exactly one of the $s - 2$ columns in $N(H)$ must remain intact after the $s - 3$ flips. Indeed, if at least two columns remain intact, these two edges define at least one switch box. Thus, $N(H)$ contains one intact column and $s - 3$ columns with exactly one flip on them.

Suppose $N(D)$ has been flipped as F_1. Regardless of the positions of two 1s in the intact column, we see that it is either a copy of an edge in F_1 or it forms a switch box with the second column of F_1. Therefore the flips in F_1 cannot occur. The same holds for F_2.

Suppose $N(D)$ has been flipped as F_3. Up to the permutation of the third and fourth rows, there is exactly one way to insert an intact column without introducing a switch box, namely

$$\left[\begin{array}{cc|c} \mathbf{0} & 0 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right].$$

Now, each of the remaining $s - 3$ columns have one flipped entry; that is, the number of 1s in these columns is one or three. If the second row does

not contain a 1 in each of these columns, we obtain a switch box because of the first column. Thus in what follows we assume that the second row always contains a 1 in the remaining columns.

Suppose first that the 1 in the second row is the result of a flip. Then the column has three 1s and we are in one of the six cases:

$$\left[\begin{array}{ccc|ccccccc} \mathbf{0} & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right].$$

The first case cannot occur because the edge joining the first and third row would complete a triangle with the intact column and the first column. The second case completes a switch box with the intact column. The third case repeats the second column. The last three cases each complete a switch box with the first three columns.

It follows that the 1 in the second row is not the result of a flip. We thus have that G is a daisy since all but the second column share the vertex that corresponds to the second row, and the second column shares a vertex with the intact column. Lemma 6.8 completes the proof. □

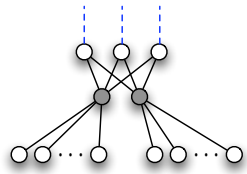
Problem 6.10 (Minimum Daisy/Star Covering) *Given an undirected graph G and a nonnegative integer k as input, determine whether the edges of G can be covered with at most k subgraphs of G, each of which is a daisy or a star.*

We observe that this problem is a relaxation of the classical Vertex Cover problem, which insists on a covering with stars only.

Theorem 6.11 (Covering is NP-complete) Minimum Daisy/Star Covering *is NP-complete when restricted to triangle-free graphs.*

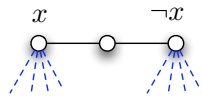
Proof. The problem is in NP because a given covering with daisies/stars can be checked in polynomial time. We reduce from 3SAT to establish completeness. Suppose we are given as input an instance of 3SAT with v variables and c clauses of length 3, such that each variable occurs in at least one clause. We construct a triangle-free graph with $3v+c(5+2(2(v+2c)+1))$ vertices and $2v+c(9+2(2(v+2c)+1))$ edges that has a covering with $v+2c$ daisies/stars if and only if the 3SAT instance is satisfiable.

For each clause in the 3SAT instance, we introduce the following gadget that consists of $5+2(2(v+2c)+1)$ vertices and $9+2(2(v+2c)+1)$ edges; in the bottom row there are two groups of $2(v+2c)+1$ vertices each.



Observe that all the black solid edges (and at most two of the dashed blue edges) in this gadget can be covered with two disjoint daisies/stars. In particular, those two must have their centers at the gray vertices. Indeed, if we do not use two daisies/stars centered at the gray vertices, we require at least $v + 2c + 1$ daisies/stars to cover the black edges.

For each variable x in the 3SAT instance, introduce the following variable gadget that consists of 3 vertices and 2 edges (dashed blue edges not included).



Join each clause gadget via the dashed blue edges to the 3 vertices in the variable gadgets that correspond to the 3 literals in the clause.

Observe that if we use $2c$ daisies/stars to cover the clause gadgets (which we must do if we want to use at most $v + 2c$ daisies/stars), then at least v daisies/stars are required to cover the variable gadgets, one for each gadget. Without loss of generality we may assume that such a daisy/star is a daisy and is centered at one of the literal vertices (x or $\neg x$) and covers all the dashed blue edges incident to the vertex. Here we apply our assumption that every variable occurs in at least one clause; that is, each variable gadget is incident to at least one dashed blue edge.

We now claim that a daisy cover of size $v + 2c$ exists if and only if the 3SAT instance is satisfiable. In the "if" direction, use a satisfying assignment to cover the variable gadgets with daisies so that each satisfied literal is the center of a daisy. Since each clause contains at least one satisfied literal, the corresponding dashed blue edge is covered by the daisy centered at the variable gadget, and the remaining at most two dashed blue edges can be covered with the daisies/stars in the clause gadget. Moreover, the daisies/stars in the cover can be taken to be pairwise disjoint. In the "only if" direction, use the centers of the daisies covering the variable gadgets to reconstruct a satisfying truth assignment. □

From the proof of the previous theorem we observe that the problem remains NP-complete if instead of a cover we ask for a partition of the edges into daisies/stars.

Theorem 6.12 (k-Nestedness and graph covering) *Let G be a simple loopless triangle-free graph with n vertices and s edges. Then G has a partition into k daisies/stars if and only if the minimum Hamming distance from $N(G)$ to a k-nested binary matrix at most $s - k$.*

Proof. Let us abbreviate $N = N(G)$. We start with the "only if" direction. Consider a daisy/star partition $S_1, S_2, \ldots, S_k$ of G. Denote by s_i the number of edges in S_i, $i = 1, 2, \ldots, k$. Consider the $n \times s_i$ submatrix $N[S_i]$ obtained by restricting N to columns S_i. From Lemma 6.9 we have that the minimum Hamming distance from $N[S_i]$ to a nested matrix is exactly $s_i - 1$. It follows that the minimum Hamming distance from N to a k-nested matrix is at most $\sum_{i=1}^{k}(s_i - 1) = s - k$.

We continue with the "if" direction. Let B be a k-nested binary matrix at Hamming distance at most $s - k$ from N, and let $S_1, S_2, \ldots, S_k$ be a partition of the columns that splits B into nested submatrices. Let $s_i = |S_i|$ for $i = 1, 2, \ldots, k$ and observe that $\sum_{i=1}^{k} s_i = s$. Let $d_i = d_{\mathrm{H}}(N[S_i], B[S_i])$. By assumption we have $\sum_{i=1}^{k} d_i \leq s - k$. For each $i = 1, 2, \ldots, k$, the submatrix $N[S_i]$ meets the conditions of Lemma 6.9, and hence $d_i \geq s_i - 1$, with equality if and only if S_i is either a daisy or a star. It follows that $d_i = s_i - 1$ and that S_i is either a daisy or a star. □

It follows from Theorems 6.11 and 6.12 that, given positive integers k, d and a binary matrix A as input, it is an NP-complete problem to decide whether the minimum Hamming distance from A to a k-nested matrix is at most d. Consequently, Closest k-Nested is NP-hard.

6.4 Heuristic algorithms for closest k-nested

Given that Closest k-Nested is NP-hard, it is unlikely that an exact polynomial-time algorithm exists for the problem. This section investigates heuristic algorithms that, given a binary matrix A, a nonnegative weight matrix W, and an integer k as input, find a k-nested matrix $\widehat{B}$ such that the distance $d_W\left(A, \widehat{B}\right)$ is relatively small.

Our focus is on algorithms that attempt to find a partition for the columns of A into k sets such that each induced submatrix $A_1, A_2, \ldots, A_k$ is almost nested. Since no polynomial-time algorithm is known for computing the distance from a given A_i to nestedness, we approximate the distance with the `GreedyNested` method (Algorithm 11). Thus we arrive from each A_i to a nested matrix $\widehat{B}_i$, and obtain an upper bound to $d_W(A_i, B_i)$ by keeping track of the flips. Given a partition of columns, we use $\sum_{i=1}^{k} d_W\left(A_i, \widehat{B}_i\right)$ as an upper bound for the minimum distance from A to a k-nested matrix.

We study three types of algorithms. First, the `Mannila-Terzi` algorithm relies on hierarchical refinement of the partition of columns based on spectral ordering. Second, we introduce two novel techniques, `SVD-`k`-Baseline` and `SVD-`k`-Sim`, which are based on computing a singular value decomposition (SVD) of the input data. Third, k`-Cut` and `AgglomerativeClustering` use a weighting on columns-pairs, which is the number of switch boxes generated by having the two columns in the same matrix.

Mannila–Terzi. The first algorithm for Closest k-Nested was proposed by Mannila and Terzi [MT07, `Partition`], in which spectral ordering [vL07] is used on a graph that represents the similarities between columns. The method is hierarchical: it starts with all columns in a single set, and in each step it splits one set into two and continues until the partition has k sets. We employ the variant with the inclusion similarity measure.

In its original form, the `Mannila-Terzi` algorithm invokes a greedy method `LocalMoves` afterwards to fine-tune the partition. We did not include this time-consuming method in the experiments, as it provides only minor improvements to the best algorithms.

SVD-k-Baseline. We recall that the *singular value decomposition* (SVD) decomposes a real-valued matrix M into a product of three real matrices $M = U\Sigma V^T$ where Σ is a diagonal matrix and U and V are unitary matrices; that is, $UU^T = VV^T = I$, where I is the identity matrix.

In the `SVD-`k`-Baseline` algorithm, given a binary matrix A and the number of nested blocks k as input, we run SVD on A and choose from V the first k columns that correspond with the k largest singular values in Σ. These vectors form the basis of a k-dimensional Euclidean space. We project the columns of A into this space and run the k`-means++` clustering algorithm [AV07], which gives us a partition of the columns.

Figure 6.5 displays the columns of a 3-nested matrix projected into a Euclidean space. We observe that the SVD apparently separates the nested blocks from each other, which enables one to find a good partition.

SVD-k-Sim. The k`-means++` clustering assumes that the data points follow approximately a mixture of Gaussian distributions, but the chains seen in Figure 6.5 do not have this property. To improve on the results of `SVD-`k`-Baseline`, we run SVD on a derived subset-similarity matrix instead of A directly. This produces more Gaussian-like concentrations of points.

We construct an $n \times n$ *subset-similarity matrix* as follows. Given an $m \times n$ binary matrix, we compare each pair of columns (i, j) and their cor-

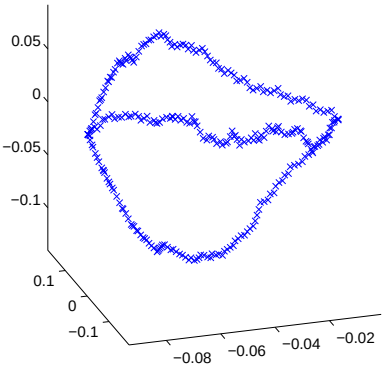


Figure 6.5: The columns of a 3-nested matrix A projected into a 3-dimensional Euclidean space. The basis vectors are the first three columns in the V matrix from the SVD of A. The observed three chains correspond to the nested blocks in A. The chains intersect roughly at two points, which correspond to columns that are full of 1s and full of 0s.

responding sets, C_i and C_j, to see how likely they belong to the same nested structure. Suppose the 1s in the columns were independent and uniformly distributed. Then the number of common 1s between the columns i and j, denoted by X, would be distributed as $X \sim \text{Hypergeom}(m, |C_i|, |C_j|)$, for which expectation $\text{E}(X)$ and variance $\text{Var}(X)$ are known. Denote by $|C_i \cap C_j|$ the actual number of common 1s between the columns i and j. The symmetric *subset-similarity* measure is defined as

$$\text{sim}(i,j) = \frac{1}{1 + \exp\left(-\frac{|C_i \cap C_j| - \text{E}(X)}{\text{Var}(X)}\right)}.$$

Values in $[0, 0.5)$ indicate that similarity is less than randomly expected, whereas values in $(0.5, 1]$ indicate greater similarity. A logistic function was chosen as the basis of the measure because the range of values is $[0, 1]$ and the mapping is smooth.

k-Cut. We recall that a *k-cut* in an undirected graph G is a set of edges whose removal partitions G into k connected components. In the Maximum k-Cut problem, we are given (a) an undirected graph G, (b) a nonnegative integer weight for each edge of G, and (c) a nonnegative integer k; the task is to find a k-cut with the maximum total weight.

To obtain a heuristic solution to Closest k-Nested, we construct a complete undirected graph G whose vertices represent the columns in the

input matrix A. The weight on edge $\{c_1, c_2\}$ is their *conflict-weight*, defined as the number of switch boxes that the two columns form together.

We employ the following greedy algorithm for MAXIMUM k-CUT. Start with k empty bins, and insert the vertices of G one by one into the bins so that each vertex v is placed into the bin that has the smallest cost, where the cost is the sum of conflict-weights between v and the vertices already in the bin. We obtain a partition of the columns of A into k sets from the connected components of the k-cut produced by the algorithm.

Agglomerative clustering. Agglomerative clustering [Ber02] starts by viewing each column as its own cluster, and then merges a pair of clusters with the smallest distance, continuing until k clusters are left. The *cluster-distance* between two clusters is either the minimum, the maximum, or the average of all pairwise conflict-weights between the columns of two clusters. We refer to these methods as `Agglo-Min`, `Agglo-Max`, and `Agglo-Ave`.

Benchmark methods. The following two benchmarks are used to assess the performance of the column-partitioning algorithms on synthetic data where an underlying ground-truth is available.

Given the sizes of the underlying nested blocks in synthetic data, method `Random` returns a partition of the columns with these sizes, selected uniformly at random.

The generative process for k-nested synthetic data includes a partition of the columns, which is used as an underlying ground-truth before adding noise. Method `Original` returns this partition.

6.5 Choosing k with MDL

This section studies the problem of choosing the number k of almost nested blocks when the data is noisy and the correct number k is unknown. This problem is analogous to the problem of choosing the number k of clusters in the context of clustering, where techniques such as BIC, MML and MDL are employed to determine a good value of k [Ber02, HY01]. Here we develop an approach based on MDL for choosing a good number of nested blocks.

The *minimum description length* (MDL) principle states that the best model for the data uses as few bits as possible to represent the data. In other words, MDL chooses the model that best compresses the data by exploiting the regularities in the data.

We introduce a model and an associated encoding scheme for binary data that succinctly encodes a k-nested structure if such a structure is

present in the data. In other words, our encoding scheme for almost k-nested matrices first computes a partition that identifies the almost nested blocks, then constructs nested matrices that are close to these blocks, and finally encodes the k-nested structure together with the differences between the nested matrices and the data. The best value of k is the one that gives the minimum encoded length for the data.

Next we describe the *nestedness scheme* in more detail. To encode an $m \times n$ binary data matrix A using $k = 1, 2, \ldots, n$ nested blocks, we proceed as follows. First, we encode the dimensions m and n of the matrix with Elias δ-coding [Eli75], which is a universal integer-coding scheme. Then, we encode the number of nested blocks k using $\log_2 n$ bits. (We adopt the convention of not rounding up the codelengths to integers.) Next we use `SVD-`k`-Sim` to find a partition of the columns of A into k almost nested blocks, and encode the partition with $n \log_2 k$ bits.

We then repeat the following operations for each almost nested block A_i of A with $i = 1, 2, \ldots, k$. Let n_i be the number of columns in A_i; the value n_i can be recovered from the encoded partition. We run `GreedyNested` on A_i and obtain (a) a matrix Q_i that is directly nested; and (b) row and column permutations for Q_i such that the permuted matrix is close to A_i. We encode the permutations using $\log_2(m!) + \log_2(n_i!)$ bits.

We observe that because Q_i is directly nested, its structure can be described compactly by using a staircase path that identifies the boundary between the 1s and 0s (see Figure 4.1). It takes $\log_2 \binom{m+n_i}{m}$ bits to encode Q_i, stemming from the number of different staircase paths (Section 4.1).

To recover A_i from the encoded Q_i and the row and column permutations, we must also encode the differences between A_i and the permuted Q_i. Observe that we can recover the number of 1s and 0s in Q_i, denoted by p_1 and p_0, respectively, from the existing encoding. We encode the number of differences, that is, the number of 1s in A_i where the permuted Q_i has 0s, denoted by e_1, as well as the number of 0s that occur in A_i but not in the permuted Q_i, denoted by e_0. Finally, the positions of all the differences are encoded using $\log_2 \binom{p_0}{e_1} + \log_2 \binom{p_1}{e_0}$ bits. The total codelength is thus

$$
\begin{aligned}
L_{\text{nest}} = {} & \delta(m) + \delta(n) + \log_2 n + n \log_2 k + \\
& + \sum_{i=1}^{k} \left(\log_2(m!) + \log_2(n_i!) + \log_2 \binom{m+n_i}{m} + \right. \\
& \left. + \log_2(p_0 + 1) + \log_2(p_1 + 1) + \log_2 \binom{p_0}{e_1} + \log_2 \binom{p_1}{e_0} \right).
\end{aligned} \tag{6.1}
$$

As a baseline encoding scheme—that is, without assuming a nested structure—we employ the following *uniform scheme*, which encodes the binary matrix as is. First, encode the dimensions m and n, again with Elias δ-coding. Then, encode the number of 1s in the data, $p = 0, 1, \ldots, mn$, together with the p entries in the matrix out of all $\binom{mn}{p}$ possibilities. The total codelength is

$$L_{\text{unif}} = \delta(m) + \delta(n) + \log_2\big(mn + 1\big) + \log_2 \binom{mn}{p}. \tag{6.2}$$

6.6 Experiments on synthetic data

In the following sections we show how to generate almost k-nested synthetic binary data, and how the generative process gives a ground-truth, against which Closest k-Nested algorithms can be compared. We then proceed to three tests, in which the algorithms need to recognize existing k-nested structure, reconstruct an underlying k-nested structure, and reliably find the correct number of nested blocks k for a data matrix.

6.6.1 Data generation

We generate an almost k-nested dataset A synthetically by first generating a k-nested binary matrix and then adding errors to it. The parameters include the number of rows m, columns n, nested blocks k, and the size for each block $(n_1, n_2, \ldots, n_k)$. Each block $i = 1, 2, \ldots, k$ is an $m \times n_i$ matrix A_i, where row r has its first $\ell_{i,r}$ entries as 1s and $\ell_{i,r}$ is a random number from $\{0, 1, \ldots, n_i\}$.

We then construct a k-nested $m \times n$ matrix B by merging all the blocks together, after which we employ the symmetric and asymmetric error models as presented in Section 2.5. Dataset A is generated by independently flipping the entries in B according to given error probabilities.

6.6.2 Distance test

In this test we measure the Hamming distances produced by the Closest k-Nested algorithms, when the correct k is given but the number of columns in each block is unknown. The smaller the distance, the less the produced matrix deviates from the underlying k-nested structure.

For each level of symmetric and asymmetric noise, we generate 30 samples of 150×150 matrices that are 3-nested with block sizes 25, 50, and 75. We use each algorithm to produce a partition for each dataset, and we

approximate the Hamming distance to a closest 3-nested matrix by using the `GreedyNested` algorithm on each submatrix induced by the partition.

The results are shown in Figure 6.6, where each data point is an average of 30 samples. We observe that `SVD-`k`-Sim` has the lowest distances. Furthermore, at low noise levels the distances are roughly the same that the benchmark method `Original` produces, which suggests `SVD-`k`-Sim` is near-optimal. At higher noise levels the partition from `SVD-`k`-Sim` describes the data even better than `Original`. At the same time, `SVD-`k`-Baseline` is close to `SVD-`k`-Sim`, and `Agglo-Max` is a bit behind. These three algorithms perform better than `Mannila-Terzi`, in which a bad early choice of splitting sets in partition might lead to poor results. The k`-Cut` algorithm has the worst performance, which is roughly the same as what a random partition with correct block sizes produces. The plots suggest that the overall behavior of the algorithms remains comparative regardless of noise model (symmetric or asymmetric). Of the three `Agglo` algorithms presented, we only show the results for `Agglo-Max`, which is the best of the three.

From now on, we include only `SVD-`k`-Sim` in further experiments, as it produces the best distances and is reasonably fast in practice.

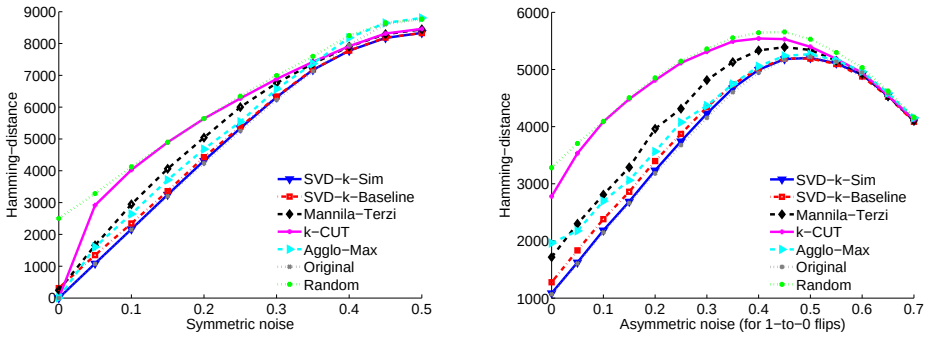


Figure 6.6: Results of the distance test with symmetric noise (left) and asymmetric noise (right). On the x-axes the noise levels are shown: on the left symmetric noise Pr(1-to-0) = Pr(0-to-1); on the right missing data Pr(1-to-0). For the latter, the misclassification parameter is fixed to Pr(0-to-1) = 0.10. On the y-axes are the approximated Hamming distances to closest 3-nested matrices. The plots show how the distances to closest 3-nested matrices change when the noise level increases. The plotted distances are averages of 30 samples.

6.6.3 Classification test

In this test we evaluate how well the partition produced by `SVD-`k`-Sim` matches with the underlying k-nested structure. To be more precise, we measure classification *accuracy*, which is the proportion of columns that a given partition assigns to the same sets as `Original`. We assume the correct k is given, but the number of columns in each block is unknown.

For each noise level, we generate 50 samples of $r \times 150$ matrices, where the number of rows is $r = 6, 25, 100, 400, 1{,}600$, and the matrices are almost 3-nested with block sizes 25, 50, and 75. For each dataset we run the `SVD-`k`-Sim` algorithm and compute the accuracy for its partition. To compare the sets in two partitions correctly, we try all bijections between sets and choose the one that results in maximum accuracy.

The results are displayed in Figure 6.7. The more rows the matrix has, the better accuracy `SVD-`k`-Sim` achieves. The columns that are full of either 1s or 0s can belong to any nested block. Despite this, `SVD-`k`-Sim` has high accuracy, and when the number of rows is low, we see satisfactory results. At high noise levels the underlying k-nested structure has vanished, and `SVD-`k`-Sim` produces partitions that fit better with the noisy data than with the k-nested structure that was used to generate the data.

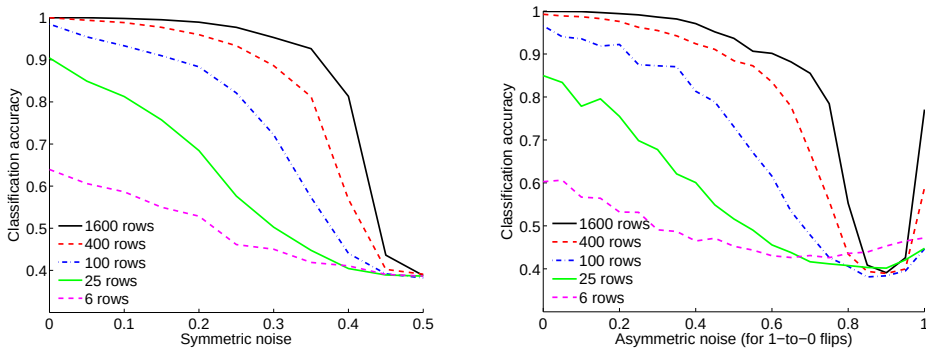


Figure 6.7: Results of the classification accuracy test with symmetric (left) and asymmetric noise (right) for `SVD-`k`-Sim`. On the x-axes are the noise levels: on the left symmetric noise Pr(1-to-0) = Pr(0-to-1); on the right missing data Pr(1-to-0). For the latter, the misclassification parameter is fixed to Pr(0-to-1) = 0.10. On the y-axes classification accuracies are shown. Each line corresponds to a number of rows r, and the average accuracy from 50 samples of $r \times 150$ data matrices are shown for that line. The more rows in the data, the better the accuracy. The expected accuracy of a uniform random partition with correct block sizes is 0.39.

6.6.4 MDL test

In this test we assess the reliability of MDL model selection, that is, test whether MDL is able to retrieve the correct k from synthetic data when we use `SVD-`k`-Sim` to find a partition and `GreedyNested` to find nested blocks.

We generate 30 samples of 150×150 matrices that are k-nested. We repeat the generation process for each true value $k = 0, 1, 2, \ldots, 10$, while keeping the sizes of k nested blocks (almost) equal. With parameter $k = 0$, we generate data that is uniformly random and has 50% fill of 1s. We add symmetric noise to each dataset.

We consider both nestedness and uniform MDL-schemes, and use the `GreedyNested` method with the Hamming distance. From $k = 1, 2, \ldots, 20$, we choose the number of blocks that produces the shortest codelength in the nestedness scheme, or 0 in case the uniform scheme has the shortest codelength.

The results of MDL model selection with symmetric noise are shown in Figure 6.8. The general observation is that the method finds the true k

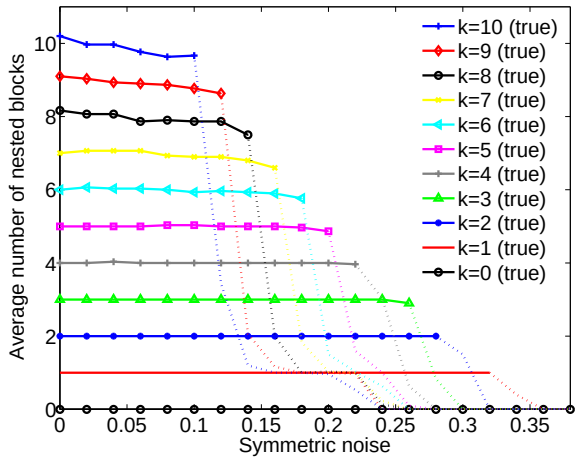


Figure 6.8: Results of the MDL test with symmetric noise. Displayed on the x-axis are the levels of symmetric noise; on the y-axis are the estimated numbers of nested blocks. The legend gives the correct value of k for each line, and plotted values show the number of blocks selected by MDL. Each value is an average from 30 samples. If the uniform encoding scheme is preferred, the number of blocks is zero. Each line is divided into two parts, solid and dotted: the solid part is for noise levels where correct k is recovered reliably, and the dotted part displays the divergent part.

reliably. If each almost nested block consists of only a few columns, or if noise levels are high, finding the correct k is hard. In these situations the MDL method usually favors uniform scheme $k = 0$ instead of choosing an incorrect positive value for k.

6.7 Experiments on real-world data

We next describe two real-world datasets and show that they have an almost k-nested structure, as found by the `SVD-`k`-Sim` method and MDL.

6.7.1 Mammals data

The Mammals dataset[1] consists of presence/absence records of European mammals: for 2,179 locations (columns) of size 40 km $\times$ 40 km we have binary information about 124 mammal species (rows). If a mammal has been observed in a location, the corresponding entry has value 1.

We use `SVD-`k`-Sim` and MDL to analyze whether the locations have a k-nested structure. If the uniform MDL scheme produces the shortest codelength, the data is considered non-nested with $k = 0$. We opted to use the Hamming distances in `GreedyNested`.

Figure 6.9 shows a sample partition of locations when MDL selects $k = 16$. Geographically coherent areas are revealed without using spatial information, which asks for a detailed ecological investigation. For example, Figure 6.10 shows the submatrices that correspond to almost nested blocks in Western France, Bulgaria and Northern Finland. To our knowledge, the observed nested areas roughly agree with common knowledge about the distribution of mammals, for example with cluster analysis [HFEM07].

Because of the randomized nature of the ``k`-means++` algorithm, the value of k selected by MDL is not always the same. From the 50 samples of running `SVD-`k`-Sim` and MDL on the locations, we obtain median 16 and standard deviation 4.7 for the number of blocks. The differences in MDL codelength are typically small with real-world data, and this is the case with values around 16. We conclude that it is convenient to describe the Mammals data as a composite of k almost nested blocks, but k is not a single number but a range of numbers around 16.

On the other hand, the same methods suggest that the species in the Mammals data are 1-nested. The data is displayed in Figure 6.11, with rows and columns permuted to reveal the nested structure.

[1] The preprocessing of the data is described in [HFEM07]. The dataset is available by request from Societas Europaea Mammalogica: `http://www.european-mammals.org/`

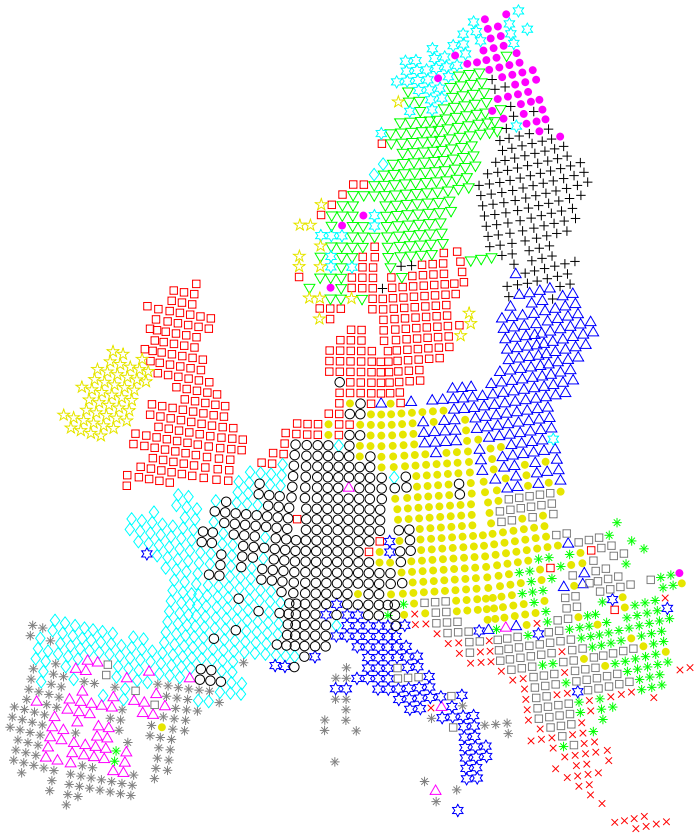


Figure 6.9: Analyzing nestedness in the Mammals data. We display European locations in their 16-nested form, as selected by MDL. All points that share a color and symbol belong to the same almost nested block. We observe that the partition produces geographically coherent areas without any use of spatial information in the algorithms.

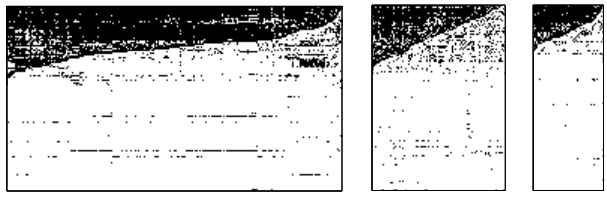


Figure 6.10: Three examples of almost nested blocks in Figure 6.9, from left to right: Western France (225 locations), Bulgaria (89), and Northern Finland (47). The rows are mammals and the columns are locations.

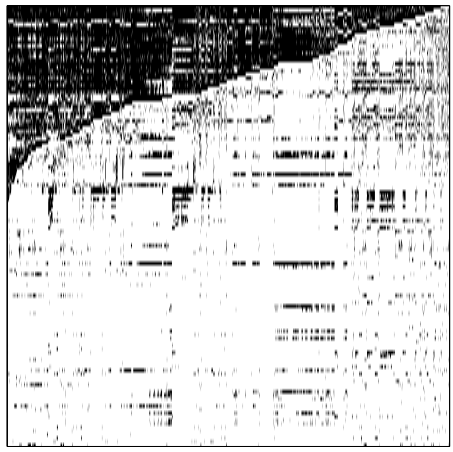


Figure 6.11: Mammals data with its rows and columns permuted to show its nested structure. Rows are mammal species, columns are European locations, and black dots indicate presence.

6.7.2 Paleontological data

The Paleontological dataset [For08][1] has binary information on fossil genera in Europe. There are 124 sites (rows) and 139 genera (columns), and value 1 indicates that a fossil of specific genus has been found on a site, whereas 0 means that it has not been found.

We assume that the data has lots of missing 1s, as is typical for this kind of data, and assign a weight parameter $w = 4$, meaning that the weight on a 1-entry (1-to-0 flip) is four times that of a 0-entry (0-to-1 flip). These weights are used in `GreedyNested` to find nested matrices.

We use `SVD-`k`-Sim` and MDL to evaluate a good choice of k for genera, and to determine whether an assumption of k-nestedness should be discarded (if the uniform scheme has the shortest codelength). The result is that MDL always selects $k = 3$; the dataset and its partition into 3 almost nested blocks are shown in Figure 6.2. This corroborates an earlier claim [MT07] that the genera are 3-nested.

[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.

6.8 Conclusions and further research

While we have shown that it is an NP-complete problem to decide whether a binary matrix has Hamming distance to k-nestedness at most d, our experiments suggest that k-nestedness can be studied on real-world data with heuristic tools, such as the SVD-based algorithms. In this context we would like to highlight two directions for further research.

First, the possibility of rigorous approximation algorithms for k-nestedness should be investigated. Also the computational complexity of d_{H}-Closest Nested is still unknown, which is the same as d_{H}-Closest k-Nested with restriction $k = 1$.

Second, we believe that the SVD-based algorithms would benefit from a more detailed analysis. In particular, we would like to highlight Figure 6.5, where the chains are easy to distinguish visually—why do such chains emerge, and what tools could be used to automatically detect such chain-like patterns in the presence of noise?

We have proposed an MDL-based model selection approach to discover k-nestedness in practice. In addition to sound performance on synthetic data, the tests on real-world data show that MDL is able to discover meaningful structures, such as geographically coherent European areas based on mammal occurrences, and a 3-nested structure in paleontological data. In this context we observe that MDL discovers a 1-nested structure for the mammal species and a 16-nested structure for the locations in the Mammals data. While a further analysis of the results from an ecological perspective is clearly warranted, it is also of theoretical interest to understand how the (almost) nestedness properties of a matrix and its transpose may interact.

A conceptual extension of nestedness that remains to be studied is *laminarity*: a collection of sets is a *laminar family* if any two sets in the collection are either disjoint or one set is a subset of the other.

Chapter 7

Bandedness

The concept of bandedness describes patterns in which we witness a variation of overlapping binary attributes. In this chapter we first give the definition of bandedness and outline potential applications where such a pattern may emerge. A theoretical study on bandedness provides combinatorial properties that can be used as the basis of algorithms. We then proceed to the recognition, distance, and permutation problems, and provide both exact and heuristic algorithms for these problems. Finally, we demonstrate the usefulness of the bandedness concept by applying our methods to synthetic datasets and several applications in life sciences: paleontological dataset, mammal dataset, DNA amplification data, and a dataset on phonological features of Finnish dialects.

7.1 The concept of bandedness

Informally, a binary matrix is banded if both the rows and columns can be permuted so that the nonzero entries exhibit a staircase pattern of overlapping rows. An illustration of a directly banded matrix is in Figure 7.1; also the matrix on the right in (3.1) (page 24) is directly banded.

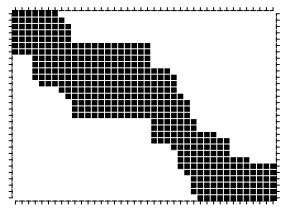


Figure 7.1: An example of a directly banded matrix.

Next, we define three types of bandedness that differ in the way how the pattern can be observed: the banded pattern is either observable as is, or we allow a permutation of the rows to bring out the pattern, or we allow permutations on both the rows and columns.

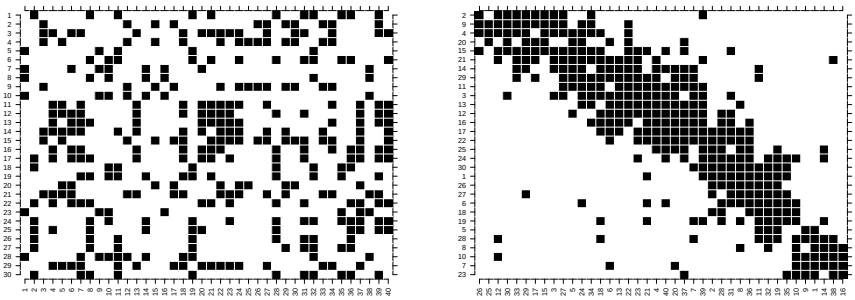
Definition 7.1 (Directly banded) *An $m \times n$ binary matrix is* directly banded *if*

(i) its row vectors are all consecutive (direct C1P),
(ii) $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_m$, and
(iii) $e_1 \leqslant e_2 \leqslant \cdots \leqslant e_m$,
where $\langle s_i, e_i \rangle$ is the consecutive row vector on the ith row.

Definition 7.2 (Columns-banded) *A binary matrix A is* columns-banded *if there exists a permutation of the rows such that the permuted matrix is directly banded.*

Definition 7.3 (Banded) *A binary matrix A is* banded *if there exist permutations of the rows and columns such that the permuted matrix is directly banded.*

Perfectly banded matrices are not expected to arise in real-world (noisy) environments. Therefore we study the problem of determining the distance from the original matrix to bandedness. A simple example is shown in Figure 7.2. In its original form the matrix appears less structured, yet when permuted suitably, it displays a high concentration of 1s in what is an almost banded pattern. Banded submatrices occur when only a subset of attributes belongs to a banded pattern.



(a) (b)

Figure 7.2: An example of a binary matrix in its original form (a), and after permuting its rows and columns to uncover an almost banded structure (b).

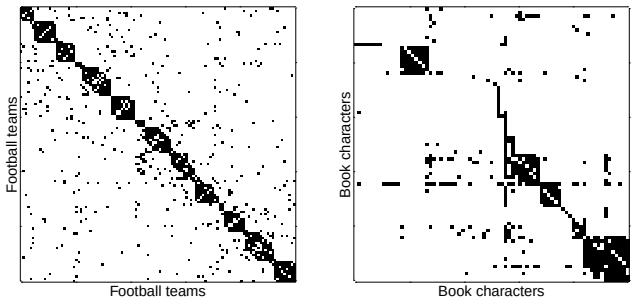
Related work. The concept of banded matrices has its origins in numerical analysis, where the matrix entries can indicate coefficients of variables. From the computational point of view, working with banded matrices is always preferable: the work load involved in performing certain operations, such as multiplication, falls significantly for banded matrices [CM69], often leading to savings in terms of computation time.

Much research has focused on minimizing the bandwidth of a matrix by applying permutations [APÇ04, CM69, Ros68]. Roughly, the *bandwidth* of a binary matrix is the maximum distance of the 1-entries from the main diagonal. Noisy data is still a problem for the bandwidth pattern, as it requires that all 1s must be close to the diagonal. The approach in this chapter allows some 1s in the banded pattern to deviate from the diagonal.

Finding a banded structure is closely related to biclustering problems [MO04] and subspace clustering [MZK+09, PHL04]: the goal is to identify groups of columns and rows that display similar value patterns (for example, similar expression patterns in the case of microarray data). In other words, we interpret the data as a bipartite graph and describe it as a vertex-disjoint union of bicliques (complete bipartite graphs), if possible. Many combinatorial problems have been studied around biclustering. However, in our case the different biclusters (groups of columns and rows with 1s) do not have to be vertex-disjoint.

Our approach can also be seen as a ranking of clusters [BYGL+08]. Rather than simply partitioning a network into clusters, in cluster ranking we also order 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. A difference with those approaches is that bandedness is a parameter-free pattern in the sense that we do not need to fix the number of clusters [CL09].

Applications. From the perspective of data analysis, banded structures occur in many applications. For example, banded patterns can be found in network datasets that consist of overlapping communities without cycles [BKG+05]. Indeed, the permutations of rows and columns can reveal communities of nodes that are strongly connected in overlapping fashion. Consider the Football dataset [GN02]: a 115×115 binary matrix that tells us which US college football teams played against each other in year 2000. An almost banded pattern in Figure 7.3(a) reveals teams that frequently play against each other and are geographically close to one another; the data matrix has Hamming distance 534 to the closest directly banded matrix. Apart from the clusters of teams, the overlapping structure is negligible.



(a) Football data (b) Les Misérables data

Figure 7.3: Almost banded patterns in two network datasets after suitable row and column permutations: Football (a) and Les Misérables (b). These permutations (from a heuristic method) do not establish symmetricity.

Another example comes from the network of characters [Knu93] in the novel Les Misérables by Victor Hugo: an almost banded pattern of this 77×77 matrix displayed in Figure 7.3(b) shows the coappearence patterns in the novel. The Hamming distance of this dataset to direct bandedness is 201. In both datasets the ordering of the rows and columns reveals clusters that are linked in an overlapping fashion. In Les Misérables data, we also observe characters that appear more independently across the chapters.

Not all network datasets are banded, but still they might contain communities. The notion of a community expresses a group of highly interacting nodes, which is more general. The idea of bandedness is that the locality structure of the communities can be mapped in an overlapping fashion. In general, bandedness conveniently describes linearly overlapping clusters.

For another application, consider the physical mapping problem of the human genome [AKNW95]. Genome biologists break the genome into pieces (clones) which by recursive breaking can eventually be 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 [AKNW95]. In this genetic fingerprinting application, rows would correspond to probes and columns to clones

Banded patterns can also exist in species occurrence data, such as in presence/absence data from paleontology. Rows represent sites and columns represent species. A banded structure signifies an overlapping pattern be-

tween a set of species occurring in a spatially correlated set of sites. Or similarly, consider a dataset of dialect words used in several locations or municipalities, represented by a binary matrix. For this linguistic application a banded pattern provides a possible visualization of the spatial distribution of dialects across the municipalities of a country.

7.2 Properties of bandedness

This section studies the definition and combinatorial nature of banded matrices. We also define an intuitive binary relation between the rows which, under a suitable column permutation, will lead to a banded structure. Several results will be exploited later in algorithms.

Consider an $m \times n$ binary matrix A. It follows from the definition that bandedness is a hereditary property: submatrices of a banded matrix are also banded. Also, A is banded if and only if its transpose A^T is banded.

Direct bandedness has several alternative characterizations. For example, consider in a matrix grid two staircase paths that start from the top-left corner, move only down or right, and end up in the bottom-right corner. Two staircase paths define a directly banded matrix that has 1s on entries that fall between these paths. Each directly banded matrix corresponds to such a pair of paths. We get a lower and upper bound for the number of directly banded $m \times n$ matrices by counting how many staircase paths there are. The lower bound $\binom{m+n}{m}$ comes from restricting to one path only, and upper bound $\binom{m+n}{m}^2$ comes from two independent paths. The latter is an upper bound, because for some directly banded matrices there are more than one suitable pair of staircase paths, since the paths may cross.

Also, recall the Definition 2.6 of zero-partitionable matrices. A matrix is directly banded if and only if its 0s can be labeled with R or C so that (a) all entries above and to the right from an R-entry are Rs, and (b) all entries below and to the left from a C-entry are Cs [Wes98].

The adjacency matrices of certain interval graphs have a banded pattern, known as *monotone consecutive arrangement* in graph theory. The details of this connection were already covered in Section 2.4.

We observe that every columns-banded matrix has direct C1P (Definition 3.4), but the matrix is not necessarily directly banded under the current permutation of the rows. Also, each banded matrix has SC1P, but the converse does not hold. For example, consider the following matrices F_1 and F_2: both of them have direct SC1P, but neither is banded.

$$\underset{F_1}{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}, \quad \underset{F_2}{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}}, \quad \underset{F_3}{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}} \tag{7.1}$$

In fact, we can separate the family of banded matrices from both zero-partitionable matrices and SC1P matrices by using characterizations by forbidden submatrices.

Theorem 7.4 (Bandedness and zero-partitionability [LSW97]) *If a binary matrix A is zero-partitionable, then A is banded if and only if it does not contain $F_1, F_1^T, F_2, F_2^T, F_3$, or F_3^T as a submatrix, as in* (7.1).

Theorem 7.5 (Bandedness and SC1P [LSW97]) *If a binary matrix A has SC1P, then A is banded if and only if it does not contain F_1, F_1^T, F_2, or F_2^T as a submatrix, as in* (7.1).

Even if a matrix is banded, all column permutations that establish direct C1P will not lead to an overlapping sequence of rows. For an example, the following matrix in (7.2) has direct SC1P but it is not columns-banded (the last column should be placed first).

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{7.2}$$

Next we characterize exactly the relation of columns-banded matrices with C1P matrices. This will be possible via the following binary relation between rows that are not zero vectors.

Definition 7.6 (Proper consecutive inclusion) *Let $\mathbf{r}_i = \langle s_i, e_i \rangle$ and $\mathbf{r}_j = \langle s_j, e_j \rangle$ be two consecutive vectors. We say that $\mathbf{r}_j$ is* properly included *in $\mathbf{r}_i$, denoted by $\mathbf{r}_j \prec \mathbf{r}_i$, if and only if both vectors are nonzero and $s_i < s_j$ and $e_j < e_i$. We write $\mathbf{r}_j \frown \mathbf{r}_i$ to indicate that $\mathbf{r}_j \not\prec \mathbf{r}_i$ and $\mathbf{r}_i \not\prec \mathbf{r}_j$.*

As an example, the third row of the matrix in (7.2) is properly included in the second row. The rows of a directly C1P matrix A form a *Sperner family of consecutive rows* if the row vectors have $\mathbf{r}_i \frown \mathbf{r}_j$ for all rows i and j. The following result follows directly from Definition 7.6.

Lemma 7.7 *For two consecutive nonzero row vectors $\mathbf{r}_i = \langle s_i, e_i \rangle$ and $\mathbf{r}_j = \langle s_j, e_j \rangle$ we have $\mathbf{r}_j \frown \mathbf{r}_i$ if and only if at least one of the following holds: (1) $s_i = s_j$, (2) $e_i = e_j$, (3) $s_i < s_j \wedge e_i < e_j$, or (4) $s_j < s_i \wedge e_j < e_i$.*

Proof. We have

$$\begin{aligned}\mathbf{r}_j \frown \mathbf{r}_i &\Leftrightarrow \mathbf{r}_j \not\prec \mathbf{r}_i \wedge \mathbf{r}_i \not\prec \mathbf{r}_j \\ &\Leftrightarrow \neg(s_i < s_j \wedge e_j < e_i) \wedge \neg(s_j < s_i \wedge e_i < e_j) \\ &\Leftrightarrow (s_j \leqslant s_i \vee e_i \leqslant e_j) \wedge (s_i \leqslant s_j \vee e_j \leqslant e_i) \\ &\Leftrightarrow (s_i = s_j) \vee (e_i = e_j) \vee (s_i < s_j \wedge e_i < e_j) \vee (s_j < s_i \wedge e_j < e_i).\end{aligned}$$

□

This Sperner property on the family of consecutive rows can be seen as a restricted version of the Sperner property[1] on the family of row sets: even if two rows i, j in a directly C1P matrix A have a proper subset relation $R_i \subset R_j$ for their set interpretations, the row j does not necessarily properly contain the row i, that is, $\mathbf{r}_i \not\prec \mathbf{r}_j$ may hold. This depends on the column permutation of the directly C1P matrix A, which, in the end, determines the starting and ending positions of consecutive 1s on the rows. For example, the third row in the matrix in (7.2) would not be properly included in the second row if the last column was placed first. The following statement characterizes exactly this relation.

Theorem 7.8 (Columns-banded and consecutive inclusion) *A binary matrix A is columns-banded if and only if it has direct C1P and for every two rows i and j the row vectors satisfy $\mathbf{r}_i \frown \mathbf{r}_j$.*

Proof. "If": Consider two consecutive row vectors $\mathbf{r}_i = \langle s_i, e_i \rangle$ and $\mathbf{r}_j = \langle s_j, e_j \rangle$ in A. Because $\mathbf{r}_i \frown \mathbf{r}_j$, none of the rows properly contains another. By Lemma 7.7 we can establish a preorder of rows by sorting them in ascending order by their s-values, while resolving ties with the ascending e-values; ties on identical rows can be resolved arbitrarily. We obtain a row permutation under which A is directly banded, as per Definition 7.1.

"Only if": Given that A is columns-banded, we know that A has direct C1P and that there exists a permutation of the rows so that A is directly banded. It follows from Definition 7.1 and Lemma 7.7 that none of the rows in a directly banded matrix properly includes another row. □

The result from Theorem 7.8 can also be expressed in terms of forbidden submatrices. Consider a directly C1P matrix A and its 2×3 submatrix as in (7.3). The columns a, b, and c in the submatrix occur in that order in A (but not necessarily on consecutive indices). We call such matrices *forbidden Sperner submatrices*, regardless of the order and position of the

[1] In the context of sets, a Sperner family is an antichain: a collection of sets such that none of the sets is a subset of another. We depart from the original definition for technical reasons.

rows. If such a submatrix exists on two rows, we say that they have a *Sperner-conflict*.

$$\begin{array}{c} \begin{matrix} a & b & c \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \tag{7.3}$$

Theorem 7.9 (Columns-banded C1P forbidden submatrices) *A binary matrix A is columns-banded if and only if it has direct C1P and none of its submatrices is a forbidden Sperner submatrix* (7.3)*, when the columns in the submatrix follow the same order as in A.*

Proof. By Theorem 7.8, a directly C1P matrix A is columns-banded if and only if we have $\mathbf{r}_i \frown \mathbf{r}_j$ for all row vectors $\mathbf{r}_i = \langle s_i, e_i \rangle$ and $\mathbf{r}_j = \langle s_j, e_j \rangle$. From Lemma 7.7 we see that this is a required and sufficient condition for the following: there are not three column indices a, b, and c (in that order) that together form a forbidden Sperner submatrix (7.3). □

7.3 Recognition of bandedness

In this section we study the problem of recognizing columns-banded and banded matrices. It turns out that both problems can be solved in polynomial time in the size $m \times n$ of the input matrix.

Problem 7.10 (Columns-Bandedness Recognition) *Given a binary matrix A, determine whether it is columns-banded.*

We can solve Problem 7.10 in time $\mathcal{O}(mn + m^2)$ by Theorem 7.8. We first check whether A has direct C1P; if it has, then we check all row pairs, and A is columns-banded if and only if no row properly contains another row, which by Lemma 7.7 takes constant time for each pair of rows.

Problem 7.11 (Bandedness Recognition) *Given a binary matrix A, determine whether it is banded.*

The set of permutations that satisfies direct bandedness is a subset of those satisfying direct C1P. Checking whether a matrix A is banded would be infeasible if we had to go through all the permutations that establish consecutive rows: there might be exponentially many such permutations. We next describe polynomial-time algorithms for Problem 7.11.

For the first algorithm, we construct a *band-conflict matrix* $\widetilde{A}$ by inserting extra rows in A and then check whether $\widetilde{A}$ has C1P. More precisely, for

every pair of rows i, j in A that have $R_j \subset R_i$ (in the set interpretation), we insert a binary row that has set interpretation $R_i \setminus R_j$. We notice that given a column permutation, this inserted row is consecutive if and only if rows i, j do not properly include each other.

Theorem 7.12 (Band recognition through C1P) *Let A be a binary matrix and $\widetilde{A}$ its band-conflict matrix. Then A is banded if and only if $\widetilde{A}$ has C1P.*

Proof. "If": Since $\widetilde{A}$ has C1P, we may assume its column permutation establishes direct C1P. Because A is a submatrix of $\widetilde{A}$ and direct C1P is a hereditary property, also A has direct C1P. All the inserted rows in $\widetilde{A}$ are consecutive, too, which means that none of the rows in A is properly contained in another row. By Theorem 7.8 the matrix A is columns-banded.

"Only if": Assume without loss of generality that A has already been permuted to be directly banded (these permutations always exist and permutations do not affect inserted rows). Since a directly banded matrix always has direct C1P, we need to show that also the inserted row vectors in $\widetilde{A}$ are consecutive under this column permutation. For each inserted row, there are rows i, j in A that have $R_j \subset R_i$. Let $\mathbf{r}_i = \langle s_i, e_i \rangle$ and $\mathbf{r}_j = \langle s_j, e_j \rangle$ be the row vectors. Because A is directly banded and $R_j \subset R_i$, we have either $s_i = s_j \wedge e_j \leqslant e_i$ or $e_i = e_j \wedge s_i \leqslant s_j$. The inserted row, $R_i \setminus R_j$, is therefore either $\langle e_j, e_i \rangle$ or $\langle s_i, s_j \rangle$, which are both consecutive. Since this holds for all inserted rows, matrix $\widetilde{A}$ has direct C1P under the given column permutation. □

The `TestBanded` method in Algorithm 12 uses Theorem 7.12 to recognize banded matrices. It relies on a subroutine `TestC1P` that tests whether a matrix has C1P, which can be done in linear time in the size of the matrix (Section 3.2). The number of inserted rows in the band-conflict matrix $\widetilde{A}$ is at most $m(m-1)/2$, which gives time complexity $\mathcal{O}(m^2 n)$ for `TestBanded`.

Another algorithm comes from Theorem 7.5. We first check if a matrix A has SC1P by testing C1P on both A and A^T. Then A is banded if and only if it does not contain any forbidden submatrices mentioned in Theorem 7.5, which all have size 3×4 or 4×3. This algorithm has time complexity $\mathcal{O}(m^3 n^4 + m^4 n^3)$. Lin and West [LW95] present a series of improvements that lead to a time complexity $\mathcal{O}((m+n)^2)$. They also establish a result [LSW97] where the construction of Theorem 7.5 gives a linear-time $\mathcal{O}(mn)$ recognition algorithm for bandedness (under the name *monotone consecutive arrangement*). There is no pseudocode for either of these algorithms.

Algorithm 12 `TestBanded`

Input: $m \times n$ binary matrix A
Output: is A banded?

1: $\widetilde{A} \leftarrow A$
2: **for all** row pairs $(i, j) \in [m] \times [m]$ such that $R_j \subset R_i$ **do**
3: Define $\mathbf{r}$ as $r_k = \begin{cases} 1 & \text{if } k \in R_i \text{ and } k \notin R_j, \\ 0 & \text{otherwise,} \end{cases}$ for all $k = 1, 2, \ldots, n$.
4: Insert row vector $\mathbf{r}$ in $\widetilde{A}$ // extra row
5: **end for**
6: **return** `TestC1P`$(\widetilde{A})$ // "yes" if and only if $\widetilde{A}$ has C1P

7.4 Distance to direct bandedness

In this section we study the problem of finding a closest directly banded matrix for a given matrix. We will give a polynomial-time algorithm for the problem.

Problem 7.13 (Closest Directly Banded**)** *Given a binary matrix A and a nonnegative weight matrix W, find a binary matrix B that is directly banded and minimizes the distance $d_W(A, B)$.*

Given an $m \times n$ matrix, Algorithm 13 solves Closest Directly Banded in time $\mathcal{O}(mn^2)$ by using dynamic programming much like in Algorithm 8. Again, it is convenient to split the weight matrix into two parts $W = U + V$, where U contains the weights of 1-entries and V those of 0-entries. Denote by A^r the submatrix of A that contains the rows $1, 2, \ldots, r$.

The algorithm proceeds row by row and keeps track of the distance from A^r to a closest directly banded matrix by computing a 3-dimensional array C. Each entry $C(r, \langle s, e \rangle)$ represents the minimum distance from A^r to a directly banded matrix whose rth row is $\langle s, e \rangle$. The algorithm relies on dynamic programming and uses previously computed distances to compute the distances on subsequent entries. We use consecutive vectors to represent rows in this directly banded matrix. Once the algorithm has computed the last row in C, the smallest distance on the last row is the minimum distance from A to a directly banded matrix.

The algorithm applies the recurrence only to triplets of indices r, s, e that satisfy $1 \leqslant r \leqslant m$ and $1 \leqslant s \leqslant e \leqslant n + 1$, where m is the number of rows and n is the number of columns. To initialize the recurrence, the algorithm assigns consecutive vector $\langle 1, 1 \rangle$ (a row full of 0s) to the first row; hence all 1s on the first row in A contribute to the distance.

Algorithm 13 `FindDirectBanded`

Input: $m \times n$ binary matrix A, and nonnegative weights W

Output: minimum distance from A to a directly banded matrix

1: $U \leftarrow \begin{cases} u_{i,j} = w_{i,j}, & \text{if } a_{i,j} = 1 \\ u_{i,j} = 0, & \text{if } a_{i,j} = 0 \end{cases}$ $V \leftarrow \begin{cases} v_{i,j} = 0, & \text{if } a_{i,j} = 1 \\ v_{i,j} = w_{i,j}, & \text{if } a_{i,j} = 0 \end{cases}$

2: Compute the 3-dimensional array C using the recurrence:

$$C(1, \langle 1, 1 \rangle) \leftarrow \sum_{j=1}^{n} u_{1,j}$$

$$C(r, \langle s, e \rangle) \leftarrow \min\{\texttt{Ext}(r, \langle s, e \rangle), \texttt{Trun}(r, \langle s, e \rangle), \texttt{Move}(r, \langle s, e \rangle)\}$$
$$\text{for } s = 1, 2, \ldots, n+1 \text{ and } e = s, s+1, \ldots, n+1$$

$$\texttt{Ext}(r, \langle s, e \rangle) = C(r, \langle s, e-1 \rangle) + v_{r,e-1} - u_{r,e-1}$$

$$\texttt{Trun}(r, \langle s, e \rangle) = C(r, \langle s-1, e \rangle) - v_{r,s-1} + u_{r,s-1}$$

$$\texttt{Move}(r, \langle s, e \rangle) = C(r-1, \langle s, e \rangle) + \sum_{j=1}^{s-1} u_{r,j} + \sum_{j=s}^{e-1} v_{r,j} + \sum_{j=e}^{n} u_{r,j}$$

3: **return** minimum value associated with the last row in C

There are three ways to reach a consecutive vector $\langle s, e \rangle$ on a row r: by *extending* a consecutive vector $\langle s, e-1 \rangle$, by *truncating* a consecutive vector $\langle s-1, e \rangle$, or *moving* from the row $r-1$ that is $\langle s, e \rangle$. Together these operators can produce any directly banded matrix. The minimum from these three operators provides the minimum distance to $C(r, \langle s, e \rangle)$. Of these three values we ignore the ones that are not available. We next describe these three operators `Ext`, `Trun`, and `Move` used in the recurrence.

- The value of $\texttt{Ext}(r, \langle s, e \rangle)$ is available if and only if $s < e$. In effect, we add one 1 more to $\langle s, e-1 \rangle$ on the row r, which becomes $\langle s, e \rangle$. The distance from A^r then changes accordingly: we either add or subtract the weight $w_{r,e-1}$, depending on whether $a_{r,e-1} = 0$ or $a_{r,e-1} = 1$.

- The value of $\texttt{Trun}(r, \langle s, e \rangle)$ is available if and only if $s > 1$. We flip the leftmost 1 in $\langle s-1, e \rangle$ into 0, which truncates the sequence of 1s in the vector. The distance from A^r then changes: we either add or subtract the weight $w_{r,s-1}$, depending on whether $a_{r,s-1} = 1$ or $a_{r,s-1} = 0$.

- The value of $\texttt{Move}(r, \langle s, e \rangle)$ is available if and only if $r > 1$. We fix the consecutive vector $\langle s, e \rangle$ on the row $r-1$ and move to the next row r. Initially, the consecutive vector on the row r is identical to that of

the row $r-1$. We need to compute the distance from A^r: it is the distance from A^{r-1} plus the errors, that is, the sum of weights on all entries where $\langle s, e\rangle$ and the row r in A differ.

The minimum distance from A to direct bandedness is then the minimum value among the last-row entries of C: $\min_{1\leqslant s\leqslant e\leqslant n+1}\{C(m, \langle s, e\rangle)\}$. We can also construct a directly banded matrix that is closest to A: trace a sequence of `Ext`, `Trun`, and `Move` used to obtain the minimum distance and deduce the consecutive vectors for every row.

The time complexity for `FindDirectBanded` is $\mathcal{O}(mn^2)$ and comes from computing the array C. Instead of using the weights in U and V directly, we construct cumulative weight matrices on the rows of U and V beforehand. With these cumulative matrices, we can evaluate in constant time all the sums in `FindDirectBanded`. Alternatively, we can implement the algorithm in $\mathcal{O}(n^2)$ space, if we represent C as a 2-dimensional array and we replace the distances for the row $r-1$ by the distances for the row r as we compute them. Also, we need the weights and cumulative weights only for one row at a time. The time requirement remains unaffected, however.

7.5 Distance to columns-bandedness

This section deals with a problem that seeks a closest columns-banded matrix, that is, a directly banded matrix after a permutation of the rows.

Problem 7.14 (CLOSEST COLUMNS-BANDED) *Given a binary matrix A and a nonnegative weight matrix W, find a binary matrix B that is columns-banded and minimizes the distance $d_W(A, B)$.*

To reveal the structure in a banded matrix, we need row and column permutations that bring out direct bandedness. In Problem 7.14 we assume that the columns of A have already been permuted, and all that is needed to reveal its almost directly banded structure is to permute the rows.

We next introduce polynomial-time algorithms for CLOSEST COLUMNS-BANDED and d_{A}-CLOSEST COLUMNS-BANDED. All the algorithms work in two phases. In the first phase we flip entries in the input matrix A so that A has direct C1P. In the second phase we flip entries in A so that the resulting matrix is columns-banded. The first two algorithms are motivated by Theorem 7.8, and their second phase consists of eliminating all Sperner-conflicts that prevent bandedness. In the third algorithm the second phase flips entries in order to eliminate forbidden Sperner submatrices that were introduced in Theorem 7.4.

Only the first method `FindColBandedAug` (Algorithm 14), which solves d_{A}-Closest Columns-Banded, is exact. The following two algorithms, `FindColBandedConf` and `FindColBandedForb`, give upper bounds for Closest Columns-Banded. These two heuristic algorithms do not come with any kind of performance guarantee. In general, it seems that Problem 7.14 would still be NP-hard for a fixed permutation of the columns, although no proof is known.

We obtain a distance from all these algorithms by comparing the input matrix A to the output matrix B that is columns-banded. It is also easy to find out a row permutation that establishes a directly banded pattern: sort the rows $\langle s, e\rangle$ of B in ascending order of s, resolving ties with the ascending order of their e.

7.5.1 Augmentation algorithm

Algorithm 14 solves d_{A}-Closest Columns-Banded (only 0-to-1 flips) in polynomial time. For notational convenience, we assume that all row vectors are nonzero. Indeed, rows that have only 0s affect neither bandedness nor the distance.

In the first phase we convert the input matrix A into a directly C1P matrix. Knowing that only 0-to-1 flips are allowed, it is enough to flip all 0s that occur between two 1s on the same row, as on Lines 2–4. We ob-

Algorithm 14 `FindColBandedAug`

Input: $m \times n$ binary matrix A
Output: B is columns-banded and minimizes the distance $d_{\mathrm{A}}(A, B)$

1: $B \leftarrow A$
2: **for all** $i = 1, 2, \ldots, m$ **do**
3: Flip all 0s falling between 1s on the row i in B
4: **end for**
5: **for all** $i = 1, 2, \ldots, m$ **do**
6: Denote by $\mathbf{r}_i = \langle s_i, e_i\rangle$ the ith row vector of B
7: $\mathcal{C} \leftarrow \{\langle s_j, e_j\rangle \mid \mathbf{r}_i \prec \mathbf{r}_j\}$ // conflicting rows
8: Set $\mathbf{r}_i = \langle x, y\rangle$ from the following options so that $y - x$ is minimum:
9: (A) $x = \min\{s_j \mid \langle s_j, e_j\rangle \in \mathcal{C}\}$ and $y = e_i$
10: (B) $x = s_i$ and $y = \max\{e_j \mid \langle s_j, e_j\rangle \in \mathcal{C}\}$
11: (C) $x = s_j$ and $y = \max\{e_k \mid \langle s_k, e_k\rangle \in \mathcal{C}$ and $s_k < s_j\}$, for every $\langle s_j, e_j\rangle \in \mathcal{C}$ // several combinations of x and y
12: **end for**
13: **return** B

serve that this is the minimum number of flips needed. Because all directly banded matrices have direct C1P, all solutions make at least the same flips. In particular, this conversion still allows an optimal solution in terms of fewest flips.

In the second phase we eliminate all conflicts that prevent bandedness. Lines 5–12 ensure that row vectors are pairwise overlapping, that is $\mathbf{r}_i \frown \mathbf{r}_j$ for all rows $i \neq j$. Since only 0-to-1 flips are allowed, the only way to eliminate conflicts is to extend the row vectors. An *extension* of a consecutive vector $\mathbf{r}_i = \langle s, e \rangle$ means to flip 0s into 1s in the vector and keep it consecutive; in other words, an extended row is $\langle s', e' \rangle$ with $s' \leqslant s \leqslant e \leqslant e'$.

We say that $\mathbf{r}_j$ is a *container* of $\mathbf{r}_i$ if $\mathbf{r}_j$ properly contains $\mathbf{r}_i$, that is $\mathbf{r}_i \prec \mathbf{r}_j$. The algorithm finds out an optimal extension for each row vector $\mathbf{r}_i$. To do so it selects all containers of $\mathbf{r}_i$, tests all the potential extensions of $\mathbf{r}_i$, and chooses the extension that none of the containers properly contains.

An extension of $\mathbf{r}_i$ that will always resolve all Sperner-conflicts for that row can either be a left-hand side extension to the leftmost container (Line 9 (A)); a right-hand side extension to the rightmost container (Line 10 (B)); or, extending $\mathbf{r}_i$ to both left and right-hand sides with a combination of two containers (Line 11 (C)). This requires going through all containers $\mathbf{r}_j$ and pairing s_j with the maximum e_k, where $\mathbf{r}_k$ contains $\mathbf{r}_i$ and $s_k < s_j$. Each (A), (B), and (C) eliminate all conflicts on row i. Eventually, for a row i the algorithm takes the extension that represents the fewest flips.

Resolving the Sperner-conflicts for row i does not change the optimal extension on the other rows $j \neq i$, that is, there is no cascading. To see this, consider a row vector $\mathbf{r}_k$ that is not properly included in $\mathbf{r}_i = \langle s_i, e_i \rangle$ but is included in extended $\mathbf{r}'_i = \langle s'_i, e'_i \rangle$. Let the row vector $\mathbf{r}_j = \langle s_j, e_j \rangle$ be a container from which the extension of $\mathbf{r}_i$ originates: we have either $s_j = s'_i$ or $e_j = e'_i$. Now the row vector $\mathbf{r}_j$ contains $\mathbf{r}_k$ by transitivity, and did that even before $\mathbf{r}_i$ was extended. This implies that when processing row k, its optimal extension is independent of a previous extension on row i.

The `FindColBandedAug` algorithm requires polynomial time to solve d_{A}-CLOSEST COLUMNS-BANDED exactly. The number of extensions is at most the number of rows m, and each extension takes at most time $\mathcal{O}(m^2)$ (especially Line 11 (C)), when we represent consecutive rows by the starting and ending positions of 1s. The complexity of the algorithm is $\mathcal{O}(m^3)$, but a more detailed analysis gives a slightly better time complexity when $n < m$. An easy parallelization is also possible on Lines 5–12, since there is no cascading: we can compute the extensions independently for each row.

As an example, we show how `FindColBandedAug` converts a small matrix into a columns-banded matrix.

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \rightsquigarrow \begin{bmatrix} 1 & \mathbf{1} & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \rightsquigarrow \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & \mathbf{1} \end{bmatrix} \tag{7.4}$$

We use two flips (in bold). The first flip makes the matrix direct C1P; the second flip eliminates the Sperner-conflict between rows 2 and 3. This renders the matrix columns-banded, and in this case it is directly banded without reordering the rows.

7.5.2 Sperner-conflicts algorithm

Next we study an upper bound algorithm for Closest Columns-Banded (Problem 7.14). The basic idea behind the algorithm is simple: first make the input matrix A have direct C1P with minimum cost and then resolve all Sperner-conflicts between rows. Despite the distant similarity, the details are, however, largely different from Algorithm 14. Algorithm 15 produces an upper bound for the problem. Again we may assume that all row vectors are nonzero.

Algorithm 15 `FindColBandedConf`

Input: $m \times n$ binary matrix A, and nonnegative weight matrix W

Output: $\widehat{B}$ is columns-banded and the distance $d_W\left(A, \widehat{B}\right)$ is an upper bound for the minimum distance

1: $(d, \widehat{B}) \leftarrow$ `FindDirectC1P`(A, W)
2: **for all** pairs of rows i, j in $\widehat{B}$ **do**
3: Let $\mathbf{r}$ and $\mathbf{t} = \langle s, e \rangle$ be the ith and jth row vectors in $\widehat{B}$
4: **if** $\mathbf{r} \prec \mathbf{t}$ **then**
5: $\mathbf{c} \leftarrow (c_s, c_{s+1}, \ldots, c_{e-1})$, where $c_k = \begin{cases} +w_{i,k} & \text{if } r_k = 0 \\ -w_{i,k} & \text{if } r_k = 1 \end{cases}$
6: $(sum, a, b) \leftarrow$ `MaximumSubvector`$(\mathbf{c})$ such that $s \leqslant a \leqslant b \leqslant e$
7: Flip entries on the row i in $\widehat{B}$ so that the row remains consecutive and $R_j \setminus R_i = \{a, a+1, \ldots, b-1\}$.
8: **end if**
9: **end for**
10: **return** $\widehat{B}$

In the first phase we make A have direct C1P with minimum cost (Line 1). We recall that given a matrix A, `FindDirectC1P` (Algorithm 4) finds a closest directly C1P matrix in linear time.

In the second phase (Lines 2–9), the matrix already has direct C1P, and the algorithm proceeds by removing the Sperner-conflicts between row

vectors in a pairwise fashion. One useful technical observation comes from Theorem 7.12: a matrix is banded if its band-conflict matrix with inserted rows has C1P. Therefore, to eliminate all possible Sperner-conflicts between the consecutive rows of A, the algorithm simply has to go through all the inserted rows in $\widetilde{A}$ and make them consecutive. When we propagate the changes back to A we end up with a banded matrix. As before, we can make use of the `MaximumSubvector` algorithm on the extra rows of $\widetilde{A}$ to make them consecutive. It only remains to update the rows in A so that they are kept consistent with the changes made over $\widetilde{A}$. The final obtained solution on A will be columns-banded. Basically, this corresponds to selecting the low-weight flips that will eliminate Sperner-conflicts in a pairwise comparison of rows in A.

The time complexity for `FindColBandedConf` is $\mathcal{O}(m^2 n)$. One pass through all the row pairs is enough to make matrix $\widehat{B}$ columns-banded: after resolving a conflict between two rows, they either share a starting or ending point of 1s, or one of them becomes a zero vector. This also holds in the end.

Algorithm `FindColBandedConf` does not, however, give an exact solution for Problem 7.14. The reason resides in the second phase: rows are compared in a pairwise fashion and globally beneficial flips may be missed. As an example of how `FindColBandedConf` finds a columns-banded matrix, consider the following.

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \rightsquigarrow \begin{bmatrix} 1 & \mathbf{1} & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \rightsquigarrow \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & \mathbf{1} \\ 0 & \mathbf{0} & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{7.5}$$

In the first phase, the algorithm flips one (bolded) entry to reach direct C1P. Then under pairwise comparisons on rows, we observe that Sperner-conflicts occur between row pairs $(1, 3)$, $(2, 4)$, and $(3, 4)$. Two flips on rows 2 and 3 remove the conflicts, and the matrix is columns-banded. To make it directly banded, move the last row between the first and second.

We get better upper bounds if we allow flips on either row i or j on Line 7, which also allows truncating the 1s on row j from left or right. Empirical tests (not shown here) suggest that this method converges when all 0s share the same weight and so do all 1s, but a proof is missing. Furthermore, there is no guarantee this scheme works with arbitrary weights.

7.5.3 Forbidden submatrices algorithm

We recall the forbidden Sperner submatrices in (7.3) that separate C1P matrices from those that are columns-banded. We use Theorem 7.9 to develop an algorithm that eliminates forbidden submatrices by flipping entries until there are no forbidden submatrices left and the matrix is banded. Note that in case of bandedness, after we flip any entry in a forbidden submatrix, the resulting submatrix is no longer forbidden.

Again we assume that matrix A already has direct C1P. If not, then we invoke the first phase of Algorithm 15.

Let us study the forbidden Sperner submatrices that occur between two consecutive row vectors $\mathbf{r}_i = \langle s_i, e_i \rangle$ and $\mathbf{r}_j = \langle s_j, e_j \rangle$ in a matrix that has direct C1P. If none of the rows properly contains the other row or either of them is a zero vector, then $\mathbf{r}_i$ and $\mathbf{r}_j$ do not form a forbidden submatrix. Assume the opposite, and let $\mathbf{r}_j \prec \mathbf{r}_i$. To make it easier to count the total number of forbidden submatrices between the rows i and j, we partition the columns into five sets S_1, S_2, S_3, S_4, and S_5 as follows.

$$\begin{aligned}
S_1 &= \{c \mid c < s_i\}, & |S_1| &= s_i - 1 \\
S_2 &= \{c \mid s_i \leqslant c < s_j\}, & |S_2| &= s_j - s_i \\
S_3 &= \{c \mid s_j \leqslant c < e_j\}, & |S_3| &= e_j - s_j \\
S_4 &= \{c \mid e_j \leqslant c < e_i\}, & |S_4| &= e_i - e_j \\
S_5 &= \{c \mid e_i \leqslant c\}, & |S_5| &= n - e_i + 1
\end{aligned} \tag{7.6}$$

For example, S_3 contains the columns in which both rows have 1s, and S_2 contains those columns c where only row i has 1 and c occurs before the columns in S_3. We observe that picking one column from each S_2, S_3, and S_4 forms a forbidden Sperner submatrix on the rows i and j. The number of forbidden submatrices between the two rows is $|S_2| \cdot |S_3| \cdot |S_4|$. In particular, the number of forbidden submatrices between two rows can be computed in constant time, given s_i, s_j, e_i, and e_j.

We concentrate on certain entries in the matrix, called *border entries*. A zero vector does not have border entries. Given a consecutive row vector $\mathbf{r} = \langle s, e \rangle$ that is nonzero, the entries on indices $s - 1$, s, $e - 1$, and e are border entries. A row may have less than four border entries if $s = e - 1$, $s = 1$, or $e - 1 = n$. In other words, the border entries on a row represent the entries that occur next to the two borders of 0s and 1s on that row.

Theorem 7.15 *A binary matrix A is columns-banded if and only if it has direct C1P and none of its border entries belongs to a forbidden Sperner submatrix.*

Proof. By Theorem 7.9 we only need to show that a directly C1P matrix A has forbidden submatrices if and only there exists a border entry in A that belongs to a forbidden submatrix.

"If": If a border entry belongs to a forbidden submatrix, then the submatrix is also in A.

"Only if': Assume that A contains a forbidden submatrix on, say, columns x, y, z. Let the two associated consecutive row vectors be $\langle s_1, e_1 + 1\rangle$ and $\langle s_2, e_2 + 1\rangle$. This situation looks like this:

s_1		x		s_2		y		e_2		z		e_1	
$\cdots$ 0	1 $\cdots$	1	$\cdots$	1	1	$\cdots$	1	$\cdots$	1	1	$\cdots$	1	$\cdots$
$\cdots$ 0	0 $\cdots$	0	$\cdots$	0	1	$\cdots$	1	$\cdots$	1	0	$\cdots$	0	$\cdots$

e_1	
1	0 $\cdots$
0	0 $\cdots$

We observe that if the columns x, y, z form a forbidden submatrix, so do the columns s_1, y, e_1. In particular, border entries on columns s_1 and e_1 belong to this forbidden submatrix. □

This result gives rise to Algorithm 16 that flips border entries until all forbidden submatrices have been eliminated. After each flip, the matrix still has direct C1P, and Theorem 7.9 can be used. In particular the forbidden submatrices are independent of the other rows: when we flip an entry, we can count the exact number of eliminated forbidden submatrices by examining the submatrices of that row alone.

If the consecutive row vectors are stored by the starting and ending positions of 1s, then the subroutine `NumBandForbOnRow` (Algorithm 17), takes time $\mathcal{O}(m)$ on a single row. Since each row has at most four border entries, Algorithm 16 calls the `NumBandForbOnRow` method $\mathcal{O}(m)$ times per flip. We can ensure convergence by never flipping an entry twice, which leads to direct bandedness after at most mn flips. Without the restriction there is no guarantee of convergence, although empirical tests indicate (not shown here) that the algorithm produces a solution when 0s have a common weight and 1s likewise.

An easy extension to the algorithm is to consider multiple flips at once: for example, entries s_i, s_{i+1}, s_{i+2} on row i may be flipped at the same time. This results in a small improvement in the upper bound, but in practice the running time increases significantly.

There is also an algorithm that eliminates different kinds of forbidden submatrices. We know that a C1P matrix is also zero-partitionable, and therefore the matrix is banded if and only if none of its submatrices is one of those listed in Theorem 7.4. The algorithm counts for each entry in how many forbidden submatrices it participates in, and then flips the entry with the best elimination efficiency with respect to the weight of the entry. The time complexity with a simple implementation is dominated by

Algorithm 16 `FindColBandedForb`

Input: $m \times n$ binary matrix A, and positive weight matrix W

Output: $\widehat{B}$ is columns-banded and the distance $d_W\left(A, \widehat{B}\right)$ is an upper bound for the minimum distance

$(d, \widehat{B}) \leftarrow$ `FindConsecutiveMatrix`(A, W)
while forbidden submatrices exist **do**
  $F \leftarrow$ zero matrix of size $m \times n$ // number of eliminated submatrices
  for all $i = 1, 2, \ldots, m$ **do**
    $t \leftarrow$ `NumBandForbOnRow`$(\widehat{B}, i)$ // forb. submatrices on the row i
    for all border entries $\widehat{b}_{i,j}$ on the row i **do**
      flip the entry $\widehat{b}_{i,j}$
      $f_{i,j} \leftarrow t -$ `NumBandForbOnRow`$(\widehat{B}, i)$
      flip the entry $\widehat{b}_{i,j}$ // cancel the latest flip
    end for
  end for
  $(r, c) \leftarrow \operatorname{argmax}_{(i,j)} \; f_{i,j}/w_{i,j}$ // best elimination efficiency
  flip the entry $\widehat{b}_{r,c}$
end while
return $\widehat{B}$

Algorithm 17 `NumBandForbOnRow`

Input: $m \times n$ matrix A that has direct C1P, and row index i

Output: number of forbidden Sperner submatrices on the row i in A

$c \leftarrow 0$ // number of forbidden submatrices on the row i
Let $\mathbf{r}_i = \langle s_i, e_i \rangle$ be the ith row vector in A
if $s_i < e_i$ **then** // is not a zero vector?
  for all nonzero rows $j \neq i$ in A **do**
    Let $\mathbf{r}_j = \langle s_j, e_j \rangle$ be the jth row vector in A
    if $\mathbf{r}_j \prec \mathbf{r}_i$ **then**
      $c \leftarrow c + (s_j - s_i) \cdot (e_j - s_j) \cdot (e_i - e_j)$
    else if $\mathbf{r}_i \prec \mathbf{r}_j$ **then**
      $c \leftarrow c + (s_i - s_j) \cdot (e_i - s_i) \cdot (e_j - e_i)$
    end if
  end for
end if
return c

the number of such forbidden submatrices, which is $\mathcal{O}(\min\{m^3n^4, m^4n^3\})$. This algorithm is probably slow for practical purposes.

7.6 Distance to bandedness

In this section we seek a closest banded matrix for a given matrix. In other words, we are allowed to permute both the rows and columns to reveal an almost banded pattern. We show that the problem is NP-hard.

Problem 7.16 (Closest Banded) *Given a binary matrix A and a non-negative weight matrix W, find a binary matrix B that is banded and minimizes the distance $d_W(A, B)$.*

To illustrate how difficult it is to find a closest banded matrix, consider the following two examples. On the left we use only 0-to-1 flips to make the matrix banded, whereas on the right both types of flips are allowed.

$$\begin{bmatrix}1&0&1&0\\1&1&1&1\\0&1&1&0\end{bmatrix} \rightsquigarrow \begin{bmatrix}1&0&1&0\\1&1&1&1\\0&1&1&\mathbf{1}\end{bmatrix}, \quad \begin{bmatrix}1&0&1&0\\1&1&1&1\\0&1&1&0\\0&1&1&0\end{bmatrix} \rightsquigarrow \begin{bmatrix}1&0&1&0\\1&1&1&\mathbf{0}\\0&1&1&0\\0&1&1&0\end{bmatrix} \tag{7.7}$$

On the left only one flip (in bold) is enough to transform the matrix into a banded matrix: the third column in the matrix can be placed between the first and second to show direct bandedness. On the right we observe the same: direct bandedness appears by flipping one entry (in bold) and reordering the columns as on the left.

Before proving that Closest Banded is NP-hard, recall the definition of incidence matrices from page 14: Given an undirected graph $G = (V, E)$ with $|V| = m$ and $E = n$, we construct an $m \times n$ binary matrix with vertices as rows and edges as columns; entry at the row v and column e is 1 if v and e are incident in G. For an example, see the matrix in (7.8).

The decision version of Closest Banded asks whether there exists a banded matrix such that the distance from the input matrix is at most a given d. Next we establish the NP-completeness of this decision problem.

Theorem 7.17 *The decision version of* Closest Banded *is NP-complete.*

Proof. The problem is in NP because we can recognize banded matrices in polynomial time with the algorithms in Section 7.3. To show that the problem is NP-hard, we use a reduction from the well-known NP-complete problem Hamiltonian Path (Problem 2.5).

In particular, we show that given an undirected graph $G = (V, E)$ with m vertices and n edges, G has a Hamiltonian path if and only if its incidence matrix has deletion distance $n - (m - 1)$ to a closest banded matrix (minimum number of 1-to-0 flips).

"If:" Consider the $m \times n$ incidence matrix A of G. We notice that after making one 1-to-0 flip on a column, the column has only one 1 left, and can be included in any banded submatrix. Therefore the deletion distance from A is the same as the number of columns on which flips are needed. Because A has deletion distance $n - (m - 1)$, in total $m - 1$ columns are left intact in the flipped matrix. Since we know that the flipped matrix is banded and no edge appears twice, the intact $m - 1$ columns must be like $e_1, e_2, \ldots, e_{m-1}$ in matrix (7.8), apart from row and column permutations. Interpreted as edges, those columns define a Hamiltonian path in the graph from which A was constructed. We obtain the actual order of vertices by restricting ourselves to the intact columns (the ones that still have two 1s) and setting as end-vertices the two rows (v_1 and v_m) that have just one 1 on the intact columns. After this, there is a unique way to order the rest of the vertices so that they form a Hamiltonian path.

$$
\begin{array}{c}
 \\
v_1 \\
v_2 \\
v_3 \\
v_4 \\
\vdots \\
v_{m-2} \\
v_{m-1} \\
v_m
\end{array}
\begin{array}{c}
\begin{array}{ccccccc|ccc}
e_1 & e_2 & e_3 & \cdots & e_{m-2} & e_{m-1} & & e_m & \cdots & e_n
\end{array} \\
\left[\begin{array}{ccccccc|ccc}
1 & 0 & 0 & & 0 & 0 & & 1 & & 0 \\
1 & 1 & 0 & & 0 & 0 & & 0 & & 1 \\
0 & 1 & 1 & & 0 & 0 & & 1 & & 0 \\
0 & 0 & 1 & & 0 & 0 & & 0 & & 0 \\
 & & & \ddots & & & & & & \\
0 & 0 & 0 & & 1 & 0 & & 0 & & 0 \\
0 & 0 & 0 & & 1 & 1 & & 0 & & 1 \\
0 & 0 & 0 & & 0 & 1 & & 0 & & 0
\end{array}\right]
\end{array}
\tag{7.8}
$$

"Only if:" Let $v_1, v_2, \ldots, v_m$ and $e_1, e_2, \ldots, e_{m-1}$ be the vertices and edges that appear in a Hamiltonian path (in that order), and let the rest of the edges be $e_m, e_{m+1}, \ldots, e_n$ in arbitrary order. Assuming these permutations, the incidence matrix of G looks like the matrix in (7.8).

We observe that the edges in the path form a banded submatrix. We flip one 1 on each edge that is not included in the path. After these $n-(m-1)$ deletion-flips the resulting matrix is banded. Therefore $n - (m - 1)$ is an upper bound for the deletion distance.

The said distance is also a lower bound. To see this, consider a subset of edges and the induced submatrix S of the incidence matrix A. Because there are no duplicate columns in A, having more than $m - 1$ columns in S leads (after careful examination) to a Tucker's forbidden submatrix

[Tuc72] that prevents consecutive columns. If S is banded, it can contain at most $m-1$ columns. Thus, there are at least $n-(m-1)$ edges that violate bandedness, each of which needs one 1-to-0 flip to make the matrix A banded. □

By the previous result, the decision versions of both Closest Banded and d_{D}-Closest Banded are NP-complete. The time complexity for d_{A} and d_{H} remain open questions, but nonetheless, we expect both of them to be hard. The basis of this assumption arises from Theorem 7.12 and the result that computing the d_{A}-distance on C1P is an NP-complete problem [GJ79, Prob. SR16]. Also a problem related to bandedness, Bandwidth Minimization of binary matrices is NP-complete [Pap76]. Section 2.4 established the connection between bandedness and interval graphs.

7.7 Heuristic algorithms for closest banded

Since Closest Banded (Problem 7.16) is NP-hard, there is no polynomial-time algorithm for the problem unless P=NP. Thus we develop heuristic algorithms for practical use that scale well with the size of an input matrix, although no guarantees can be given for performance.

The first class of algorithms works in two phases: (i) finding a permutation of columns that establishes almost direct C1P, and (ii) flipping entries in the directly C1P matrix until the matrix becomes banded. Given an input matrix, we simply use the heuristic algorithms in Section 3.6 to solve Closest C1P on the matrix, from which we obtain a column permutation that establishes almost direct C1P. Given this permutation for columns, we employ the columns-banded algorithms in Section 7.5 to make the matrix banded, namely, `SpectralOrdering` or `HamiltonianOrdering` is combined with the algorithms `FindColBandedAug`, `FindColBandedConf`, or `FindColBandedForb`.

Using the Jaccard coefficient in heuristic algorithms seems convenient, as it captures the particularities of bandedness where columns should be increasingly overlapping one with the other. Indeed, for identical columns the coefficient is 1 and for non-intersecting columns it is 0.

The second class of algorithms considers both row and column permutations simultaneously, instead of splitting Problem 7.16 into two parts. The algorithms of the second class are likely to lead to algorithms that find smaller distances. We introduce three such algorithms: `Alternating` and `Simulated Annealing` are new algorithms, whereas `Barycentric` is a pre-existing method, now adapted to solve problems on bandedness.

7.7.1 Alternating

The first of the algorithms follows from the observations that matrix A is banded if and only if its transpose A^T is banded, and transposing A does not affect its distance to a closest banded matrix. This means that in practice we can solve the CLOSEST BANDED problem on either A or A^T. This suggests the following alternating approach: Given the current column permutation, we solve the columns-banded Problem 7.14 with any algorithm in Section 7.5 and obtain a good row permutation. Then we transpose the matrix and keep on solving Problem 7.14 on transposed matrices until convergence, or until a certain number of iterations is reached. The pseudocode of this `AlternatingBand` method is shown in Algorithm 18.

Algorithm 18 `AlternatingBand`

Input: $m \times n$ binary matrix A, nonnegative weight matrix W, and number of iterations t

Output: permutation σ for the rows and τ for the columns such that the permuted matrix A is almost directly banded.

1: Let σ be a random permutation for the rows
2: $M \leftarrow A$; $V \leftarrow W$; $d_{\text{best}} \leftarrow \infty$
3: **for all** $i = 1, 2, \ldots, t$ **do**
4: $\quad M \leftarrow M^T$; $V \leftarrow V^T$; $\tau \leftarrow \sigma$ // transpose, rows become columns
5: $\quad \widehat{B} \leftarrow$ `AnyColBandedAlgorithm`$(\tau(M), \tau(V))$ // any such algorithm
6: $\quad$ Sort the rows $\langle s, e \rangle$ of $\widehat{B}$ in nondecreasing order of s, resolving ties with nondecreasing e; let σ be the corresponding row permutation.
7: $\quad d \leftarrow$ `FindDirectBanded`$(\sigma(\tau(M)), \sigma(\tau(V)))$
8: $\quad$ **if** $d < d_{\text{best}}$ **then**
9: $\quad\quad d_{\text{best}} \leftarrow d$ and let $(\sigma_{\text{best}}, \tau_{\text{best}}) \leftarrow \begin{cases} (\sigma, \tau) & \text{if } i \text{ is even} \\ (\tau, \sigma) & \text{if } i \text{ is odd} \end{cases}$ // new best
10: $\quad$ **end if**
11: **end for**
12: **return** $(\sigma_{\text{best}}, \tau_{\text{best}})$

Notice that the alternating strategy just described does not necessarily converge for all matrices. Indeed, it is not possible to guarantee that the distance to the closest banded matrix 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 a pair of permutations for the rows and columns that achieved the minimum distance among the t iterations. Given these permutations, we use `FindDirectBanded` (Algorithm 13) to obtain the minimum distance and a closest directly banded matrix.

For example, the banded structures in Figure 7.3 were found by the `AlternatingBand` algorithm.

7.7.2 Barycentric

Another strategy that transposes the matrix is presented by the `Barycentric` algorithm, used previously for example to draw graphs [STT81], to seriate paleontological data [BK88], and more recently to reorder binary matrices [MS05]. In essence, the `Barycentric` algorithm finds permutations for both rows and columns such that 1s reside close to each other. It is based on a barycenter measure, which is the average position of 1s in a row/column. Given a binary matrix A, let the barycenter of row i be defined as follows:

$$\text{Barycenter}(i) = \frac{\sum_{j=1}^{n} j \cdot a_{i,j}}{\sum_{j=1}^{n} a_{i,j}}.$$

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 A 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 a direct band at any iteration. Indeed, `Barycentric` only orders the rows and columns according to the barycenter measure in an iterative fashion; one of its advantages is its simplicity.

7.7.3 Simulated annealing

The third algorithm for CLOSEST BANDED (Problem 7.16) is *simulated annealing* [KGV83], which is a general stochastic optimization method. It has been used successfully to solve various combinatorial problems.

The origin of simulated annealing comes from physics and metallurgy: when the temperature of metal is decreased, the molecular structure tends to move to a state that has low energy. If this annealing process is performed slowly enough, the lowest-energy state is finally found.

To apply this method in finding banded patterns, we use *states* to reflect the permutations of the rows and columns of the input matrix. The *energy value* of a state is the distance from the permuted matrix to a closest directly banded matrix. We seek permutations of the rows and columns (states) such that the distance from the permuted matrix A to a directly banded matrix (energy) is minimum. The search space includes all permutations of rows and columns.

Algorithm 19 `SimulatedAnnealingBand`

Input: $m \times n$ binary matrix A, nonnegative weight matrix W, number of iterations t, temperature T, and temperature multiplier α

Output: permutation σ for the rows and τ for the columns such that the permuted matrix A is almost directly banded.

1: $(\sigma_{\text{cur}}, \tau_{\text{cur}}) \leftarrow$ random permutations for the rows and columns in A
2: $E_{\text{best}} \leftarrow \infty$; $E_{\text{cur}} \leftarrow \infty$
3: **for all** $i = 1, 2, \ldots, t$ **do**
4: $\quad(\sigma, \tau) \leftarrow$ `Neighbor`$(\sigma_{\text{cur}}, \tau_{\text{cur}})$ // generate a neighbor state
5: $\quad E \leftarrow$ `FindDirectBanded`$(\sigma(\tau(A)), \sigma(\tau(W)))$ // candidate energy
6: $\quad$**if** $E < E_{\text{best}}$ **then** // best solution so far?
7: $\quad\quad E_{\text{best}} \leftarrow E$; $(\sigma_{\text{best}}, \tau_{\text{best}}) \leftarrow (\sigma, \tau)$
8: $\quad$**end if**
9: $\quad$With probability $\min\{1, e^{(E_{\text{cur}} - E)/T}\}$: $E_{\text{cur}} \leftarrow E$; $(\sigma_{\text{cur}}, \tau_{\text{cur}}) \leftarrow (\sigma, \tau)$
10: $\quad T \leftarrow \alpha T$ // decrease temperature
11: **end for**
12: **return** $(\sigma_{\text{best}}, \tau_{\text{best}})$

Algorithm 19 describes the simulated annealing method. The algorithm starts with a random state and a large temperature parameter T, such as 10. The algorithm keeps track of the current state (permutations) 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; a common choice is to multiply the temperature by $\alpha \in [0.95, 1)$. At each iteration, a new candidate state (permutations) is generated stochastically by a function `Neighbor`. The candidate state is then accepted as a new current state with a probability that depends on the difference of their energy values and on the temperature: the lower the temperature, the more the algorithm favors reductions in energy. In our case, the polynomial-time Algorithm 13, `FindDirectBanded`, serves as the exact energy function.

We next define the details for the auxiliary method `Neighbor` for the simulated annealing algorithm. The most important requirement for the function `Neighbor` is that all states (in our case, permutations) in the search space should be accessible by repeating the candidate generation process repeatedly. We consider several candidate generation methods [TMZ99] for finding good permutations and we apply them in our 2-dimensional permutation problem. Given permutations for the rows and columns, all the following methods describe how the permutations of the underlying matrix change. We treat an ordering as a cycle: after the last row the first row follows. We summarize the methods in the following.

- `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 the (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 an interval of rows, then relocate the rows and reverse their order. Do the same for columns.

We do not expect this stochastic method to be as competitive in terms of running time as the other algorithms presented this far for the problem. In practice the simulated annealing strategy comes with a high computational cost, but yields near-optimal results when implemented properly. For this reason we use this strategy as a benchmark method for comparing the different algorithms for bandedness.

7.8 Exact algorithms for closest banded

Next we give an exact MAX-SAT method for Closest Banded (Problem 7.16). As with all exponential-time exact methods, we do not expect this method to be practical with matrices that have more than 20 rows or columns.

Recall the SAT formulae for C1P in Section 3.7.1 and for nestedness in Section 4.5. Given an $m \times n$ binary matrix A, we construct a propositional logic formula in CNF that is satisfiable if and only if A is banded.

- To ensure consecutive rows, we need all the variables and clauses from the C1P formula in Section 3.7.1. There are $mn + n(n-1)$ entry and pairwise order variables; antisymmetry, transitivity, and consecutivity require $n(n-1) + n(n-1)(n-2) + mn(n-1)(n-2)$ clauses.

- Add *band clauses* for each ordered pair of rows (i, j) and each ordered triplet of columns (a, b, c) in A.

  No forbidden Sperner submatrices:

  $$\neg(Z_{a<b} \wedge Z_{b<c} \wedge e_{i,a} \wedge e_{i,b} \wedge e_{i,c} \wedge \neg e_{j,a} \wedge e_{j,b} \wedge \neg e_{j,c})$$

  Same clause in CNF:

  $$(\neg Z_{a<b} \vee \neg Z_{b<c} \vee \neg e_{i,a} \vee \neg e_{i,b} \vee \neg e_{i,c} \vee e_{j,a} \vee \neg e_{j,b} \vee e_{j,c})$$

  The band clauses ensure that, given consecutive rows and a total order on the columns, no forbidden Sperner submatrices exist as in (7.3). Contributes $m(m-1)n(n-1)(n-2)$ clauses.

If a given assignment on this formula satisfies all its antisymmetry, transitivity, consecutivity, and band clauses, the corresponding matrix is banded. We require all these to be satisfied, and allow leaving some entry clauses unsatisfied. Then a partial MAX-SAT solver finds an assignment that corresponds to a banded matrix that is d_{H}-closest from A, and the actual Hamming distance is the number of unsatisfied entry clauses. Moreover, Closest Banded can be solved by a weighted MAX-SAT solver: the entry clauses have weights equal to the weights W of the corresponding entries in A and the other clauses have weights larger than the sum of all weights. Using this formula, the minimum distance is the sum of the weights on the entry clauses that are unsatisfied, when a weighted MAX-SAT solver produces an assignment.

7.9 Banded submatrices

Sometimes only a part of a dataset is banded, but not the whole dataset. Then it makes sense to find submatrices that are (almost) banded.

Problem 7.18 (Maximum-Size Banded Submatrix) *Given a binary matrix A, find in A a submatrix that is banded and maximizes $a + b$, where a is the number of rows in the submatrix and b that of columns.*

Theorem 7.19 Maximum-Size Banded Submatrix *is NP-hard.*

Proof. Let $\mathcal{M}$ be the collection of all banded matrices. We note that $\mathcal{M}$ is nontrivial: upper triangular matrices (page 35) are banded, but zero diagonal matrices of size at least 3×3 are not banded. Collection $\mathcal{M}$ is closed under permutation of rows and columns by definition. It is also closed under deletion of the rows and columns, because bandedness has a

forbidden submatrix characterization. The result follows from Theorem 2.8, because $\mathcal{M}$ contains upper triangular matrices with unbounded ranks. □

As in Section 2.7, we introduce the problem of finding almost banded submatrices that are useful with respect to a utility function f.

Problem 7.20 (Almost Banded Submatrix) *Given a binary matrix A, a nonnegative weight matrix W, and a fixed distance d, find a submatrix S of A that has distance to bandedness at most d and maximizes the utility $f(S)$.*

Solving Problem 7.20 is useful on datasets that contain several independent band structures, and also when noise is too high to identify a banded structure from the complete dataset. In essence, a solution extracts the most relevant banded pattern in a dataset. Removing the rows and columns that appear in a banded submatrix allows us to run the algorithm several times and obtain a distinct submatrix every time.

For finding banded submatrices, we use `FindSubmatrix` (Algorithm 1) that removes rows and/or columns one by one until the distance is within acceptable range. We settle for measures recall, precision, and accuracy, and leave the development of utility functions for bandedness as future work.

7.10 Experiments on synthetic data

In this section we demonstrate that our algorithms for bandedness are fast in practice and tolerate noise in synthetic datasets.

7.10.1 Data generation

Given the number of rows m, that of columns n, and a width parameter $2w$, we describe how to generate a synthetic matrix that is banded. We rely on generating a staircase path by a random walk along the matrix grid of an $m \times n$ zero matrix. Starting at coordinate $(0, 0)$, 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 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 always reaches the final position (m, n), and when it does, the matrix has a clear solid band of maximum thickness $2w$ like shown in Figure 7.1.

Additionally we can introduce noise, as in Section 2.5, by flipping the original values from 0 to 1, or from 1 to 0, according to given probabilities Pr(0-to-1) and Pr(1-to-0).

We generate samples of 50×55 binary matrices, which are used in all synthetic data experiments. The width parameter is $2w = 30$, which yields approximately a 50% fill of 1s. Before we run the experiments on these matrices, we add noise and randomize the permutations on the rows and columns. We prepare a variety of test settings, including both symmetric noise (Pr(0-to-1) = Pr(1-to-0)) and asymmetric noise, and different distance measures.

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

7.10.2 Methods

The algorithms used in the synthetic experiments include the following: `AlternatingBand`, `Barycentric`, and `SimulatedAnnealingBand`, as well as `FindColBandedAug` and `FindColBandedConf` coupled with several similarity measures and column-ordering methods. We obtained the distances for the algorithms by letting the algorithms choose the permutations for rows and columns, and then running `FindDirectBanded` to evaluate the minimum distance to direct bandedness.

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

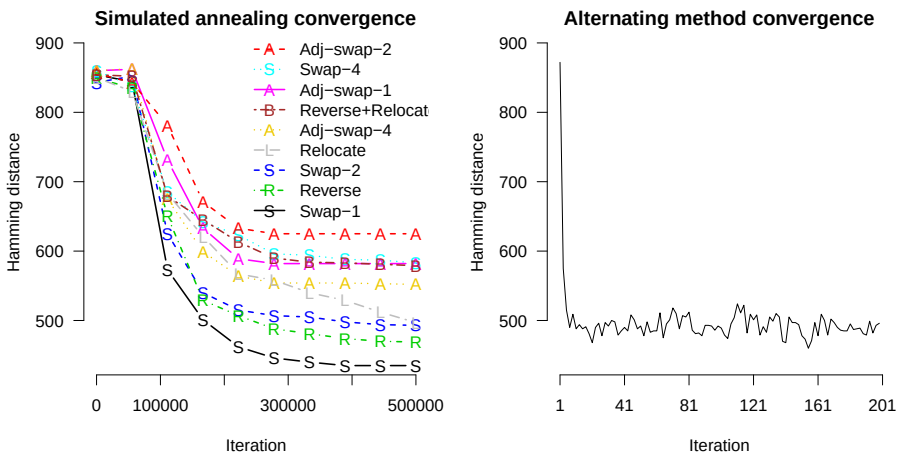
As expected, whenever noise exists, `FindColBandedConf` outperforms `FindColBandedAug`, so we decided to leave `FindColBandedAug` out of the results presented next. Preliminary results from `FindColBandedForb` suggest that it performs slightly better than `FindColBandedConf`, but it is also slower, and was not included in extensive experiments. Also the other submatrix-eliminating algorithm in Section 7.5.3 was tried: the preliminary results from a simple implementation were promising, but the time consumption hindered large-scale experiments.

For `FindColBandedConf`, we sorted the columns via `SpectralOrdering` (Section 3.6.1) with three similarity measures: Pearson correlation, dot product, and Jaccard similarity. We also experimented with finding the column permutations with `HamiltonianOrdering` (Section 3.6.2) that uses

either the Hamming or Jaccard distance measure on columns. For visual clarity, only the best results are shown: `SpectralOrdering` using the dot product and `HamiltonianOrdering` with the Jaccard distance.

For `SimulatedAnnealingBand`, we first compared the neighbor schemes: `Swap-`k and `Adj-swap-`k for $k \in \{1, 2, 4\}$, as well as `Reverse`, `Relocate`, and `Reverse+Relocate`. The results are shown in Figure 7.4(a). Using one million iterations and starting temperature 10 produced results that we consider near-optimal. In terms of convergence speed, `Swap-1` proved to be the fastest, as seen in Figure 7.4(a). This scheme was chosen to be the neighbor scheme for all experiments that include `SimulatedAnnealingBand`.

Because of the 2-dimensional nature of bandedness, the number of iterations that `SimulatedAnnealingBand` needs is larger than other ordinary permutation problems of the same size. We studied the convergence of `SimulatedAnnealingBand` with the `Swap-1` scheme extensively by using different numbers 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 (per sample) and settled for the multiplier $\alpha = 0.9999$ with 100,000 iterations. Other parameter choices that satisfy



(a) `SimulatedAnnealingBand` (b) `AlternatingBand`

Figure 7.4: Displayed on the left (a) is the convergence analysis of `SimulatedAnnealingBand` with different `Neighbor` schemes; on the right (b) is the analysis of `AlternatingBand`. On the y-axes are the minimum Hamming distances to bandedness; on the x-axes are the iterations used.

the time limit were tested but they failed to produce better results. The chosen parameter combination is conservative: the temperature is allowed to drop low enough so that further improvement is unlikely, thus reducing variance. These parameters yield Hamming distances that are, on average, 3% higher than those from one million iterations.

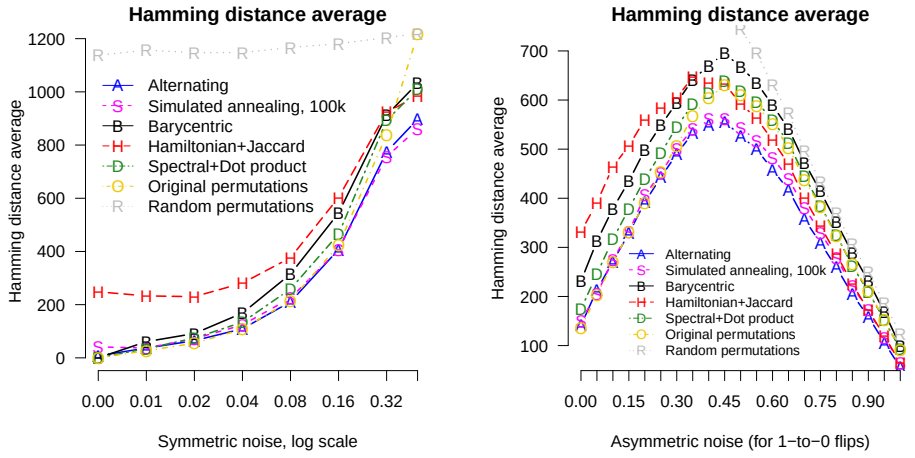
In most cases, both `AlternatingBand` and `Barycentric` converge fast, taking on average less than 30 iterations to find good permutations, regardless of the matrix size and initial permutations. After the first iterations, the `AlternatingBand` algorithm does not strictly converge towards an optimal solution, but starts oscillating, as seen in Figure 7.4(b). Because of this, we always return the best solution among the first 100 iterations.

7.10.3 Results

We performed two types of tests. In the first test we studied the Hamming distances obtained from the algorithms when the banded datasets contained symmetric or asymmetric noise. In the second test we compared the ground-truth permutation to the ones obtained from the algorithms.

We added symmetric noise to a sample of 30 synthetically generated banded matrices and then ran each algorithm to obtain an upper bound for d_{H}-Closest Banded. The average results from the 30 runs for each method are shown in Figure 7.5(a). We observe that both `AlternatingBand` and `SimulatedAnnealingBand` consistently perform as well as the `Original` method, and at high noise levels, they perform even better. This is a good sign since only `Original` has access to the original permutations. The other methods do not beat `Original` until extreme noise levels. Whenever we talk about `Spectral` or `Hamiltonian`, we mean the `FindColBandedConf` method that uses either `SpectralOrdering` or `HamiltonianOrdering` to preorder the columns. We noticed that non-optimal cycle-to-path transformations cause `Hamiltonian` to have a large variance.

In the asymmetric noise case we chose to fix one of the noise probabilities as Pr(0-to-1) = 0.1 and added different levels (Pr(1-to-0)) of asymmetric noise to the input matrices; again, with a sample of 30 matrices in total. To make the comparison reliable, we use the Hamming distance. As in the symmetric noise case, the average result of 30 runs was collected for each method; the results are shown in Figure 7.5(b). Again, the `AlternatingBand` algorithm and `SimulatedAnnealingBand` perform very well, beating `Original` at noise levels Pr(1-to-0) > 0.25. Overall it seems that `AlternatingBand` produces the best results on average, although we noticed its variance to be a bit larger than that of `SimulatedAnnealingBand`. Curiously, the `Barycentric` method performs poorly here: at high noise lev-



(a) Symmetric noise

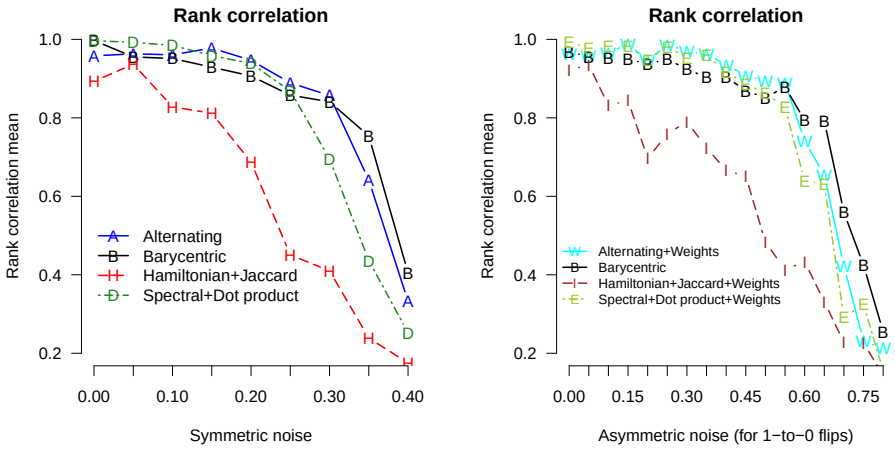
(b) Asymmetric noise

Figure 7.5: Displayed on the left (a) are the average Hamming distances from the synthetic experiment with symmetric noise; on the right (b) the same with asymmetric noise Pr(1-to-0) when Pr(0-to-1) = 0.1. On the y-axes are the Hamming distances (average of 30 samples); on the x-axis symmetric and asymmetric noise probabilities. Algorithms `AlternatingBand` and `SimulatedAnnealingBand` are the best performers.

els, the results barely beat a random permutation. The `Spectral` method performs a bit better, whereas the results for `Hamiltonian` are mixed: mediocre results at low noise levels; good results at high noise levels.

In addition to distance measurements, we want 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 the *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 is the same as the row order in the generation process of noiseless data; the recovered ranking comes from a row ordering retrieved by any of our algorithms. The correlation value 1.0 means perfect agreement between the rankings; values close to 0.0 indicate that there is no correlation between the two.

The results of the rank correlation experiment with symmetric noise are shown in Figure 7.6(a). We see that none of the methods dominates the others: at low noise levels under 0.15, the `Spectral` method is able to recover



(a) Symmetric noise (b) Asymmetric noise

Figure 7.6: Displayed on the left (a) are the results of rank correlation on symmetric noise: correlations between the original row permutation and a permutation from an algorithm. Displayed on the right (b) are the results of rank correlation on asymmetric noise Pr(1-to-0) when Pr(0-to-1) = 0.1. On the y-axes are the mean measurements of Spearman rank correlation from 30 samples; on the x-axes are the symmetric and asymmetric noise probabilities. The weight parameter was passed to the methods with "+Weights".

the original order almost perfectly; from 0.15 to 0.30, the `AlternatingBand` 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 good in recovering the original row order. `Hamiltonian`, however, is able to find the original order only at the lowest noise levels. Occasionally the `AlternatingBand` method converges towards local minimum, as demonstrated by the correlation value below 1.0 with noiseless data.

The results of correlation with asymmetric noise are shown in Figure 7.6(b). To better deal with asymmetric 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 the input matrix, namely p_1 and p_0, where $p_1 + p_0 = 1$. If the weight for a 0-entry is 1, then the weight of a 1-entry is $\log(p_1)/\log(p_0)$.

Unlike the other methods, `Barycentric` is almost unaffected by the type of noise (symmetric, asymmetric), and needs no weight parameter. Once we have analyzed a weight parameter from the input data, `AlternatingBand` and `Spectral` perform well. Until noise level 0.6 these methods are the best

Runtime (ms) Matrix size	`Barycentric` (100 iterations)	`Alternating` (100 iterations)	`Hamiltonian`	`Spectral`	`Simulated Annealing`
50×50	2	12	4	8	136,000
200×200	24	150	120	170	–
800×800	240	2,110	4,850	12,050	–
3200×3200	4,250	44,300	344,800	1,163,000	–

Table 7.1: Running times for the algorithms (in ms) with matrices that have 50% fill and 10% symmetric noise.

at recovering original permutations that establish almost direct bandedness. As was the case with symmetric noise, the `Hamiltonian` method has larger variance and has difficulties in recovering the original order.

Table 7.1 shows running-time examples for the algorithms. The fastest methods are `Barycentric` and `AlternatingBand`, even though both perform 100 iterations. Their running times increase steadily with the size of an input matrix, whereas the increase is more pronounced for `Hamiltonian` and `Spectral`. Overall, most of the methods are able to handle large matrices in reasonable time.

7.11 Experiments on real-world data

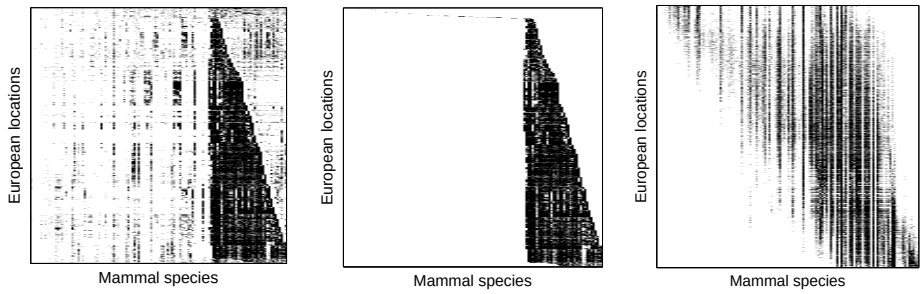
In the next sections we will present a variety of real-world datasets, especially from life sciences. We show that our algorithms discover almost banded patterns and the results have meaningful interpretations.

7.11.1 Mammals data

The Mammals dataset was described in Section 6.7.1. We use its transpose: we have presence/absence records of 124 European mammals (columns) in 2,179 locations (rows).

Figure 7.7(a) shows the band obtained by the `AlternatingBand` algorithm that uses the Hamming distance. We see an almost nested structure of locations and mammals so far unknown in the dataset. The correlation between the ordering of the sites and the temperature variable is 0.64, which affirms that the temperature of a location heavily affects the diversity of mammal species living there. Furthermore, the nested structure is dense: of the 54,155 total 1s in the dataset, 70% of them are accumulated within the direct band retrieved by `AlternatingBand` (Figure 7.7(b)). The matrix has Hamming distance 23,997 to a banded matrix (upper bound).

For comparison, Figure 7.7(c) shows the data as seen by the `Barycentric`



(a) Mammals data by `AlternatingBand`

(b) Extracted banded part from (a)

(c) Mammals data by `Barycentric`

Figure 7.7: Permuted Mammals data. Displayed on the left (a) is the data permuted by `AlternatingBand` that uses the Hamming distance; in the middle (b) is a depiction of the borders of the band, that is, all the 1s in the Mammals data (70%) that belong to the pattern; on the right (c) is the Mammals data as permuted by `Barycentric`. The permutation of the locations (rows) has strong correlations with latitude (0.92) and average temperatures (−0.85) of the locations.

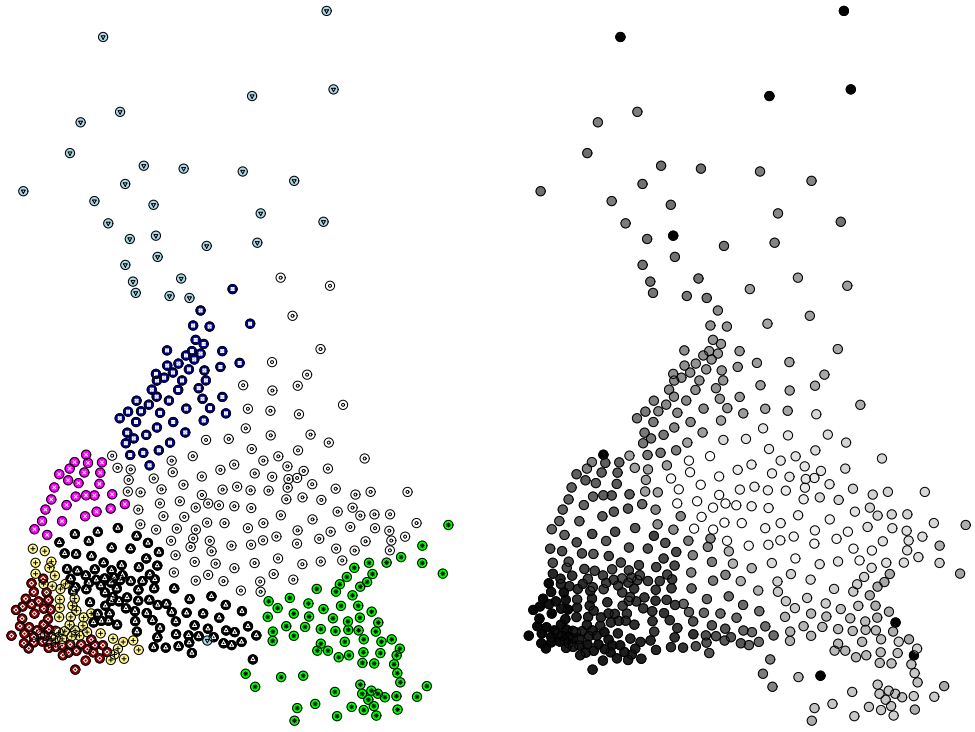
method. The Hamming distance from this matrix to a banded matrix is 42041 (upper bound). Nonetheless, the row ordering has high correlation with certain variables associated with the location, such as latitude (0.92) and average temperature (−0.85).

It seems that both `AlternatingBand` and `Barycentric` produce reasonable, yet visually different results here: `AlternatingBand` prefers large dense clusters of 1s, whereas `Barycentric` assumes that all the rows and columns belong to the banded pattern. Which shape is preferable depends on the application. Note that assigning different weighting schemes causes the `AlternatingBand` method to change the shape of the band.

7.11.2 Dialect data

The Dialect dataset [EW00], originally published in 1940, contains data about the usage of dialectical features in spoken Finnish language. The data is in binary form and represents 1,334 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 the Western and Eastern dialects; further divisions bring out more detail inside these two main dialects. Figure 7.8(a) displays the known division into eight dialect areas.

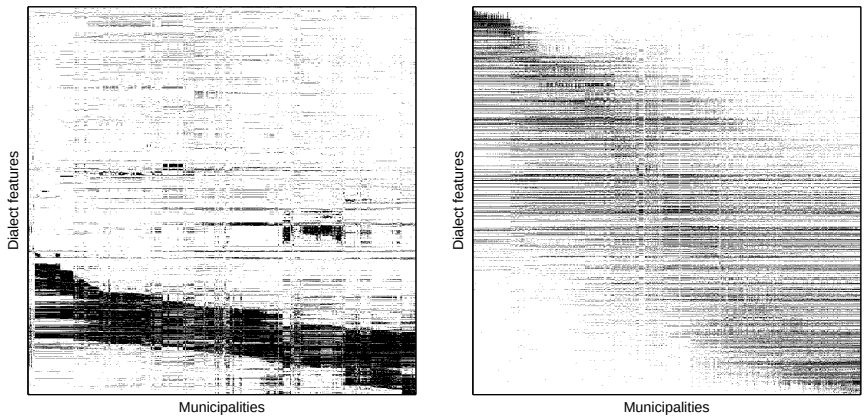


(a) Traditional division into 8 dialect areas

(b) Municipalities in band-order

Figure 7.8: Displayed on the left (a) is the traditional division of the Finnish-speaking municipalities into eight areas by their dialects; on the right (b) is a visualization of an ordering of the municipalities, obtained from `AlternatingBand`. The position of a municipality in the band-ordering is depicted by its color, ranging from black to white. For each area in (a) the associated municipalities share roughly the same grayscale color in (b). The ordering of the band has captured the main variation between the Western and Eastern dialects without any use of spatial data.

We used both `AlternatingBand` and `Barycentric` on the Dialect data to uncover a band structure. In Figure 7.9(a) we display the best ordering found by `AlternatingBand` after 1,000 iterations. 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 7.9(b) shows the band of the `Barycentric` method. The two bands are very different: while `AlternatingBand` aims at creating dense bands of 1s, the `Barycentric` method aims at creating wider banded patterns.



(a) `AlternatingBand` band (b) `Barycentric` band

Figure 7.9: Displayed on the left (a) is the Dialect data permuted by `AlternatingBand`; on the right (b) is the data permuted by `Barycentric`. The Hamming distances to direct bandedness are 54,461 and 105,410.

From `AlternatingBand` we obtain a permutation for municipalities, which can be interpreted as a crude estimate of the dialect spoken in a municipality: the ordering captures the variation from the Western dialects to Eastern dialects. Indeed, the municipalities interpolated from the ordering are plotted in Figure 7.8(b). We can see that this interpolation is similar to the known division shown in Figure 7.8(a), despite some outliers. The outliers are those municipalities that have exceptionally small number of data points (1s). Of course the dialectical variation cannot be fully described by a single ordering: some municipalities do not fit anywhere in the linear order, for example those in Karelia. The results show many geographically coherent dialect areas, although the algorithm did not have access to spatial information about the municipalities.

7.11.3 Paleontological data

Recall the Paleontological data in Section 6.7.2: we have binary information on 139 fossil genera (columns) from 124 sites (rows). The banded structure found in the Paleontological dataset is shown in Figure 7.10. The result is given by the `AlternatingBand` method, and the ordering of the rows reflects the relative ages of the sites.

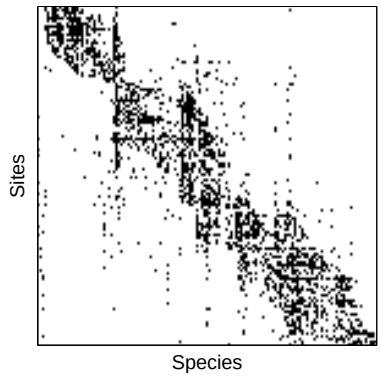


Figure 7.10: The Paleontological data permuted by `AlternatingBand` where the weights for the 1-entries are four times those for the 0-entries.

As is common with data gathering in real-world, the Paleontological data has more missing 1s than false 1s. Therefore 0-entries should have smaller weights than 1-entries. In general it is difficult to select the best weights for a real-world application, and many criteria for "correct" band-shape can be given. On this dataset, we decided that each 1-entry has a weight that is four times that of a 0-entry, as it matches our estimation of error rates.

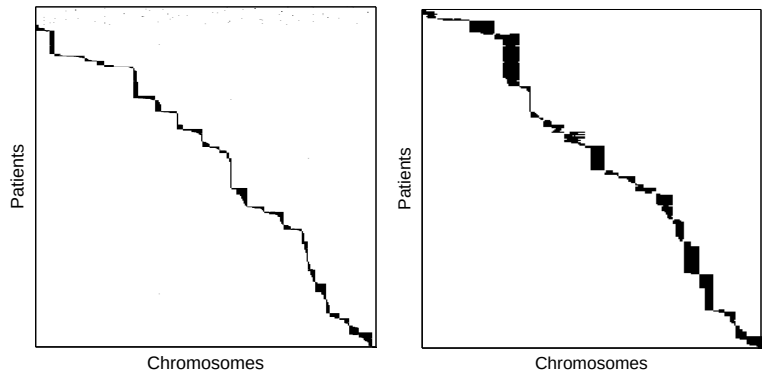
7.11.4 DNA amplification data

The DNA amplification data is available upon request from the authors of [MHB$^+$06]. It contains information on the DNA copy number amplifications recorded in 4,590 cases (rows) and 393 specific chromosomal locations (columns). A 0-entry denotes no DNA copy number amplification in the corresponding chromosomal location, and a 1-entry 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 `FindSubmatrix` (Algorithm 1) to find such banded submatrices. We use `SpectralOrdering` with dot product to reorder the columns, and `FindColBandedConf` to obtain ground-truth matrices. As utility measures, we use the performance measures accuracy and precision, while allowing a Hamming distance of at most 100 to a banded submatrix.

Figures 7.11(a,b) show the two submatrices retrieved from the original data, found by accuracy and precision. For both submatrices we observe immediately that the distance to a directly banded submatrix is relatively small, with almost no noise outside the band. The submatrices are largely distinct: they identify different reduced collections of cancer types.



(a) DNA submatrix with accuracy (b) DNA submatrix with precision

Figure 7.11: Two submatrices from the DNA data. Displayed on the left (a) is a 1,402 × 282 submatrix, found by `FindSubmatrix` that uses accuracy as the utility function; on the right (b) is a 291 × 244 submatrix (with precision). Both submatrices have their Hamming distances to direct bandedness at most 100. We used a combination of `FindColBandedConf`, `SpectralOrdering`, and dot product to compute accuracy and precision.

7.12 Conclusions

We introduced a new data mining concept, banded patterns in binary matrices, which illustrates the variation of attributes in the data. In particular, bandedness is a generalization of nested patterns, and a special case of SC1P matrices. The theoretical results on bandedness, both old and new, give rise to polynomial-time recognition algorithms as well as exact and

heuristic algorithms for finding a closest banded matrix. The idea is to use a combinatorial view on a dataset to uncover hidden banded structures from the data.

The 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 use synthetically generated data to show that the proposed algorithms are able to detect almost banded structures in reasonable time.

Experiments on real-world datasets show that bands occur in a wide range of applications, from life sciences to dialect data. For two of the datasets, we discover previously unknown banded structures that have natural interpretations in the final ordering of the 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 banded structure has captured the main variation between the Eastern and Western dialects of the spoken Finnish language.

We see several directions for future work. First, relaxing the requirement of binary values opens an opportunity to form banded patterns from, say, positive integers or real values. The structure in a banded pattern can then be described through the variation from large to small values within the pattern. Second, finding banded submatrices more efficiently [ABJ10] allows analyzing larger datasets. Third, analyzing `TestBanded` (Algorithm 12) and the inserted rows more closely may improve the time complexity. Fourth, for applications it would be crucial to develop a null model for bandedness: is the banded structure statistically significant in data?

Chapter 8

Discussion

We have described a framework of reorderable patterns that considers different permutations of the rows and columns of a matrix. The goal is not to modify the data, but to reorganize the data so as to reveal its hidden structure.

The reorderable patterns presented in this thesis form a sequence in which a pattern includes all matrices that have a more specific pattern, namely, let $\mathcal{N}$, $\mathcal{B}$, $\mathcal{S}$, $\mathcal{C}$, $\mathcal{Z}$, and $\mathcal{M}$ be the following patterns: **n**ested, **b**anded, **S**C1P, **C**1P, **z**ero-partitionable, and all binary **m**atrices. Then we have the following chain of proper inclusions.

$$\mathcal{N} \subset \mathcal{B} \subset \mathcal{S} \subset \mathcal{C} \subset \mathcal{Z} \subset \mathcal{M} \tag{8.1}$$

Of course, this selection of patterns is incomplete and many patterns are missing; for example, the pattern of k-nestedness cannot be included in the chain. Another matrix pattern not studied here, but related to k-nestedness, is *laminarity*: in a collection of sets each two sets are either disjoint or one is a subset of the other.

To see that the inclusions are proper, we give examples of minimum-size matrices that separate the patterns in (8.1). The first matrix in (8.2) is not in $\mathcal{N}$ but is in $\mathcal{B}$, and the last one is not in $\mathcal{Z}$ but it is in $\mathcal{M}$; the rest of the matrices fall between the patterns similarly.

$$\underset{\mathcal{B}}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} \quad \underset{\mathcal{S}}{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}} \quad \underset{\mathcal{C}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}} \quad \underset{\mathcal{Z}}{\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}} \quad \underset{\mathcal{M}}{\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}} \tag{8.2}$$

By studying the combinatorial properties of consecutive, nested, k-nested, and banded patterns, we developed efficient algorithms for recog-

nizing these patterns in noise-free binary matrices. For example, there exist polynomial-time algorithms for recognizing the following patterns: C1P, SC1P, nestedness, k-nestedness, and bandedness. If the dataset contains noise, however, the problem becomes that of finding a closest matrix that has the reorderable pattern. Alas, the results on computational complexity suggest that such problems are often NP-hard, and we need to settle for heuristic or approximative approaches.

When solving some matrix-related problems it is convenient to view a binary matrix as a bipartite graph, and vice versa. Thus, we can have the best from both worlds, which helps to develop both theory and algorithms. The field of data analysis has long been dominated by real-valued methods such as principal component analysis and clustering. Therefore it is intriguing to see that purely combinatorial methods find meaningful patterns in real-world datasets. However simple datasets binary matrices may present, matrix decomposition methods designed for binary data perform well [MMG$^+$08] in comparison to more established decomposition methods.

Some notable problems remain open. In this thesis we concentrated on weighted distances, but we can consider each specific distance separately, such as the augmentation, deletion and Hamming distances. Indeed, the computational complexity results are known only for some of these distances. One notable open problem is the time complexity of d_{H}-Closest Nested. In addition, we do not know the complexity of computing the d_{H}-distance to a closest C1P, SC1P, or banded matrix. Same holds for d_{A}-Closest Banded and Closest Columns-Banded. We expect that all these problems are NP-hard. On the other hand, there is a polynomial-time algorithm [Mül97] for recognizing whether a graph is a directed interval graph. Since zero-partitionable matrices and adjacency matrices of directed graphs differ only by diagonal entries (self-loops), this could lead to a polynomial-time algorithm for checking zero-partitionability.

The experimental results reported in this thesis show that a variety of structures that appear in real-world datasets can be described by reorderable patterns. Furthermore, new heuristic algorithms are able to detect these patterns automatically. Once detected, these patterns offer a different view on the dataset, which often helps to understand the hidden structure and processes behind the data. In particular, tests on synthetic data verify that the heuristic methods find reorderable patterns reliably. However, since reordering the rows and columns only produces one linear order for each set of attributes, the descriptive power of reorderable patterns is limited in cases where the structure is richer than that.

Detecting that a real-world dataset almost has a pattern is not enough—we need a statistical significance test to assess whether the pattern is exceptional or a mere occurrence of random chance. In this thesis we concentrated on nestedness and showed that the selection of a null model is not straighforward. The other patterns would benefit from a null model, too: for example, `Contin` could be used to test whether bandedness is significant.

Even if we find a pattern in a real-world dataset, we cannot tell whether the pattern has an important meaning or if it is just a by-product of another pattern. For example, we can see a nested dataset as a C1P matrix, if we are looking for such consecutive patterns, but consecutiveness alone cannot explain all the processes behind the dataset. Of course, simple models and patterns cannot fully explain real-world datasets, but they still give a direction. To distinguish better between different pattern classes, such as C1P and nestedness, we need specifically developed null models.

The distance from a matrix to a pattern is an absolute measure. In addition to using statistical inference, we could also develop a relative measure. To measure how strong a pattern is in a matrix, we can take into account, for example, the distance, the dimensions of the matrix, and the number of 1s. We can employ also a null model as a part of the measure, like in nestedness intensity.

It is worthwhile to ask whether there are any alternatives to the flips-based distance measure used. The approach in this thesis is based on earlier work on edit distances that assume independent entries and flips. Still, in some applications it might be beneficial to fine-tune the distance measure to deal with known dependencies in the dataset. Indeed, adding dependency to the error models may produce more realistic matrices, which increases the relevancy of synthetic tests. Developing such a dependent error model is of course application-sensitive, and requires expertise on the topic at hand.

A generalization from binary patterns to multivariate patterns is possible. For example, a generalization from binary nestedness into multivariate values could contain any nonnegative integers, with largest values in the innermost layer (top-left corner) and 0s in the outermost layer of values. We can also consider generalizing the patterns into higher dimensions: the patterns described in this thesis appear on two-dimensional matrices, but also more general patterns on n-dimensional tensors can be studied. For example, a three-dimensional tensor could contain two-dimensional data at several points in time. In addition to global patterns and submatrices, we could also consider datasets that consist of two parts: pattern-part and noise-part. The challenge would be to identify the rows and columns of the data that have relevant structure, despite noisy rows and columns.

References

[ABH98] Jonathan E. Atkins, Erik G. Boman, and Bruce Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28(1):297–310, 1998.

[ABJ10] Faris Alqadah, Raj Bhatnagar, and Anil G. Jegga. Mining maximally banded matrices in binary data. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM'10)*, pages 942–953, 2010.

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216, 1993.

[AKNW95] Farid Alizadeh, Richard M. Karp, Lee Aaron Newberg, and Deborah K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Algorithmica*, 13(1–2):52–76, 1995.

[ANGG+08] Mário Almeida-Neto, Paulo Guimarães, Paulo R. Guimarães, Jr, Rafael D. Loyola, and Werner Ulrich. A consistent metric for nestedness analysis in ecological systems: reconciling concept and measurement. *Oikos*, 117(8):1227–1239, 2008.

[ANGL07] Mário Almeida-Neto, Paulo R. Guimarães, Jr, and Thomas M. Lewinsohn. On nestedness analyses: rethinking matrix temperature and anti-nestedness. *Oikos*, 116(4):716–722, 2007.

[AP93] Wirt Atmar and Bruce D. Patterson. The measure of order and disorder in the distribution of species in fragmented habitat. *Oecologia*, 96(3):373–382, 1993.

[APÇ04] Cevdet Aykanat, Ali Pinar, and Ümit V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form.

SIAM Journal on Scientific Computing, 25(6):1860–1879, 2004.

[AV07] David Arthur and Sergei Vassilvitskii. `k-means++`: the advantages of careful seeding. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 1027–1035, 2007.

[BBD06] Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006.

[Ben84] Jon Bentley. Programming pearls: algorithm design techniques. *Communications of the ACM*, 27(9):865–871, 1984.

[Ber99] Jacques Bertin. Graphics and graphic information processing. In Stuart K. Card, Jock Mackinlay, and Ben Shneiderman, editors, *Readings in information visualization: using vision to think*, pages 62–65. Morgan Kaufmann, San Francisco, USA, 1999.

[Ber02] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, Inc., 2002.

[BJMO03] Jordi Bascompte, Pedro Jordano, Carlos J. Melián, and Jens M. Olesen. The nested assembly of plant-animal mutualistic networks. *Proceedings of the National Academy of Sciences USA (PNAS)*, 100(16):9383–9387, 2003.

[BK88] James C. Brower and Kenneth M. Kile. Seriation of an original data matrix as applied to paleoecology. *Lethaia*, 21(1):79–93, 1988.

[BKG+05] Arindam Banerjee, Chase Krumpelman, Joydeep Ghosh, Sugato Basu, and Raymond J. Mooney. Model-based overlapping clustering. In *Proceedings of the 11th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'05)*, pages 532–537, 2005.

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[Boo75] Kellogg S. Booth. *PQ-tree algorithms*. PhD thesis, University of California, 1975.

[Bru80] Richard A. Brualdi. Matrices of zeros and ones with fixed row and column sum vectors. *Linear Algebra and its Applications*, 33:159–231, 1980.

[BRV10] Guillaume Blin, Romeo Rizzi, and Stéphane Vialette. A faster algorithm for finding minimum Tucker submatrices. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes*, volume 6158 of *Lectures Notes in Computer Science*, pages 69–77. 2010.

[BS99] Richard Brualdi and James Sanderson. Nested species subsets, gaps, and discrepancy. *Oecologia*, 119(2):256–264, 1999.

[BYGL+08] Ziv Bar-Yossef, Ido Guy, Ronny Lempel, Yoëlle S. Maarek, and Vladimir Soroka. Cluster ranking with an application to mining mailbox networks. *Knowledge and Information Systems*, 14(1):101–139, 2008.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[CC03] George W. Cobb and Yung-Pin Chen. An application of Markov chain Monte Carlo to community ecology. *American Mathematical Monthly*, 110(4):265–288, 2003.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report Report 388, Graduate School of Industrial Administration, CMU, 1976.

[CHSY10] Cedric Chauve, Utz-Uwe Haus, Tamon Stephen, and Vivija P. You. Minimal conflicting sets for the consecutive ones property in ancestral genome reconstruction. *Journal of Computational Biology*, 17(9):1167–1181, 2010.

[CL09] Keke Chen and Ling Liu. "Best K": critical clustering structures in categorical datasets. *Knowledge and Information Systems*, 20(1):1–33, 2009.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[CM69] Elizabeth Cuthill and James M. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th ACM National Conference*, pages 157–172, 1969.

[CNHS00] Emmanuelle Cam, James D. Nichols, James E. Hines, and John R. Sauer. Inferences about nested subsets structure when not all species are detected. *Oikos*, 91(3):428–434, 2000.

[DG99] Jitender S. Deogun and K. Gopalakrishnan. Consecutive retrieval property — revisited. *Information Processing Letters*, 69(1):15–20, 1999.

[DGN10] Michael Dom, Jiong Guo, and Rolf Niedermeier. Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *Journal of Computer and System Sciences*, 76(3–4):204–221, 2010.

[Die10] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, fourth edition, 2010.

[Dil50] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *The Annals of Mathematics, Second Series*, 51(1):161–166, 1950.

[Eli75] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21(2):194–203, 1975.

[EW00] Sheila M. Embleton and Eric S. Wheeler. Computerized dialect atlas of Finnish: Dealing with ambiguity. *Journal of Quantitative Linguistics*, 7(3):227–231, 2000.

[FG65] Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[FL02] Joern Fischer and David B. Lindenmayer. Treating the nestedness temperature calculator as a "black box" can lead to false conclusions. *Oikos*, 99(1):193–199, 2002.

[FL05] Joern Fischer and David B. Lindenmayer. Perfectly nested or significantly nested – an important difference for conservation management. *Oikos*, 109(3):485–494, 2005.

[FM95] Gerhard H. Fischer and Ivo W. Molenaar, editors. *Rasch Models: Foundations, Recent Developments, and Applications.* Springer-Verlag, 1995.

[FMT09] Tomás Feder, Heikki Mannila, and Evimaria Terzi. Approximating the minimum chain completion problem. *Information Processing Letters*, 109(17):980–985, 2009.

[For08] Mikael Fortelius. Neogene of the old world database of fossil mammals (NOW), 2008. URL: `http://www.helsinki.fi/science/now`.

[FSO+10] Miguel A. Fortuna, Daniel B. Stouffer, Jens M. Olesen, Pedro Jordano, David Mouillot, Boris R. Krasnov, Robert Poulin, and Jordi Bascompte. Nestedness versus modularity in ecological networks: two sides of the same coin? *Journal of Animal Ecology*, 79(4):811–817, 2010.

[Gal86] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *Computing Surveys*, 18(1):23–38, 1986.

[GC06] Michelle Greve and Steven L. Chown. Endemicity biases nestedness metrics: a demonstration, explanation and solution. *Ecography*, 29(3):347–356, 2006.

[GD82] Michael Gilpin and Jared Diamond. Factors contributing to non-randomness in species co-occurrences on islands. *Oecologia*, 52(1):75–84, 1982.

[GGKS95] Paul W. Goldberg, Martin C. Golumbic, Haim Kaplan, and Ron Shamir. Four strikes agains physical mapping of DNA. *Journal of Computational Biology*, 2(1):139–152, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman Co, New York, 1979.

[GJM08] Gemma C. Garriga, Esa Junttila, and Heikki Mannila. Banded structure in binary matrices. In *Proceedings of the 14th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'08)*, pages 292–300, 2008.

[GJM11] Gemma C. Garriga, Esa Junttila, and Heikki Mannila. Banded structure in binary matrices. *Knowledge and Information Systems*, 28(1):197–226, 2011. doi: 10.1007/s10115-010-0319-7.

[GKSS08] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 2. Elsevier, 2008.

[GN02] Michelle Girvan and Mark E. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences USA (PNAS)*, 99(12):7821–7826, 2002.

[Goe03] Bart Goethals. Survey on frequent pattern mining. Technical report, Helsinki Institute for Information Technology HIIT, 2003.

[Got00] Nicholas J. Gotelli. Null model analysis of species co-occurrence patterns. *Ecology*, 81(9):2606–2621, 2000.

[GPI09] Javier Galeano, Juan M. Pastor, and Jose M. Iriondo. Weighted-interaction nestedness estimator (WINE): a new estimator to calculate over frequency matrices. *Environmental Modelling & Software*, 24(11):1342–1346, 2009.

[HFEM07] Hannes Heikinheimo, Mikael Fortelius, Jussi Eronen, and Heikki Mannila. Biogeography of European land mammals shows environmentally distinct and spatially coherent clusters. *Journal of Biogeography*, 34(6):1053–1064, 2007.

[HG02] Mohammad Taghi Hajiaghayi and Yashar Ganjali. A note on the consecutive ones submatrix problem. *Information Processing Letters*, 83(3):163–166, 2002.

[HLO08] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[Hsu02] Wen-Lian Hsu. A simple test for the consecutive ones property. *Journal of Algorithms*, 43(1):1–16, 2002.

[HWS06] Christopher L. Higgins, Michael R. Willig, and Richard E. Strauss. The role of stochastic processes in producing nested patterns of species distributions. *Oikos*, 114(1):159–167, 2006.

[HY01] Mark H. Hansen and Bin Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774, 2001.

[JK11] Esa Junttila and Petteri Kaski. Segmented nestedness in binary data. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM'11)*, pages 235–246, 2011.

[JTM10] Esa Junttila, Evimaria Terzi, and Heikki Mannila. Sensitivity of nestedness measures to noise. Unpublished manuscript, 2010.

[JXFD08] Ruoming Jin, Yang Xiang, David Fuhry, and Feodor F. Dragan. Overlapping matrix pattern visualization: a hypergraph approach. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08)*, pages 313–322, 2008.

[KGV83] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Knu93] Donald E. Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. ACM, New York, USA, 1993.

[Lii10] Innar Liiv. Seriation and matrix reordering methods: An historical overview. *Statistical Analysis and Data Mining*, 3(2):70–91, 2010.

[LIPJ+06] Thomas M. Lewinsohn, Paulo Inácio Prado, Pedro Jordano, Jordi Bascompte, and Jens M. Olesen. Structure in plant–animal interaction assemblages. *Oikos*, 113(1):174–184, 2006.

[LSW97] In-Jen Lin, Malay K. Sen, and Douglas B. West. Classes of interval digraphs and 0,1-matrices. In *Proc. 28th SE Conf. Congressus Numer.*, volume 125, pages 201–209, 1997.

[LW95] In-Jen Lin and Douglas B. West. Interval digraphs that are indifference digraphs. In Y. Alavi and A. Schwenk, editors, *Graph theory, Combinatorics, and Algorithms*, pages 751–765. Wiley, New York, 1995. Also in: Proc. 7th Intl. Conf. Graph Th. Comb. Alg., Kalamazoo, 1992.

[LY80] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[Max11] Max-SAT Evaluation web page, 2011. URL: `http://maxsat.ia.udl.cat/`.

[MHB+06] Samuel Myllykangas, Johan Himberg, Tom Böhling, Balint Nagy, Jaakko Hollmén, and Sakari Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.

[Mie05] Taneli Mielikäinen. *Summarization Techniques for Pattern Collections in Data Mining*. PhD thesis, University of Helsinki, 2005.

[MMG+08] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1348–1362, 2008.

[MO04] Sara C. Madeira and Arlindo L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[MP04] István Miklós and János Podani. Randomization of presence-absence matrices: comments and new algorithms. *Ecology*, 85(1):86–92, 2004.

[MS00] Erkki Mäkinen and Harri Siirtola. Reordering the reorderable matrix as an algorithmic problem. In *Proceedings of the 1st International Conference on the Theory and Application of Diagrams (Diagrams'00)*, pages 453–467. Springer-Verlag, 2000.

[MS05] Erkki Mäkinen and Harri Siirtola. The barycenter heuristic and the reorderable matrix. *Informatica*, 29(3):357–363, 2005.

[MS07] Jeffrey E. Moore and Robert K. Swihart. Toward ecologically explicit null models of nestedness. *Oecologia*, 152(4):763–777, 2007.

[MT07] Heikki Mannila and Evimaria Terzi. Nestedness and segmented nestedness. In *Proceedings of the 13th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 480–489, 2007.

[Mül97] Haiko Müller. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, 78:189–205, 1997.

[MZK+09] Gabriela Moise, Arthur Zimek, Peer Kröger, Hans-Peter Kriegel, and Jörg Sander. Subspace and projected clustering: experimental evaluation and analysis. *Knowledge and Information Systems*, 21(3):299–326, 2009.

[NB07] Anders Nielsen and Jordi Bascompte. Ecological networks, nestedness and sampling effort. *Journal of Ecology*, 95(5):1134–1141, 2007.

[NH79] Simeon C. Ntafos and S. Louis Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, 5(5):520–529, 1979.

[OR00] Marcus Oswald and Gerhard Reinelt. Polyhedral aspects of the consecutive ones problem. In Ding-Zhu Du, Peter Eades, and Xuemin Lin, editors, *Computing and Combinatorics*, volume 1858 of *Lecture Notes in Computer Science*, pages 373–382. Springer, 2000.

[OR03] Marcus Oswald and Gerhard Reinelt. The weighted consecutive ones problem for a fixed number of rows or columns. *Operations Research Letters*, 31(5):350–356, 2003.

[OR09] Marcus Oswald and Gerhard Reinelt. The simultaneous consecutive ones problem. *Theoretical Computer Science*, 410(21–23):1986–1992, 2009.

[Osw03] Marcus Oswald. *Weighted consecutive ones problems*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2003.

[PA86] Bruce D. Patterson and Wirt Atmar. Nested subsets and the structure of insular mammalian faunas and archipelagos. *Biological Journal of the Linnean Society*, 28(1–2):65–82, 1986.

[Pap76] Christos H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[Pat87] Bruce D. Patterson. The principle of nested subsets and its implication for biological conservation. *Conservation Biology*, 1(4):323–334, 1987.

[PFM06] Kai Puolamäki, Mikael Fortelius, and Heikki Mannila. Seriation in paleontological data using Markov chain Monte Carlo methods. *PLoS Computational Biology*, 2(2), 2006.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.

[Ras60] George Rasch. *Probabilistic Models for Some Intelligence and Attainment Tests.* Danmarks Paedogogiske Institut, Copenhagen, Denmark, 1960. Reprinted 1993 by MESA Press.

[RGS06] Miguel A. Rodríguez-Gironés and Luis Santamaría. A new algorithm to calculate the nestedness temperature of presence–absence matrices. *Journal of Biogeography*, 33(5):924–935, 2006.

[RJB07] Enrico L. Rezende, Pedro Jordano, and Jordi Bascompte. Effects of phenotypic complementarity and phylogeny on the nested structure of mutualistic networks. *Oikos*, 116(11):1919–1929, 2007.

[Rob69] Fred S. Roberts. Indifference graphs. In Frank Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, New York, 1969.

[Ros68] Richard Rosen. Matrix bandwidth minimization. In *Proceedings of the 1969 23rd ACM National Conference*, pages 585–595, 1968.

[RSL77] Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[Rys57] Herbert John Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957.

[Rys60] Herbert John Ryser. Matrices of zeros and ones. *Bulletin of the American Mathematical Society*, 66(6):442–464, 1960.

[SAP10] Brice X. Semmens, Peter J. Auster, and Michelle J. Paddack. Using ecological null models to assess the potential for marine protected area networks to protect biodiversity. *PLoS ONE*, 5(1), January 2010.

[Sat11] The international SAT competitions web page, 2011. URL: `http://www.satcompetition.org/2011/`.

[SF07] Nuria Selva and Miguel A. Fortuna. The nested structure of a scavenger community. *Proceedings of the Royal Society B*, 274(1613):1101–1108, 2007.

[SS94] Malay K. Sen and Barum K. Sanyal. Indifference digraphs: A generalization of indifference graphs and semiorders. *SIAM Journal on Discrete Mathematics*, 7(2):157–165, 1994.

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.

[TMZ99] Peng Tian, Jian Ma, and Dong-Mo 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.

[TP07] Juan T. Timi and Robert Poulin. Different methods, different results: temporal trends in the study of nested subset patterns in parasite communities. *Parasitology*, 135(1):131–138, 2007.

[Tuc72] Alan Tucker. A structure theorem for the consecutive 1's property. *Journal of Combinatorial Theory, Series B*, 12(2):153–162, 1972.

[TZ07] Jinsong Tan and Louxin Zhang. The consecutive ones submatrix problem for sparse matrices. *Algorithmica*, 48(3):287–299, 2007.

[UANG09] Werner Ulrich, Mário Almeida-Neto, and Nicholas J. Gotelli. A consumer's guide to nestedness analysis. *Oikos*, 118(1):3–17, 2009.

[UG07a] Werner Ulrich and Nicholas J. Gotelli. Disentangling community patterns of nestedness and species co-occurrence. *Oikos*, 116(12):2053–2061, 2007.

[UG07b] Werner Ulrich and Nicholas J. Gotelli. Null model analysis of species nestedness patterns. *Ecology*, 88(7):1824–1831, 2007.

[UPGM09] Antti Ukkonen, Kai Puolamäki, Aristides Gionis, and Heikki Mannila. A randomized approximation algorithm for computing bucket orders. *Information Processing Letters*, 109(7):356–359, 2009.

[Vel85] Marinus Veldhorst. Approximation of the consecutive ones matrix augmentation problem. *SIAM Journal on Computing*, 14(3):709–729, 1985.

[vL07] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[Vuo10] Niko Vuokko. Consecutive ones property and spectral ordering. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM'10)*, pages 350–360, 2010.

[Wes98] Douglas B. West. Short proofs for interval digraphs. *Discrete Mathematics*, 178:287–292, 1998.

[WPM+98] David H. Wright, Bruce D. Patterson, Greg M. Mikkelson, Alan Cutler, and Wirt Atmar. A comparative analysis of nested subset patterns of species composition. *Oecologia*, 113(1):1–20, 1998.

[WZ98] Bo-Ying Wang and Fuzhen Zhang. On the precise number of (0,1)-matrices in $\mathfrak{A}(R, S)$. *Discrete Mathematics*, 187(1–3):211–220, 1998.

[Yan81a] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[Yan81b] Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981.

[YCM98] Chang-Wu Yu, Gen-Huey Chen, and Tze-Heng Ma. On the complexity of the k-chain subgraph cover problem. *Theoretical Computer Science*, 205(1–2):85–98, 1998.

Index

Notation

Notation	Explanation
$\subseteq, \subset$	subset, proper subset
$[k]$	set $\{1, 2, \ldots, k\}$
$\mathbf{a}$	vector
a_i	entry of the vector $\mathbf{a}$ at index i
A	matrix, set, or list
$a_{r,c}$	entry of the matrix A at row index r, column index c
A^T	transpose of a matrix A (for all entries $a^T_{r,c} = a_{c,r}$)
$\lvert s \rvert$	s is a number: absolute value of the number s
$\lvert S \rvert$	S is a set: cardinality of the set S (the number of elements)
$\widetilde{A}$	band-conflict matrix constructed from a matrix A
$\widehat{A}$	matrix that is in some sense close to a matrix A
R_i, C_j	set interpretation of the row i and column j (positions of 1s)
$\langle s, e \rangle$	consecutive binary vector with 1s on the indices $s, s+1, \ldots, e-1$ and 0s elsewhere
$m \times n$	size of a matrix: m rows, n columns
$U \times V$	cartesian product of two sets U and V
$d_W(A, B)$	distance from the matrix A to B relative to weights W
d_{A}, d_{D}, d_{H}	augmentation, deletion, and Hamming distance on matrices
$G = (V, E)$	graph with the vertex set V and edge set E
$G = (R \cup C, E)$	bipartite graph with vertex sets R (rows) and C (columns) and edge set $E \subseteq R \times C$
$\mathcal{P}$	pattern: set of binary matrices that have a certain property
π, σ, τ	permutation; σ for rows, τ for columns
Problem	font used for problems
`Method`	font used for algorithms and methods
Other	font used for other technical elements

Errors found in my doctoral thesis *Patterns in Permuted Binary Matrices*

Esa Junttila

August 8, 2011

Pages 46 and 47: Algorithm 10, Theorems 4.9 and 4.10 The reasoning in the proof of Theorem 4.9 has a flaw. The entry (r_3, c_3) used in the proof is assumed to have value 1 in the input matrix. Yet, it is possible that the entry has been flipped from 0 to 1 by Algorithm 10. There are matrices where Algorithm 10 generates a new switch box, which makes the resulting matrix non-nested.

An example: Given the matrix A below, Algorithm 10 transforms it to B, which contains a switch box.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \rightsquigarrow \quad B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Consequences:

1. The proofs for Theorems 4.9 and 4.10 are not valid. Both Theorems may still be true, but no proofs or counter-examples are known for them.
2. Tight examples (in the proof of Theorem 4.10) hold nonetheless.
3. Repeating Algorithm 10 will lead to a nested matrix. This works because a submatrix full of 1s is nested.
4. On page 48, the corrective move in Algorithm 11 can be implemented by repeating Algorithm 10 (instead of running it only once).
5. The rest of the thesis remains unaffected.