Combinatorial Algorithms for the Seriation Problem

Matteo Seminaroti

This research has been carried out at the Centrum Wiskunde & Informatica in Amsterdam.

Combinatorial Algorithms for the Seriation Problem

Proefschrift

ter verkrijging van de graad van doctor aan
Tilburg University
op gezag van de rector magnificus,
prof.dr. E.H.L. Aarts,
in het openbaar te verdedigen ten overstaan van een
door het college voor promoties aangewezen commissie
in de aula van de Universiteit
op vrijdag 2 december 2016 om 10.00 uur

door

Matteo Seminaroti

geboren op 27 april 1988
te Rome, Italië

Acknowledgements

As every story, also my PhD journey comes finally to an end. When I decided to accept this position, I was extremely scared by the difficulty of this new challenge. In fact, these three years have been pretty intense, and if I have reached this point I should thank many people that helped and supported me during my PhD.

It is needless to say that this achievement would have never been possible without the supervision of Monique. With her enthusiasm and passion for research, Monique has been a fundamental point of reference in these three years, teaching me the mathematical reasoning and guiding me through the PhD. From her I have learned an invaluable lesson of life: to always give the maximum in what you do, to keep positive and persist also when insurmountable obstacles occur. I am grateful to her for giving me freedom in carrying on the research, for believing in me and for her infinite patience. It was a privilege to work with her.

Furthermore, I want to thank my co-supervisor Renata Sotirov and the rest of the committee, Edwin van Dam, Michel Habib, Etienne de Klerk, Shinichi Tanigawa and Gerhard Woeginger for their time reviewing the thesis and for the precious feedback to improve the manuscript.

For the outcome of this thesis, I am also extremely grateful to Alexandre d'Aspremont, since from his presentation in Lunteren 2014 we started researching over Robinsonian matrices. Although I did not have external collaborations, I would like to thank Mohammed El-Kebir, Gunnar Klau, Alexander Schrijver and Utz-Uwe Haus for the many discussions and brainstorming sessions. My research was funded by the Initial Training Network MINO, so I thank all the professors, PhD students and experienced researchers involved in the project. I am also grateful to my master thesis supervisor Antonio Sassano, who convinced me to apply for this PhD position, supporting my application. Furthermore, I would like to thank my colleagues at CWI for the nice lunches and the time spent together: Krzysztof Apt, Daniel Dadush, Sander Gribling, Cristobal Guzman, Irving van Heuven, Sabine Burgdorf, Bart de Keijzer, Pieter Kleer, David de Laat, Neil Olver and Guido Schäfer. Thanks also to Minnie Middelberg, Susanne van Dam, Irma van Lunenburg and Karin van Gemert from P&O.

Beside the PhD, in these three years I have met fantastic people I spent most of the time with, and which made my staying in Amsterdam unique. First of all, the Kramatweg crew: my flatmates Gabriele (Re dei Re, Buono e Pacifico, Fortunato e Pio), Falco, and my neighbors Sara and Geert. Special thanks go to Madara and Chaim for the epic trips and adventures, and to Daniele and Nefeli for the nice weekends and the (not so nice) S.S. Lazio games watched often together. Many thanks to Teresa, Valerio, Pablo and Jennifer and to my dear friends from Rome spread all over Europe that I have been visiting in these three years: Alessandro, Maurizio, Stefano, Lorenzo, Davide, Martina, Riccardo and Tommaso.

Most importantly, I would like to thank my family for their unconditional support: my parents Anna and Enzo, my brothers Marco, Andrea, Luca and my sister-in-law Lucia. Finally, this thesis is dedicated to my two beautiful nephews, Davide and Alessio. Maybe in twenty-five years from now they will also be able to read and understand what is written in this thesis (or maybe not!).

Amsterdam, September 2016.

Contents

1 Introduction **1**
1.1 Background and motivation 2
1.2 Contributions 6
1.3 Outline of the thesis 10

I Robinsonian matrices and Lex-BFS **15**

2 Preliminaries **17**
2.1 Proximity matrices 17
2.2 Orderings and partition refinement 19
2.2.1 Permutations 19
2.2.2 (Weak) linear orders 20
2.2.3 Partition refinement 22
2.2.4 PQ-trees 23
2.3 Graphs 24

3 Robinsonian matrices **29**
3.1 Introduction 29
3.1.1 Basic definitions 30
3.1.2 Motivation 33
3.2 Recognition algorithms 35
3.2.1 Binary Robinsonian matrices 36
3.2.2 Graph characterizations 38
3.2.3 Spectral characterization 41
3.3 Conclusions 43

4 Lexicographic Breadth-First Search **45**
4.1 Motivation 45
4.2 The algorithm 47

4.2.1 Lex-BFS 47
4.2.2 Lex-BFS$_+$ 49
4.2.3 Partition refinement implementation 49
4.2.4 Multisweep algorithms 53
4.3 Structural properties 54
4.3.1 Chordal and interval graphs 55
4.3.2 Unit interval graphs 56
4.4 Conclusions 57

II Recognition algorithms for Robinsonian matrices 59

5 Lex-BFS based algorithm 61
5.1 Introduction 61
5.2 Preliminaries 62
5.3 Subroutines 65
5.3.1 Component ordering 65
5.3.2 Straight enumeration 68
5.3.3 Refinements of weak linear orders 70
5.4 The algorithm 71
5.4.1 Correctness 73
5.4.2 Complexity analysis 75
5.5 Finding all Robinson orderings 77
5.6 Example 81
5.7 Conclusions and future work 86

6 Similarity-First Search 89
6.1 Introduction 89
6.2 Preliminaries 90
6.2.1 Basics facts 91
6.2.2 Characterization of anchors 93
6.3 The SFS algorithm 97
6.3.1 Description 97
6.3.2 Characterization of SFS orderings 99
6.3.3 The Path Avoiding Lemma 100
6.3.4 End points of SFS orderings 106
6.4 The SFS$_+$ algorithm 107
6.4.1 Description 107
6.4.2 Similarity layers 110
6.5 The multisweep algorithm 113
6.5.1 Description 113
6.5.2 3-good SFS orderings 116
6.5.3 Proof of correctness 119

6.5.4 Complexity 120
6.5.5 Example 124
6.6 Conclusions and future work 127

III QAP and Robinsonian approximation 129

7 Seriation and the quadratic assignment problem 131
7.1 The Quadratic Assignment Problem 131
7.1.1 Polynomial solvable cases of QAP 132
7.1.2 2-SUM and seriation 133
7.2 QAP is easy for Robinsonian matrices 135
7.2.1 Sketch of proof 136
7.2.2 Intermediate results 137
7.2.3 Conclusion of the proof 139
7.2.4 Applications of the main result 143
7.3 Conclusions and future work 145

8 Robinsonian matrix approximation 147
8.1 Combinatorial aspects 147
8.2 Fitting Robinsonian matrices 149
8.2.1 l_∞-fitting Robinsonian matrices 149
8.2.2 ϵ-Robinsonian recognition 154
8.3 The ϵ-multisweep algorithm 157
8.3.1 ϵ-Similarity-First Search 158
8.3.2 ϵ-Robinson orderings 158
8.3.3 ϵ-SFS-based heuristic 160
8.3.4 Example 162
8.4 Conclusions and future work 164

9 Computational experiments 165
9.1 Design of experiments 165
9.1.1 Parameters 166
9.1.2 Robinsonian matrices generation 167
9.1.3 Error generation 170
9.2 Results with Robinsonian matrices 170
9.3 Results with non-Robinsonian matrices 183
9.4 Conclusions 188

Bibliography 195

List of symbols and abbreviations 205

Index 207

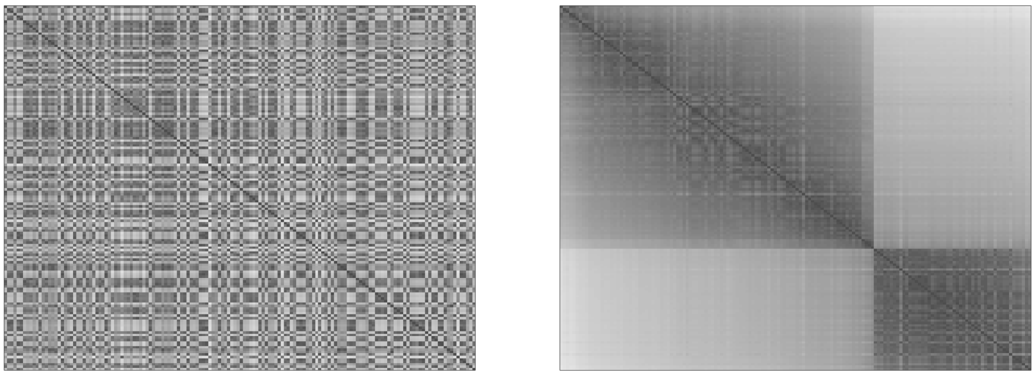
1 Introduction

In many real-world applications it is required to order a given set of objects when the only available information among the objects is their pairwise similarity (or dissimilarity). This is becoming increasingly important, for instance, in applications where companies have to analyze large sets of data which do not have a natural order, like user ratings, images, music, movies, or nodes in social networks. In fact, especially for large data sets, it is often more practical to express preferences as (pairwise) relative comparisons between any two objects rather than as absolute comparisons among all the objects to order (see, e.g., [111]). Suppose, for example, that you would like to rank two thousand movies you have watched, from your most preferred to your least preferred. Then, most likely it is hard for you to give an absolute ranking value to each movie. However, if you only have two movies, then it is much easier to decide which of the two movies is better by giving a binary preference (i.e., say which one is the best one among the two). Then, based on the set of pairwise preferences, one can attempt to construct a ranking of the movies which respects these preferences (see, e.g., [55]).

In general, the problem of ordering objects according to their (pairwise) similarities is a relevant topic in *data analysis*. Given a collection of objects and their pairwise similarities, the objective of data analysis is to extract information from the data such as patterns or an underlying structure, which are easier to visualize and can help a decision making process. The field of data analysis comprehends two famous techniques: classification and clustering. Both approaches aim to group together similar objects, called respectively *classes* and *clusters*. The main difference is that in classification the classes are defined in advance, while in clustering the clusters are identified later (see, e.g., [89]).

As example of structure retrieved with data analysis, consider the matrices in Figure 1.1, where each row (column) represents an object and each entry corresponds to the pairwise relative measurements among two objects. The intensity of the color in the greyscale reflects the values of the matrix. The closer to black, the higher the value. Vice versa, the closer to white, the lower the value. Note that both matrices contain the same information, but Figure 1.1b reveals two clusters, while from Figure 1.1a the same property cannot be retrieved immediately.

In this thesis we will focus on a third well known data analysis technique, which is used to sort objects according to their similarity: seriation.



(a) Original unordered matrix. (b) Ordered matrix.

Figure 1.1: Example of hidden information of the iris data set for 50 flowers from each of 3 species of the iris family (Iris setosa, versicolor and virginica) [63].

1.1 Background and motivation

The seriation problem is a combinatorial problem arising in data analysis, which asks to sequence a set of objects in such a way that similar objects are ordered close to each other. The pioneer of seriation is considered to be Flinders Petrie, an egyptologist and archaeologist who introduced seriation to sequence chronologically the tombs found in Egypt at the end of XIV century [93]. Since common dating techniques such as stratigraphy or radio carboning were not available, Petrie invented a *relative dating* procedure to sequence the tombs. Specifically, he assumed that, as we now have trends for clothes, objects, etc, also Egyptian had. Hence, he classified the potteries in the tombs according to their style, and then he tried to reorder the objects according to their similarity, with the idea that similar potteries would belong to closer periods in time. Finally, the order of the tombs was obtained by linking each tomb to the corresponding sequenced pottery element.

One can map the set of pairwise relative measurements among the objects to a (similarity) matrix A, which represents the information of the objects to order (e.g., the similarities among potteries in the example of Petrie). Then, the seriation problem can be modeled as a discrete optimization problem, where the goal is to find a permutation of the rows and columns of A which minimizes (resp., maximizes) a given loss (resp., merit) objective function, called *seriation criterion* [62] or *seriation measure* [60].

In what follows, we let $\mathcal{P}_n$ denote the set of all possible permutations of $[n] = \{1, \ldots, n\}$ and $(A_{\pi(i),\pi(j)})_{i,j=1}^n$ be the matrix obtained by symmetrically permuting the rows and columns of A according to π. A classic seriation criterion is then 2-SUM (corresponding to the seriation measure 'Inertia' in [63]), which is considered, e.g., in [58, 5, 56] and consists in solving the following program:

$$\min_{\pi \in \mathcal{P}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{\pi(i),\pi(j)} |i - j|^2. \tag{1.1}$$

An analogous measure is the '*Least Square*' criterion discussed in [24], which consists in solving the following program:

$$\max_{\pi \in \mathcal{P}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} \big(A_{\pi(i),\pi(j)} - |i - j| \big)^2. \tag{1.2}$$

The idea behind both seriation measures (1.1) and (1.2) is that, intuitively, objects with high similarity $A_{\pi(i),\pi(j)}$ are pushed close to the diagonal. Finally, another common seriation criterion is the so-called '*Measure of effectiveness*', originally defined by McCormick *et al.* [86], where the goal is to reorder A in such a way that each object has a value similar to the one of its two adjacent vertices in π, or equivalently, in such a way that each entry in A_π has similar value with respect to its four neighboring elements, i.e.,

$$\max_{\pi \in \mathcal{P}_n} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{\pi(i),\pi(j)} (A_{\pi(i),\pi(j+1)} + A_{\pi(i),\pi(j-1)} + A_{\pi(i+1),\pi(j)} + A_{\pi(i-1),\pi(j)}). \tag{1.3}$$

For a complete list of seriation measures we refer the interested reader to [63] and [60]. Seriation has many applications in a wide range of different subjects. As discussed before, it was originally introduced for applications arising in archaeology [93, 75]. It can also be used in paleontology, where the objects to be ordered are the sites of excavation and the proximity measures are mammal genera whose remains are found at these sites [85]. Furthermore, seriation can be used for data visualization and exploratory analysis [69, 18], with applications in bioinformatics (e.g., microarray gene expression [109]) and psychiatric data [27, 112].

In machine learning, seriation is used to pre-estimate the number of clusters or the tendency of data patterns to form clusters [65], with applications to text mining [46]. Further applications comprehend anthropology, cartography, database design, document processing, network analysis, ecology, linguistics, manufacturing, circuit design and ranking [1, 84, 63, 55]. For a more exhaustive and complete list of applications of seriation we refer the interested reader to the survey [81].

Seriation is conceptually similar to clustering. However, while clustering aims to order the data into groups whose members are similar to each other, seriation seeks for a linear order of the objects such that similar objects are close to each other, which is more restrictive. Hence, although related, the two problems are substantially different.

There exist several packages to solve seriation. The package 'seriation' is implemented in the open source statistical software R [1] [63] and uses different algorithms depending on the seriation measure chosen to model the seriation problem. For example, to solve 2-SUM (1.1) and Least Square (1.2), partial enumeration methods (e.g., dynamic programming [69] and branch-and-bound [18]) are used, as well as QAP's solvers (see, e.g., [19]). To solve the seriation criterion 'Measure of effectiveness' (1.3), 'Bond Energy' algorithms (see, e.g., [86, 4]) are used. The seriation measure 2-SUM (1.1) can be additionally solved via spectral methods (see [5]) and through convex relaxations (see [56]).

There also exist some packages tailored for specific applications. For example, in the domain of matrix visualization and exploratory data analysis there exist the softwares 'GAP' [2] [27], and 'SPIN' [3] [110].

The goal of seriation to order similar objects close to each other is best achieved by a special class of structured matrices, namely Robinson(ian) matrices, introduced by Robinson in [100]. Specifically, a symmetric $n \times n$ matrix A is called a *Robinson similarity* if its entries are monotone nondecreasing in the rows and columns when moving *towards* the main diagonal, i.e., if

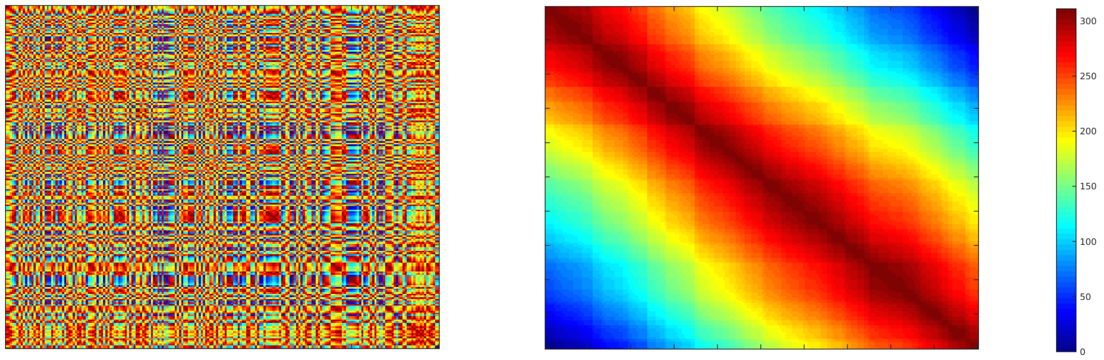
$$A_{xz} \leq \min\{A_{xy}, A_{yz}\} \quad \text{for each} \quad 1 \leq x < y < z \leq n. \tag{1.4}$$

If the rows and columns of A can be symmetrically reordered by a permutation π to get a Robinson similarity, then A is said to be a *Robinsonian similarity* and π is called a *Robinson ordering* of A. The unimodal structure of Robinsonian matrices is shown in Figure 1.2.

In the literature, a distinction is made between Robinson(ian) similarities and Robinson(ian) dissimilarities matrices. A symmetric $n \times n$ matrix D is called a *Robinson dissimilarity* if its entries are monotone nondecreasing in the rows and columns when moving *away* from the main diagonal, i.e., if $A_{xz} \geq \max\{A_{xy}, A_{yz}\}$, for each $1 \leq x < y < z \leq n$.

[1] https://cran.r-project.org/web/packages/seriation/index.html
[2] http://www.hmwu.idv.tw/gapsoftware/
[3] https://webhome.weizmann.ac.il/home/tsafriri/spin.html

The concepts of *Robinsonian dissimilarity* and Robinson ordering naturally extend to dissimilarities. In fact, any result on one class can be transferred to the other class, as A is a Robinson(ian) similarity if and only if $D = -A$ is a Robinson(ian) dissimilarity. Therefore, in the rest of the thesis we will often use the term Robinson(ian) matrix, meaning a Robinson(ian) similarity matrix.



(a) Unordered Robinsonian matrix.

(b) Ordered Robinsonian matrix.

Figure 1.2: Heatmaps representing the unimodal structure of a Robinsonian similarity matrix.

Robinsonian matrices represent an ideal seriation instance, because for any three objects $x <_\pi y <_\pi z$ in a Robinson ordering π, in view of (1.4) we have that objects x, y (which are ordered closer in π than objects x, z) are more similar to each other than objects x, z. The same holds for objects y, z. Hence, even though real world data is unlikely to have a Robinsonian structure, Robinsonian recognition algorithms can be used as core subroutines to design efficient heuristics or approximation algorithms to solve the seriation problem, e.g., by approximating the Robinsonian structure [30] (see Chapter 8 for more details). Furthermore, Robinsonian matrices can be used, e.g., in defining the *Robinson violation* seriation criterion appearing in [96, 27, 109], where the goal is to find the permutation which minimizes the number of triples violating the Robinson property (1.4), i.e.,

$$\min_{\pi \in \mathcal{P}_n} \sum_{i<j<k} g(A_{\pi(i),\pi(j)}, A_{\pi(i),\pi(k)}) + \sum_{i<j<k} g(A_{\pi(j),\pi(k)}, A_{\pi(i),\pi(k)}), \tag{1.5}$$

where:

$$g(x,y) = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{otherwise.} \end{cases}$$

Hence, the objective function in (1.5) quantifies the divergence of π from being a Robinson ordering.

Robinsonian matrices play a role also in ranking problems where, given pairwise comparisons among the items, the goal is to find a consistent ordering of the items (meaning that if item i is preferred to item j, then there exists a linear order π where $i <_\pi j$, for each i, j). Fogel *et al.* [55] show how to build such a ranking by constructing a Robinsonian similarity matrix A consistent with the pairwise comparisons. Roughly speaking, an element A_{ij} represents the number of matching comparisons between items i and j, i.e., a counter of how many times items i and j are both preferred (or, vice versa, both not preferred) with respect to a third item k, with $k \neq i, j$. Then, the authors show that a Robinson ordering of A leads to a ranking of the original items consistent with the comparisons.

The importance of Robinsonian matrices in the seriation problem and their application to other ordering problems motivate our interest for Robinsonian recognition algorithms. Therefore, Robinsonian matrices will play a fundamental role in this thesis.

1.2 Contributions

In this thesis we study the seriation problem focusing on the combinatorial structure and properties of Robinsonian matrices. Our contribution is both theoretical and practical, with a particular emphasis on algorithms.

Recognition algorithms for Robinsonian matrices As first contribution, we introduce two new algorithms to recognize Robinsonian matrices, i.e., to decide if a given matrix A is Robinsonian (and then return a Robinson ordering π of A) or not. Both algorithms are inspired by *Lexicographic Breadth-First Search* (abbreviated Lex-BFS), which is a variant of the well known graph traversal algorithm Breadth-First Search (BFS) where vertices are explored by giving preference to those vertices whose neighbors have been visited first. A central role will be played by a simple main task about sets, namely *partition refinement*, which will be repeatedly used in both recognition algorithms as core subroutine. In both cases, we introduce a new characterization of Robinsonian (similarity) matrices, which we will exploit to design new recognition algorithms. Nevertheless, the two algorithms are substantially different.

The first recognition algorithm is based on a new characterization of Robinsonian similarity matrices in relation to unit interval graphs. A *unit interval graph* $G = (V = [n], E)$ is a graph whose vertices can be mapped to unit intervals and whose edges consist of the pairs of vertices whose corresponding intervals intersect. The interesting aspect of unit interval graphs is that they coincide with the class of graphs whose (extended) adjacency matrices are 0/1 Robinsonian matrices [97]. Hence, general (not necessarily 0/1) Robinsonian matrices can be seen as a generalization of unit interval graphs to weighted graphs.

Throughout the thesis we will always assume (Robinsonian) matrices to be nonnegative. This assumption can be done without loss of generality (see Subsection 3.1.1). Let $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_L$ denote the distinct values taken by the entries of A. The graph $G^{(\ell)} = (V, E_\ell)$, whose edges are the pairs $\{x, y\}$ with $A_{xy} \geq \alpha_\ell$, is called the *ℓ-th level graph* of A, and $A_{G^{(\ell)}}$ denotes the corresponding (extended) adjacency matrix, for $\ell \in \{1, \dots, L\}$. Then, it is well known that one can decompose a given similarity matrix A as a conic combination of 0/1 matrices, i.e.,

$$A = \alpha_0 J + \sum_{\ell=1}^{L} (\alpha_\ell - \alpha_{\ell-1})\, A_{G^{(\ell)}}.$$

Our main contribution is the introduction of a new characterization of the level graphs $G^{(1)}, \dots, G^{(L)}$ in the above decomposition when A is a Robinsonian (similarity) matrix, using the concept of straight enumerations.

Given a vertex x of a graph $G = (V, E)$, the closed neighborhood of x is the subset of V consisting of the adjacent vertices of x and x itself. A *straight enumeration* is then a special ordered partition $\phi = (B_1, \dots, B_p)$ of V, where each block B_i is a subset of vertices having the same closed neighborhood and with the property, roughly speaking, that consecutive blocks induce a clique in G (for a formal definition see Definition 2.3.4). The reason we are interested in straight enumerations is that a graph is a unit interval graph if and only if it has a straight enumeration (see [43, 67]). In fact, straight enumerations of a unit interval graph G embed all the Robinson orderings of the (extended) adjacency matrix A_G. Roughly speaking, given a unit interval graph $G = (V, E)$ and a straight enumeration $\phi = (B_1, \dots, B_p)$ of V, any linear order π of V obtained by rearranging the elements within each block B_i of ϕ induces a Robinson ordering for A_G. We then say that such π is *compatible* with ϕ. Our main result is that A is Robinsonian if and only if there exists a linear order π of V and straight enumerations $\phi_1, \dots, \phi_L$ of its level graphs $G^{(1)}, \dots, G^{(L)}$ such that π is compatible with ϕ_i for all $i = 1, \dots, L$. Since straight enumerations can be found in linear time using Lex-BFS, this motivated us to introduce a new Lex-BFS based polynomial-time algorithm to recognize Robinsonian matrices.

Roughly speaking, our algorithm computes straight enumerations of the level graphs and it returns a linear order π compatible with all of them (if one exists). However, instead of refining the level graphs one by one on the full set V, we use a recursive algorithm based on a divide-and-conquer strategy, which splits the problem in subproblems according to the connected components of the level graphs, to exploit the sparsity of the matrix. In addition, since the Robinson ordering might not be unique, we also show how to modify our algorithm to return all the possible Robinson orderings of a given Robinsonian matrix.

Our algorithm uncovers an interesting link between (straight enumerations of) unit interval graphs and Robinsonian matrices which, to the best of our knowledge, has not been observed before, as Robinsonian matrices have been mainly studied in relation to interval (hyper)graphs. Moreover it provides an answer to an open question posed by M. Habib at the PRIMA Conference in Shanghai in June 2013, who wondered about the possibility of using Lex-BFS to recognize Robinsonian matrices (see [39]).

The second recognition algorithm is based on novel combinatorial properties of Robinsonian similarity matrices, derived in particular by introducing the concept of '*path avoiding a vertex*' (see Definition 6.2.2) and by characterizing the end points of Robinson orderings (called '*anchors*'). As mentioned before, Robinsonian similarities matrices can be seen as an extension of unit interval graphs to weighted graphs. Then, the motivation for our second approach is to extend some of the immense work present in the literature for unit interval graphs to Robinsonian similarity matrices. In particular, since unit interval graphs can be recognized with a simple Lex-BFS algorithm [33], we introduce a novel algorithm, named *Similarity-First Search* (SFS), which represents a generalization of the classical Lex-BFS algorithm to weighted graphs. Intuitively, the SFS algorithm traverses vertices of a weighted graph in such a way that most similar vertices (i.e., corresponding to largest edge weights) are visited first, while still respecting the priorities imposed by previously visited vertices. By definition, the SFS algorithm reduces to Lex-BFS when applied to an unweighted graph. As we will show, the SFS algorithm captures the Robinsonian structure, and it will play a central role in our second Robinsonian recognition algorithm.

In fact, we introduce a multisweep algorithm, where we compute repeated SFS iterations, each named a *sweep*, which uses the order returned by the previous sweep to break ties in the (weighted) graph search. Our main result is that our multisweep algorithm can recognize in polynomial time after at most $n-1$ sweeps whether a given $n \times n$ matrix A is Robinsonian. Our algorithm represents the first multisweep search algorithm for weighted graphs (while multisweep algorithms for unweighted graphs are well studied). We consider this algorithm to be the simplest combinatorial Robinsonian recognition algorithm in the literature (both conceptually and to implement), although it is not proven to be the best one complexity wise (see Table 3.6). In addition, we introduce some concepts extending analogous notions in graphs and we develop some combinatorial tools for the study of Robinsonian similarities matrices that could be useful to further characterize the Robinsonian structure, e.g., defining a certificate for non-Robinsonian similarity matrices.

From a practical point of view, we implemented in C++ the new recognition algorithms. Both algorithms perform well, recognizing 1000×1000 dense Robinsonian matrices in few seconds (or even milliseconds). Therefore, the code developed could be potentially included in some software libraries (e.g., R) to make it available for the scientific community to solve real world problems.

QAP and Robinsonian approximation The second contribution of this thesis is related to solving the seriation problem when the similarity matrix A is not Robinsonian.

Our first result is theoretical and motivates our interest in Robinsonian matrices recognition algorithms. Following [71], we model seriation as a special case of the quadratic assignment problem (QAP). Given $n \times n$ symmetric matrices A and B, QAP is the following optimization problem [76]:

$$\min_{\pi \in \mathcal{P}_n} \sum_{i,j=1}^{n} A_{\pi(i),\pi(j)} B_{ij}. \tag{1.6}$$

As we have seen in Section 1.1, a possible choice of the matrix B is the one appearing in the 2-SUM problem (1.1), where $B_{ij} = |i-j|^2$.

QAP is a hard problem which has been extensively studied in the literature (see [23] and references therein). Recently, many authors have focused on some special cases which are solvable in polynomial time by exploiting the structure of the matrices A, B. In our thesis we introduce a new instance of polynomially solvable QAP related to Robinsonian matrices and seriation. Specifically, a symmetric matrix B is *Toeplitz* if it has constant values on its diagonals, i.e., $B_{ij} = B_{i+1,j+1}$, for each $1 \leq i, j \leq n-1$. Our main result is the following: if A is a Robinson similarity matrix and B is a Toeplitz Robinson dissimilarity matrix, then the identity permutation is optimal for (1.6). This result generalizes two results in the literature, namely in [56] and [31] (see Subsection 7.2.4 for more details). Moreover, as an application, if A is a Robinsonian similarity matrix then any Robinson ordering π of A optimally solves (1.1). This also motivates our interest in recognizing Robinsonian matrices.

As a second result, we introduce a new heuristic to solve seriation, based on a generalization of the SFS algorithm. Specifically, given a symmetric matrix A, we are interested in finding a 'close' Robinsonian approximation A^R of A. Following previous works in [29, 30], we relax the notion of Robinsonian matrix: for a fixed real $\epsilon > 0$, we consider the concept of ϵ-Robinsonian similarity, defined as a matrix whose rows and columns can be reordered in such a way that the following relaxed version of (1.4) holds:

$$A_{xz} \leq \min\{A_{xy}, A_{yz}\} + \epsilon \quad \text{for each} \quad 1 \leq x < y < z \leq n.$$

Then, the output of our research is a simple extension of the SFS algorithm, named ϵ-SFS, to the recognition of ϵ-Robinsonian similarities. However, differently from the above two algorithms, this is a heuristic and represents a preliminary work. The main reason we discuss it is to show that the SFS algorithm could be easily modified and potentially applied to other classes of matrices (not necessarily having a Robinson structure).

1.3 Outline of the thesis

The results in this thesis are organized in three parts. The first part (Chapters 2-4) contains preliminaries and basic concepts related to Robinsonian matrices and Lex-BFS, which as we have seen above play a central role in this thesis. The second part (Chapters 5-6) is focused on the description of the two new recognition algorithms for Robinsonian matrices. Finally, in the third part (Chapters 7-9) we show a new instance of QAP involving Robinsonian matrices that admit an optimal solution in closed form and we introduce a heuristic to solve the seriation problem by approximating the Robinsonian structure. We conclude the thesis with giving some computational experiments.

Part I. Robinsonian matrices and seriation

Chapter 2. Preliminaries In this chapter we introduce some concepts, notation and important facts recurrent in the whole thesis. In particular, we define the concepts of proximity matrix, permutation, linear order, weak linear order, PQ-trees, and we discuss a basic algorithm about sets, namely partition refinement, which plays an important role in the recognition algorithms for Robinsonian matrices. We also introduce some basic concepts in graph theory and discuss the graph classes of chordal, interval (hyper)graphs and unit interval graphs. For the latter graphs, we introduce the important concept of straight enumerations.

Chapter 3. Robinsonian matrices In this chapter we present Robinsonian matrices, giving a formal definition and outlining their applications in some important combinatorial and classification problems. We discuss the main known characterizations of Robinsonian matrices in terms of interval, unit interval graphs and interval hypergraphs, and we discuss the combinatorial and spectral recognition algorithms existing in the literature.

Chapter 4. Lexicographic Breadth-First Search In this chapter we discuss Lexicographic Breadth-First search (Lex-BFS), which is a special graph traversal algorithm used for the recognition of several classes of graphs and Robinsonian matrices. We present in detail how Lex-BFS and its variant Lex-BFS$_+$ work and discuss their linear time implementation using the partition refinement paradigm introduced in Chapter 2. Finally, we show their use in multisweep algorithms, and we discuss some structural properties of Lex-BFS when applied to special graph classes.

Part II. Recognition algorithms for Robinsonian matrices

Chapter 5. Lex-BFS based algorithm In this chapter we introduce a new Lex-BFS based algorithm to recognize Robinsonian matrices. First we discuss a new characterization of Robinsonian matrices in terms of straight enumerations of unit interval graphs. Based on this characterization, we present the main subroutines constituting the new Robinsonian recognition algorithm. In the last part of the chapter we show how to extend the above recognition algorithm to return all the Robinson orderings of a given Robinsonian similarity matrix using the PQ-Tree structure presented in Chapter 2. The content of this chapter is based on our work [77].

Chapter 6. Similarity-First Search In this chapter we introduce the novel Similarity-First Search algorithm (SFS) and its application to the recognition of Robinsonian matrices. We introduce the fundamental concepts of 'path avoiding a vertex', valid vertex and anchors, which we will use to provide new properties for Robinsonian matrices. We then describe the SFS algorithm, we show many properties when applied to Robinsonian matrices, and we present the multisweep recognition algorithm. The content of the chapter is based on our work [79].

Part III. QAP and Robinsonian approximation

Chapter 7. Seriation and the quadratic assignment problem In this chapter we model the seriation problem as an instance of QAP as in (1.6). We show that if both matrices A and B are Robinsonian, one can find an explicit solution to QAP by using a Robinsonian recognition algorithm to find Robinson orderings of A and B. The content of the chapter is based on our work [78].

Chapter 8. Robinsonian matrix approximation In this chapter we discuss how to solve the seriation problem by finding a Robinsonian approximation of the original proximity matrix. We define the l_∞-FITTING-BY-ROBINSONIAN problem and the ϵ-ROBINSONIAN-RECOGNITION problem. Then, we introduce the ϵ-Similarity-First Search algorithm (ϵ-SFS), an extension of the SFS algorithm presented in Chapter 6, and we discuss a multisweep algorithm based on ϵ-SFS aiming to recognize ϵ-Robinsonian matrices.

Chapter 9. Computational experiments In this final chapter we discuss some experiments which we have carried out in order to compare our algorithms introduced in Chapters 5, 6 and 8, and the spectral algorithm presented in Chapter 3. We explain how we designed the experiments and discuss the performance of the aforementioned algorithms depending on the structure of the input matrices (e.g., density and/or number of distinct values).

Publications and preprints

The content of this thesis is based on the following publications and preprints:

- M. Laurent and M. Seminaroti. The quadratic assignment problem is easy for Robinsonian matrices with Toeplitz structure. *Operations Research Letters*, 43(1):103–109, 2015.
- M. Laurent and M. Seminaroti. A Lex-BFS-based recognition algorithm for Robinsonian matrices. In *Algorithms and Complexity: Proceedings of the 9th International Conference CIAC 2015*, volume 9079 of *Lecture Notes in Computer Science*, pages 325–338. Springer-Verlag, 2015.
  The extended version is currently under review in *Discrete Applied Mathematics.*
  Available at: *arXiv:1504.06586.*
- M. Laurent and M. Seminaroti. Similarity-First Search: a new algorithm with application to Robinsonian matrix recognition, 2016.
  Currently under revision in *SIAM Journal on Discrete Mathematics.*
  Available at: *arXiv:1601.03521.*

Part I

Robinsonian matrices and Lex-BFS

2

Preliminaries

In this chapter we introduce some notations and important facts used throughout the thesis. Whenever necessary, they will be reintroduced in the specific chapters to help the reading of the manuscript. In Section 2.1 we introduce the concept of proximity matrix. In Section 2.2 we introduce the concepts of permutations, linear orders, weak linear orders, PQ-trees, and we discuss a basic operation about sets, namely partition refinement. Finally, in Section 2.3 we introduce some basic concepts in graph theory and we discuss the classes of chordal, interval, unit interval graphs, and of interval hypergraphs.

2.1 Proximity matrices

In this thesis we deal with *proximity* matrices, a common term in the literature to refer to any numerical measure between elements or objects mapped into a matrix. Specifically, rows and columns represent a set of objects, while the numerical measure in each entry represents the similarity or dissimilarity information between the objects (i.e., how similar or dissimilar two objects are). The main difference between similarity and dissimilarity matrices relies on the fact that similarities have the largest values on the main diagonal (since the similarity between an element and itself is maximum), while dissimilarities have the smallest value on the main diagonal (because the dissimilarity between an element and itself is minimum). To help distinguishing between these two classes, throughout the thesis we will use the symbol A to denote similarity matrices and the symbol D to denote dissimilarity matrices.

An example of similarity matrix is a correlation matrix $A = (A_{ij}) \in \mathbb{R}^{n\times n}$, where we have n random variables and each entry A_{ij} represents the correlation between the ith and jth random variables. Specifically, each entry A_{ij} assumes any value in the interval $[-1, 1]$. Values close to 1 (-1) indicate a strong (inverse) correlation and an entry is equal to zero if the corresponding variables are independent. In this case, the elements on the main diagonal are always equal to one (i.e., the largest possible value), as they represent the correlation of a random variable with itself.

An example of dissimilarity matrix is the distance matrix of a finite *metric*, i.e., a symmetric matrix $D = (D_{ij}) \in \mathbb{R}^{n\times n}$ satisfying the following properties for each $1 \le i, j, k \le n$: *(i)* $D_{ij} \ge 0$ and $D_{ij} = 0$ if and only if $i = j$ (nonnegativity); *(ii)* $D_{ik} \le D_{ij} + D_{jk}$ (triangle inequality). A Euclidean distance matrix is a classic example of distance matrix, defined as follows. We are given n (distinct) points $x_1, \dots, x_n \in \mathbb{R}^m$ and the entries of D are defined by $D_{ij} = \|x_i - x_j\|_2^2$, for all $i, j = 1, \dots, n$. For $m = 1$, the n elements are points on a line and we obtain a special distance matrix which is a Robinson dissimilarity matrix.

Given a proximity matrix, we are interested in reordering its rows and columns to retrieve some special hidden structure of the data. An example of such structure is represented, e.g., by 0/1 matrices with the so-called consecutive ones property, defined as follows.

2.1.1 Definition. (**C1P**) A 0/1 matrix has the *consecutive ones property* (C1P) if its columns can be reordered in such a way that the ones are consecutive in each row. Moreover, a 0/1 matrix has the *symmetric consecutive ones property* if its rows and columns can be symmetrically reordered in such a way that the ones are consecutive in each row and column.

As we will see in Subsection 3.2.1, matrices with C1P are related to Robinsonian matrices, which play an important role in the seriation problem and a central role in this thesis. For more details about matrices with the consecutive ones property we refer the interested reader to [47].

In Parts II and III of the thesis we will introduce many different algorithms to retrieve a special hidden structures of the data, namely the Robinson structure introduced in Section 1.1.

In the rest of the thesis, we denote by $\mathcal{S}^n$ the set of symmetric $n \times n$ matrices. Then, we consider the finite set $[n] = \{1, \dots, n\}$ as index set for the rows (columns) of a given matrix $A \in \mathcal{S}^n$. We will often denote the set $[n]$ by V, and V will sometimes represent the set of vertices of a graph. For this reason, elements of V are often also called *vertices*. Furthermore, we denote by $e \in \mathbb{1}^{n\times 1}$ the all-ones vector and by $J_n = ee^\mathsf{T}$ the all-ones matrix. For $A, B \in \mathbb{R}^{n\times n}$, $\langle A, B\rangle = \mathrm{Tr}(A^\mathsf{T} B) = \sum_{i,j=1}^n A_{ij}B_{ij}$ denotes the trace inner product on $\mathbb{R}^{n\times n}$. For $U \subseteq [n]$, $A[U] = (A_{ij})_{i,j\in U}$ is the principal submatrix of A indexed by U.

2.2 Orderings and partition refinement

In this section we introduce the concepts of permutations (Subsection 2.2.1), linear orders and weak linear orders (Subsection 2.2.2), which we will repeatedly use in the next chapters. Furthermore, we discuss the basic operation of partition refinement (Subsection 2.2.3), which will play an important role in this thesis. Finally, we introduce the concept of PQ-tree (Subsection 2.2.4).

2.2.1 Permutations

Given a finite set $[n] = \{1, \ldots, n\}$ of elements, called *ground set*, a permutation π of $[n]$ is a one-to-one mapping from $[n]$ to itself. There exist many equivalent ways to write a permutation. In this thesis, we will represent a permutation of $[n]$ by a vector $\pi \in \mathbb{N}^n$ listing the objects in the order they appear in π. Hence the permutation π is represented by the sequence $(x_1, \ldots, x_n)$, where $\pi(x_i) = i$ for $i \in [n]$. For example, consider the permutation $\pi = (2, 4, 1, 3, 5)$. Then $\pi(2) = 1$, $\pi(4) = 2$ and so on. Hence, $(1, \ldots, n)$ represents the *identity permutation.*

Every permutation π of the set $[n]$ corresponds in a unique way to an $n \times n$ *permutation matrix* $\Pi \in \{0, 1\}^{n \times n}$, where the generic element Π_{ij} is equal to:

$$\Pi_{ij} = \begin{cases} 1 & \text{if } \pi(i) = j, \\ 0 & \text{else.} \end{cases}$$

For this reason, in the following we will use the notation π and Π indiscriminately. We let $\mathcal{P}_n$ denote the set of all possible permutations of $[n]$ as well as the set of all $n \times n$ permutations matrices.

Then, for a matrix $A \in \mathbb{R}^{n \times n}$, the matrix ΠA results in a row-permutation of A, with elements $(\Pi A)_{ij} = A_{\pi(i)j}$, while $A\Pi^{\mathsf{T}}$ results in a column-permutation of A, with elements $(A\Pi^{\mathsf{T}})_{ij} = A_{i\pi(j)}$. Hence, $\Pi A \Pi^{\mathsf{T}} = (A_{\pi(i),\pi(j)})_{i,j=1}^{n}$, sometimes also denoted by A_π, is the matrix obtained by symmetrically permuting the rows and columns of A. We say that A is ordered according to π, when its rows and columns are ordered according to π.

Using the above notation, we can formalize the definition of C1P matrix (see Definition 2.1.1). Specifically, a matrix $A \in \{0, 1\}^{m \times n}$ has C1P if there exists a permutation matrix $\Pi \in \{0, 1\}^{n \times n}$ such that $A\Pi^{\mathsf{T}}$ has consecutive ones in each row. Analogously, a symmetric matrix $A \in \{0, 1\}^{n \times n}$ has symmetric C1P if there exists a permutation matrix $\Pi \in \{0, 1\}^{n \times n}$ such that $\Pi A \Pi^{\mathsf{T}}$ has consecutive ones in each row and column.

A useful property of permutations we will use in Chapter 7 is that, for any $A, B \in \mathcal{S}^n$ and $\pi, \tau \in \mathcal{P}_n$, we have:

$$(A_\pi)_\tau = A_{\pi\tau} \quad \text{and} \quad \langle A, B_\tau \rangle = \langle A_{\tau^{-1}}, B \rangle. \tag{2.1}$$

The $n \times n$ permutation matrices are all the possible binary solutions to the following linear system:

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1 \qquad i,j \in [n], \\ \sum_{j=1}^{n} x_{ij} &= 1 \qquad i,j \in [n], \\ x_{ij} &\geq 0 \qquad i,j \in [n]. \end{aligned} \tag{2.2}$$

The linear system (2.2) defines the set of doubly stochastic matrices, denoted by $\mathcal{D}_n$. The set $\mathcal{D}_n$ is a bounded polyhedron [1], therefore a polytope, and it is also known as the Birkhoff polytope. In fact, as the following classical result shows, $\mathcal{D}_n$ is equal to the convex hull of the set of permutation matrices $\mathcal{P}_n$.

2.2.1 Theorem. *(**Birkhoff–von Neumann**)[13] Every doubly stochastic matrix $A \in \mathcal{D}_n$ is a convex combination of permutation matrices $\Pi_1, \dots, \Pi_k \in \mathcal{P}_n$, i.e., $A = \lambda_1 \Pi_1 + \cdots + \lambda_k \Pi_k$ with $\lambda_1, \dots, \lambda_k \in [0,1]$ and $\sum_{i=1}^{k} \lambda_i = 1$.*

Moreover, each permutation matrix is a vertex of $\mathcal{D}_n$ (see, e.g., [105]). This result is used, e.g., to define simple convex relaxations for the seriation problem (see [56]).

2.2.2 (Weak) linear orders

A *linear order* (or *total order*) on $V = [n]$ is a relation $\leq$ on $[n]$ satisfying the following properties for any three elements $x, y, z \in [n]$: *(i)* $x \leq x$ (reflexivity); *(ii)* if $x \leq y$ and $y \leq x$ then $x = y$ (antisymmetry); *(iii)* if $x \leq y$ and $y \leq z$ then $x \leq z$ (transitivity); *(iv)* either $x \leq y$ or $y \leq x$ (comparability).

A permutation induces a linear order, denoted by π or $<_\pi$, where we write $i <_\pi j$ meaning that i comes before j after reordering the elements according to π, i.e., $\pi(i) < \pi(j)$. As for permutations, it will be convenient to represent a linear order π as a sequence $(x_1, \dots, x_n)$ with $x_1 <_\pi \dots <_\pi x_n$ and $[n] = \{x_1, \dots, x_n\}$. Hence, the permutation $\pi = (x_1, \dots, x_n)$ corresponds to the linear order $x_1 <_\pi x_2 <_\pi \cdots <_\pi x_{n-1} <_\pi x_n$. Moreover, the *reversal* of π, denoted by $\overline{\pi}$, is the linear order $(x_n, x_{n-1}, \dots, x_1)$.

For $U \subseteq [n]$, $\pi[U]$ denotes the *restriction* of the linear order π to the subset U. Given disjoint subsets $U, W \subseteq [n]$, we say $U <_\pi W$ if $x <_\pi y$ for all $x \in U$, $y \in W$. Furthermore, if π_1 and π_2 are two linear orders on disjoint subsets V_1 and V_2, then $\pi = (\pi_1, \pi_2)$ denotes their *concatenation*, which is a linear order on $V_1 \cup V_2$.

Since a linear order uniquely represents a permutation, we will use the two terms indiscriminately in this thesis. Sometimes, we will also use the term *ordering* to denote a linear order.

[1]A polyhedron is the set of solutions of a system of linear inequalities.

We introduce a particular linear order which will be used in Chapter 4. Given a finite set $[n]$, called *alphabet*, and whose elements are called *letters*, a *word* is a sequence of letters. Then the *lexicographic order* (also known as *lexical order*, *dictionary order* or *alphabetical order*) is a particular linear order π on the set of words defined as follows. Given two words $x = (x_1, \dots, x_p)$ and $y = (y_1, \dots, y_r)$, then $x \leq_\pi y$ if there exists an index $j \geq 1$ such that $x_j > y_j$ and $x_i = y_i$ for all $i < j$. If $p \neq r$, then the shortest word is completed to a word of length $\max\{p, r\}$ by adding the letter $\varnothing$, which is considered the smallest letter of the alphabet. For example, we have that $(5, 3, 4) \leq_\pi (5, 3) \leq_\pi (5, 2, 1)$.

A *weak linear order* (or *partial order*) on V is a relaxation of a linear order, i.e., a relation on V satisfying only the reflexivity, antisymmetry and transitivity properties (from which the adjective weak comes from).

An *ordered partition* of the ground set V is an ordered sequence of pairwise disjoint subsets of V (each called *class* or *block*) whose union is equal to V.

Then, an ordered partition induces a weak liner order on V, which is denoted by $\psi = (B_1, \dots, B_k)$ or $B_1 <_\psi \dots <_\psi B_k$, and is obtained by setting $x =_\psi y$ if x, y belong to the same class B_i, and $x <_\psi y$ if $x \in B_i$ and $y \in B_j$ with $i < j$. Hence, we write $x \leq_\psi y$ if $x \in B_i, y \in B_j$ with $i \leq j$.

The *reversal* of ψ, denoted $\overline{\psi}$, is the weak linear order of the reversed ordered partition $(B_k, \dots, B_1)$. For $U \subseteq V$, $\psi[U] = (B_1 \cap U, \dots, B_k \cap U)$ denotes the *restriction* of the weak linear order ψ to U (keeping only nonempty intersections). Given disjoint subsets $U, W \subseteq V$, we say $U \leq_\psi W$ if $x \leq_\psi y$ for all $x \in U$, $y \in W$. If ψ_1 and ψ_2 are weak linear orders on disjoint sets V_1 and V_2, then $\psi = (\psi_1, \psi_2)$ denotes their *concatenation* which is a weak linear order on $V_1 \cup V_2$.

When all classes B_i are singletons then ψ reduces to a linear order (i.e., total order) of V. As for linear orders, since a weak liner order uniquely represents an ordered partition, we will use the two terms indiscriminately in this thesis.

The following notions of compatibility and refinement will play an important role in our discussion. A linear order π of $[n]$ is *compatible* with a weak linear order $\psi = (B_1, \dots, B_k)$ if $x <_\pi y$ implies that $x \in B_i, y \in B_j$ with $i \leq j$.

Two weak linear orders ψ_1 and ψ_2 on the same set $[n]$ are then said to be *compatible* if there exists a linear order π of $[n]$ which is compatible with both ψ_1 and ψ_2, i.e., there do not exist elements $x, y \in V$ such that $x <_{\psi_1} y$ and $y <_{\psi_2} x$. Then, their *common refinement* is the weak linear order $\Psi = \psi_1 \wedge \psi_2$ on V defined by $x =_\Psi y$ if $x =_{\psi_\ell} y$ for all $\ell \in \{1, 2\}$, and $x <_\Psi y$ if $x \leq_{\psi_\ell} y$ for all $\ell \in \{1, 2\}$ with at least one strict inequality. Consider, for example, the weak linear orders $\psi_1 = (\{1, 2, 3\}, \{4, 5\}, \{6\})$ and $\psi_2 = (\{1, 3\}, \{2, 5\}, \{4, 6\})$. Then ψ_1 and ψ_2 are compatible, and their common refinement is $\Phi = (\{1, 3\}, \{2\}, \{5\}, \{4\}, \{6\})$. If we modify the second weak linear order to $\psi_2 = (\{1, 5\}, \{2, 3\}, \{4, 6\})$, then ψ_1 and ψ_2 are not compatible anymore, as $2 <_{\psi_1} 5$ and $5 <_{\psi_2} 2$.

In Chapter 5 we will use the following fact, whose easy proof is omitted.

2.2.2 Lemma. *Let $\psi_1, \dots, \psi_L$ be weak linear orders on V. Then $\psi_1, \dots, \psi_L$ are pairwise compatible if and only if there exists a linear order π on V which is compatible with each of $\psi_1, \dots, \psi_L$, in which case π is compatible with their common refinement $\psi_1 \wedge \cdots \wedge \psi_L$.*

2.2.3 Partition refinement

Partition refinement is a basic algorithm introduced in [92] about sets, which we will use repeatedly as a basic ingredient in the algorithms presented in Chapters 4, 5 and 6.

Algorithm 2.1: Partition refinement (ψ, S)

```
input: an ordered partition $\psi = (B_1, \dots, B_k)$ of set $V$ and a subset $S \subseteq V$
output: a (refined) ordered partition $\psi' = (B'_1, \dots, B'_h)$

1 foreach $B_i \in \psi$ do
2     Let $X = B_i \cap S$
3     if $|X| \neq \emptyset$ and $X \neq B_i$ then
4         remove $X$ from $B_i$ and insert it immediately before $B_i$ in $\psi$
5 Let $\psi'$ the refined ordered partition
6 return $\psi'$
```

Algorithm 2.1 is based on the work presented by Habib *et al.* [61] and goes as follows. We are given an ordered partition $\psi = (B_1, \dots, B_k)$ of V and a subset $S \subseteq V$. Then, refining ψ with respect to S produces a new ordered partition $\psi' = (B'_1, \dots, B'_h)$ of V obtained by splitting each class B_i of ψ into the intersection $B_i \cap S$ and the difference $B_i \setminus S$. Equivalently, each class B_i of ψ is replaced by the sequence $(B_i \cap S, B_i \setminus S)$, keeping only nonempty classes (see Figure 2.1). Hence, by construction, for each block $B'_i \in \psi'$ either $B'_i \subseteq S$ or $B'_i \cap S = \emptyset$ holds.

As we will see in Chapter 4, the weak linear order ψ will represent the priority queue of a graph traversal algorithm. Most of the algorithms developed in Chapters 5 and 6 will be inspired by the refinement framework of Algorithm 2.1 mainly due to its simplicity and its efficient implementation as stated below.

2.2.3 Theorem. *[61] Given an ordered partition ψ of V and a subset $S \subseteq V$, Algorithm 2.1 can be implemented in $O(|S|)$ time.*

Proof. We assume that the ordered partition ψ is stored in a doubly linked list, whose elements are the classes $B_1, \dots, B_k$. Moreover each element of V has a pointer to the class B_i containing it as well as a pointer to its position in the class B_i, which are updated throughout the algorithm. This data structure

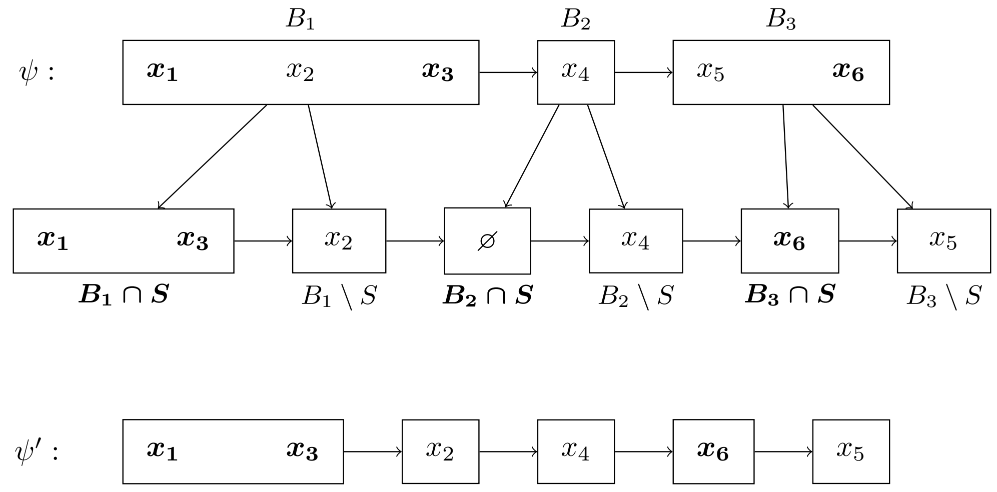

B_1
B_2
B_3
ψ :
$\boldsymbol{x_1}$ x_2 $\boldsymbol{x_3}$
x_4
x_5 $\boldsymbol{x_6}$
$\boldsymbol{x_1}$ $\boldsymbol{x_3}$
x_2
$\varnothing$
x_4
$\boldsymbol{x_6}$
x_5
$\boldsymbol{B_1 \cap S}$
$B_1 \setminus S$
$\boldsymbol{B_2 \cap S}$
$B_2 \setminus S$
$\boldsymbol{B_3 \cap S}$
$B_3 \setminus S$
ψ' :
$\boldsymbol{x_1}$ $\boldsymbol{x_3}$
x_2
x_4
$\boldsymbol{x_6}$
x_5

Figure 2.1: Example of the partition refinement of $\psi = (B_1, \dots, B_k)$ with respect to the subset $S = \{x_1, x_3, x_6\}$ (in bold).

permits constant time insertion and deletion of an element in a class of ψ. The main task of the refinement process is to split each class B_i in the intersection $B_i \cap S$ and the difference $B_i \setminus S$. This is equivalent to remove $B_i \cap S$ from B_i and place it in a new block immediately before B_i in ψ. To do so, we use an additional counter n_i for each block B_i which will denote the size of the intersection of $B_i \cap S$ and it is initialized equal to zero. Then, for each $x \in S$ with (say) $x \in B_i$, we remove it from its current position in B_i and we place it in position $n_i + 1$ in B_i (which can be done in $O(1)$ time as each vertex has a pointer to the position in the class containing it) and we increase n_i by one. After all the elements of S have been visited, in order to remove the subset $B_i \cap S$ from B_i we simply remove the first $n_i = |B_i \cap N(p)|$ elements of B_i, and we push them in a new block B_i' which we insert immediately before B_i in ψ. Once a vertex is relocated in ψ, its pointers to the corresponding block and position in ψ are updated accordingly. Therefore, removing the elements of $B_i \cap S$ from B_i can be done in $O(|B_i \cap S|) \leq O(|S|)$. Since we need to pass through the elements of S at least once, which requires $O(|S|)$, we get an overall complexity of $O(|S|)$ time. Note that if both S and ψ are ordered according to the same linear order τ, then such order in the new block B_i' is preserved. □

2.2.4 PQ-trees

A compact way to represent a set of permutations is using a special data structure, named *PQ-tree*. A PQ-tree $\mathcal{T}$ is a special rooted ordered tree. The leaves are in one-to-one correspondence with the elements of the groundset $[n]$ and their order gives a linear order of $[n]$. The nodes of $\mathcal{T}$ can be of two types, depending on how their children can be ordered. Namely, for a *P-node* (represented by a circle), its children may be arbitrary reordered; for a *Q-node* (represented by a rectangle), only the order of its children may be reversed. Moreover, every node has at least two children. Consider the example illustrated in Figure 2.2. Then, the set of permutations represented by $\mathcal{T}$ is $(1,2,3,4,5), (1,2,3,5,4)$ and $(1,2,4,3,5), (1,2,4,5,3), (1,2,5,3,4), (1,2,5,4,3)$ and their reversals.

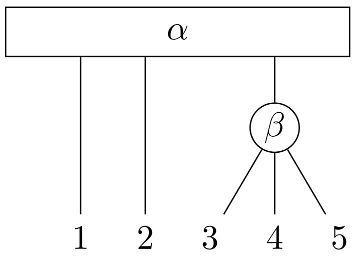


Figure 2.2: PQ-tree $\mathcal{T}$: α is a Q-node, while β is a P-node.

PQ-trees were used by Booth and Leuker [14] for the recognition of 0/1 matrices with C1P introduced earlier in Section 2.1. As we will see in Chapter 3, the recognition of C1P matrices using PQ-trees plays an important role in the recognition of Robinsonian matrices.

2.3 Graphs

In what follows $V = [n]$ is the vertex set of a graph $G = (V, E)$, whose edges are pairs $\{x, y\}$ of distinct vertices $x, y \in V$, and $|E| = m$. For $x \in V$, the neighborhood $N(x)$ of x is the set of adjacent vertices to x, i.e., $N(x) = \{y \in V : \{x, y\} \in E\}$. A graph is complete if any two vertices are adjacent. Given a subset $V' \subseteq V$, the *induced subgraph* $G' = (V', E')$ of G is the graph whose set of vertices is V' and whose edges are the pairs $\{x, y\} \in E$ with $x, y \in V'$. A *clique* of G induces a complete subgraph of G. For $x \in V$, the *closed neighborhood* is the set $N[x] = \{x\} \cup N(x)$. Two vertices $x, y \in V$ are *undistinguishable* if $N[x] = N[y]$ (see [35]). This undistinguishability defines an equivalence relation on V, whose classes are called the *blocks* of G. Clearly, each block is a clique of G. Two distinct blocks B and B' are said to be *adjacent* if there exist two vertices $x \in B$, $y \in B'$ that are adjacent in G or, equivalently, if $B \cup B'$ is a clique of G.

There exist two equivalent representations of a graph: through its adjacency matrix or its adjacency list. The *adjacency matrix* $A = (A_{xy})$ is the $n \times n$ matrix whose rows and columns are indexed by the vertices of the graph and with entry $A_{xy} = 1$ if there exists an edge between vertices x and y, and $A_{xy} = 0$ otherwise. Sometimes we will also consider the *extended adjacency matrix*, which is obtained by setting the diagonal entries to one in the adjacency matrix. Independently of the number of edges, the adjacency matrix requires $O(n^2)$ memory, and permits $O(1)$ time access to a specific edge $\{x, y\}$ of the graph. Therefore, it is suitable when the given graph is dense, i.e., m is comparable with n^2, and when several accesses to edges are required.

The *adjacency list* consists instead of an array of n lists. Each element x of the array represents a vertex of the graph, and the corresponding list represents its neighborhood $N(x)$. The adjacency list requires $O(n+m)$ memory, as only the existing edges in the graph are stored. However, it requires $O(|N(x)|)$ time to access a specific edge $\{x,y\}$, as in the worst case one has to pass through the whole list of neighbors of x. Therefore, it is suitable when the given graph is sparse, i.e., m is much smaller than n^2, and when several explorations of the neighborhoods are required.

Both of the above representations can be used for directed and undirected graphs, and they can be easily extended for weighted graphs. In this thesis we will use adjacency lists to represent the graphs, as the main routines involve the exploration of the neighborhoods of the vertices. Nevertheless, we will use the adjacency matrix to illustrate a graph in the figures.

We now discuss some graph classes, namely chordal, interval, unit interval graphs and interval hypergraphs, whose concepts will be used especially in Chapters 3, 4 and 5 in relation with Robinsonian matrices. For each graph class, we briefly give its definition, characterizations and recognition algorithms.

Chordal graphs A graph $G=(V,E)$ is a *chordal graph* if it does not contain an induced cycle of length four or more. Equivalently, every cycle of more than three vertices has a *chord*, i.e., an edge not in the cycle but connecting two vertices of the cycle. Chordal graphs can be characterized using perfect elimination orderings. Specifically, a vertex is *simplicial* if its neighborhood is a clique. Then, an ordering $\pi=(x_1,\dots,x_n)$ of V is a *perfect elimination ordering* (PEO) if x_i is simplicial in the subgraph induced on $x_1,\dots,x_i$ for each $i\in[n]$. The following result holds.

2.3.1 Theorem. *[101] G is a chordal graph if and only if it admits a PEO.*

As we will see in Subsection 4.3.1, chordal graphs can be recognized in linear time using lexicographic breadth-first search (Lex-BFS), which in fact produces a perfect elimination ordering of the graph (if it is chordal) [101].

Interval graphs Interval graphs are a subclass of chordal graphs. Specifically, a graph $G=(V=[n],E)$ is called an *interval graph* if its vertices can be mapped to intervals $I_1,\dots,I_n$ of the real line in such a way that two distinct vertices $x,y\in V$ are adjacent in G if and only if $I_x\cap I_y\neq\emptyset$. This map is also called a *realization* of G.

Recall that a matrix has C1P if its columns can be reordered in such a way that ones are consecutive in its rows. Furthermore, let the vertex-clique incidence matrix be the 0/1 matrix whose rows are indexed by the vertices and the columns by the maximal cliques of G, with entry (i,j) equal to one if and only if vertex i is contained in the clique corresponding to column j. Then the following holds.

2.3.2 Theorem. *[57] G is an interval graph if and only if its vertex-clique incidence matrix has C1P.*

As for chordal graphs, also interval graphs can be recognized in linear time [14, 107, 37, 61, 38]. A famous algorithm is the one in [14], which uses PQ-trees to check whether the vertex-clique incidence matrix of the given graph has C1P.

Unit interval graphs Unit interval graphs are a subclass of interval graphs admitting a realization by intervals of the same (unit) length. There exist several equivalent characterizations for unit interval graphs. Most of the recognition algorithms are based on the equivalence between unit interval graphs and proper interval graphs or indifference graphs [97]. Specifically, a proper interval graph is an interval graph admitting a realization by pairwise incomparable intervals. An indifference graph is instead an interval graph admitting a realization with the property that there exists an edge between two vertices if the corresponding intervals are within one unit of each other. The next theorem summarizes several known characterizations for unit interval graphs, combining results from [35, 91, 82, 97, 98, 59]. Recall that $K_{1,3}$ is the graph with one degree-3 vertex connected to three degree-1 vertices (also known as *claw*).

2.3.3 Theorem. *The following are equivalent for a graph $G = (V, E)$.*

(i) *G is a unit interval graph.*

(ii) *G is an interval graph with no induced subgraph $K_{1,3}$.*

(iii) **(3-vertex condition)** *There is a linear order π of V such that,*

$$x <_\pi y <_\pi z,\ \{x,z\} \in E \Longrightarrow \{x,y\},\{y,z\} \in E \quad \forall x,y,z \in V. \tag{2.3}$$

(iv) **(Neighborhood condition)** *There is a linear order π of V such that, for any $x \in V$, the vertices in $N[x]$ are consecutive with respect to π.*

(v) **(Clique condition)** *There is a linear order π of V such that the vertices contained in the same maximal clique of G are consecutive with respect to π.*

(vi) *Its extended adjacency matrix has symmetric C1P.*

We will see in Section 3.2 that matrices with C1P are a special class of Robinsonian matrices; this motivates our interest in unit interval graphs and represents also the inspiration for the recognition algorithms in Chapters 5 and 6. Another important characterization of unit interval graphs we will use in Chapter 5 is the one in terms of straight enumerations. Recall that the blocks of a graph $G = (V, E)$ are the classes of the undistinguishability equivalence relation on V. Then, a straight enumeration of G is a special orderings of the blocks of G, which is defined as follows.

2.3.4 Definition. (**Straight enumeration**) [67] A *straight enumeration* of G is a linear order $\phi = (B_1, \dots, B_p)$ of the blocks of G such that, for any block B_i, the block B_i and the blocks B_j adjacent to it are consecutive in the linear order. The blocks B_1 and B_p are called the *end blocks* of ϕ and B_i (with $1 < i < p$) are its *inner blocks*.

In Subsection 5.3.2 we will show how to compute straight enumerations of unit interval graphs with the Lex-BFS algorithm discussed in Chapter 4. In fact, the following theorem will play a central role in Chapter 5.

2.3.5 Theorem. *[43] A graph G is a unit interval graph if and only if it has a straight enumeration. Moreover, if G is connected, then it has a unique (up to reversal) straight enumeration.*

On the other hand, if G is not connected, then any possible ordering of the connected components combined with any possible orientation of the straight enumeration of each connected component induces a straight enumeration of G. Theorem 2.3.5 will be the main motivation for our Lex-BFS based recognition algorithm for Robinsonian matrices described in Chapter 5.

Interval hypergraphs A hypergraph $H = (V, \mathcal{E})$ is a generalization of the notion of graph where elements of $\mathcal{E}$, called *hyperedges*, are subsets of V. The incidence matrix of H is the 0/1 matrix whose rows and columns are labeled, respectively, by the hyperedges and the vertices and with an entry 1 when the hyperedge contains the corresponding vertex. Then H is an *interval hypergraph* if its vertices can be ordered in such a way that hyperedges are intervals, i.e., if its incidence matrix has C1P. As for interval graphs, C1P can be checked using the PQ-tree algorithm of Booth and Leuker [14]. We will use interval hypergraphs in Subsection 3.2.2 to characterize Robinsonian matrices.

3

Robinsonian matrices

In this chapter we present Robinsonian matrices, a special structured class of matrices which will play a fundamental role in this thesis. In Section 3.1 we give a formal definition, outlining their applications in some important combinatorial and classification problems. Then, in Section 3.2 we describe the existing recognition algorithms, which we group in two classes: combinatorial algorithms, derived from graph characterizations of Robinsonian matrices, and spectral algorithms. Finally, in Section 3.3 we conclude the chapter anticipating the work which will be discussed in the second and third part of the thesis with respect to Robinsonian matrices.

3.1 Introduction

Robinsonian matrices were introduced by Robinson [100] to model the seriation problem discussed in Chapter 1. They also play an important role in many classification problems, where the goal is to find an order of a collection of objects such that similar objects are ordered close to each other. In Subsection 3.1.1 we formally define Robinson(ian) matrices. Then, in Subsection 3.1.2 we motivate our interest in Robinsonian matrices by discussing their applications to the seriation problem and to pyramidal clustering among others.

As already mentioned in Section 2.1, we will use the letter A or D to denote, respectively, a similarity or dissimilarity matrix. Hence, we consider a symmetric proximity matrix $A \in \mathcal{S}^n$, where rows (and columns) represent the objects that need to be reordered. Each entry A_{xy} represents the pairwise measure between objects x and y, for $x, y \in [n]$. Since we deal mainly with symmetric matrices, we will depict in figures only their upper triangular part. The support graph of A is the undirected graph indexed by $[n]$ whose edges are the pairs $\{x, y\}$ with $A_{xy} > 0$, for $x, y \in [n]$. Recall from Subsection 2.2.1 that, given a permutation π of $[n]$, $A_\pi := (A_{\pi(x)\pi(y)})_{x,y=1}^n$ is the matrix obtained from A by permuting both its rows and columns simultaneously according to π. Hence, when we say that A is ordered according to π we mean in fact both its rows and columns.

3.1.1 Basic definitions

Robinsonian similarities A symmetric matrix $A \in \mathcal{S}^n$ is called a *Robinson similarity* (also known as R-matrix) if its entries are monotone nondecreasing in the rows and columns when moving towards the main diagonal, i.e., if

$$A_{xz} \le \min\{A_{xy}, A_{yz}\} \quad \text{for each} \quad 1 \le x < y < z \le n. \tag{3.1}$$

If there exists a permutation π of $[n]$ such that the matrix A_π is a Robinson similarity, then A is said to be a *Robinsonian similarity* and π is called a *Robinson ordering* of A. One can easily verify that A is a Robinson similarity if and only if it satisfies the following four-points condition (see [29]):

$$A_{xy} \ge A_{uv} \quad \text{for each} \quad 1 \le u \le x < y \le v \le n. \tag{3.2}$$

Moreover, we say that A is a *strongly-Robinsonian similarity* if there exists a Robinson ordering π of A such that the following holds (see [89]):

$$A_{uv} < \min\{A_{uy}, A_{xv}\} \Rightarrow A_{xy} > \max\{A_{uy}, A_{xv}\}, \quad \forall u <_\pi x <_\pi y <_\pi v. \tag{3.3}$$

Note that in (3.1) the diagonal entries do not play a role. Hence we can ignore them, denoting by $*$ the entries on the main diagonal, as in the similarity matrix A in (3.4) shown in the example below.

$$A = \begin{array}{c} \\ \mathbf{1}\\ \mathbf{2}\\ \mathbf{3}\\ \mathbf{4}\\ \mathbf{5}\\ \mathbf{6}\\ \mathbf{7} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \end{array} \\ \begin{pmatrix} * & 0 & 0 & 0 & 7 & 0 & 6 \\ & * & 7 & 3 & 2 & 5 & 2 \\ & & * & 3 & 1 & 6 & 2 \\ & & & * & 6 & 3 & 7 \\ & & & & * & 1 & 7 \\ & & & & & * & 1 \\ & & & & & & * \end{pmatrix} \end{array}, \quad A_\pi = \begin{array}{c} \\ \mathbf{1}\\ \mathbf{5}\\ \mathbf{7}\\ \mathbf{4}\\ \mathbf{2}\\ \mathbf{3}\\ \mathbf{6} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathbf{1} & \mathbf{5} & \mathbf{7} & \mathbf{4} & \mathbf{2} & \mathbf{3} & \mathbf{6} \end{array} \\ \begin{pmatrix} * & 7 & 6 & 0 & 0 & 0 & 0 \\ & * & 7 & 6 & 2 & 1 & 1 \\ & & * & 7 & 2 & 2 & 1 \\ & & & * & 3 & 3 & 3 \\ & & & & * & 7 & 5 \\ & & & & & * & 6 \\ & & & & & & * \end{pmatrix} \end{array} \tag{3.4}$$

Then A is a Robinsonian similarity, since, $\pi = (1, 5, 7, 4, 2, 3, 6)$ is a Robinson ordering of A. Indeed, the matrix A_π given in (3.4) is a Robinson similarity.

Robinsonian dissimilarities In the literature, a distinction is made between Robinson(ian) similarities and Robinson(ian) dissimilarities. A symmetric matrix D is called a *Robinson dissimilarity* matrix (also known as *anti-R* or *anti-Robinson* matrix) if its entries are monotone nondecreasing in the rows and columns when moving away from the main diagonal, i.e., if:

$$D_{xz} \geq \max\{D_{xy}, D_{yz}\} \quad \text{for each} \quad 1 \leq x < y < z \leq n. \tag{3.5}$$

As for Robinson similarity matrices, if there exists a permutation π of $[n]$ such that the matrix D_π is a Robinson dissimilarity matrix, then D is said to be a *Robinsonian dissimilarity* and π is called a *Robinson ordering* of D. One can easily verify that D is a Robinson dissimilarity if and only if it satisfies the following four-points condition (see [29]):

$$A_{xy} \leq A_{uv} \quad \text{for each} \quad 1 \leq u \leq x < y \leq v \leq n. \tag{3.6}$$

Then, we say that A is a *strongly-Robinsonian dissimilarity* if there exists a Robinson ordering π of D such that the following holds (see [89]):

$$A_{uv} > \max\{A_{uy}, A_{xv}\} \Rightarrow A_{xy} < \min\{A_{uy}, A_{xv}\}, \quad \forall u <_\pi x <_\pi y <_\pi v \tag{3.7}$$

Again, main diagonal entries do not play a role in expression (3.5). However, it is commonly assumed in the literature that they are equal to 0. Consider, for example, the dissimilarity matrix A given below in (3.8).

$$D = \begin{array}{c|ccccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \\ \hline \mathbf{1} & * & 8 & 8 & 8 & 1 & 8 & 2 \\ \mathbf{2} & & * & 1 & 5 & 6 & 3 & 6 \\ \mathbf{3} & & & * & 5 & 7 & 2 & 6 \\ \mathbf{4} & & & & * & 2 & 5 & 1 \\ \mathbf{5} & & & & & * & 7 & 1 \\ \mathbf{6} & & & & & & * & 7 \\ \mathbf{7} & & & & & & & * \end{array}, \quad D_\pi = \begin{array}{c|ccccccc} & \mathbf{1} & \mathbf{5} & \mathbf{7} & \mathbf{4} & \mathbf{2} & \mathbf{3} & \mathbf{6} \\ \hline \mathbf{1} & * & 1 & 2 & 8 & 8 & 8 & 8 \\ \mathbf{5} & & * & 1 & 2 & 6 & 7 & 7 \\ \mathbf{7} & & & * & 1 & 6 & 6 & 7 \\ \mathbf{4} & & & & * & 5 & 5 & 5 \\ \mathbf{2} & & & & & * & 1 & 3 \\ \mathbf{3} & & & & & & * & 2 \\ \mathbf{6} & & & & & & & * \end{array} \tag{3.8}$$

Then D is a Robinsonian dissimilarity, as the permutation $\pi = (1, 5, 7, 4, 2, 3, 6)$ is a Robinson ordering of D. Indeed, the matrix D_π, given in (3.8), is a Robinson dissimilarity matrix.

Robinson(ian) similarities and dissimilarities are different sides of the same coin: $A \in \mathcal{S}^n$ is a Robinson(ian) similarity if and only if $-A$ is a Robinson(ian) dissimilarity. Furthermore, if π is a Robinson ordering for the similarity A, then it is also a Robinson ordering for the dissimilarity $-A$. In this thesis, we stick to the original definition in [100], and thus we shall focus only on Robinson(ian) similarity matrices, as the properties extend directly from one class to the other one. Hence, unless specified otherwise, when talking about Robinson(ian) matrices we always mean Robinson(ian) similarities.

Note that the all-ones matrix J_n is both a similarity and a dissimilarity Robinson matrix and thus adding any multiple of J_n preserves the Robinson property. In other words, if A is a Robinson(ian) matrix then $A+\lambda J_n$ is also a Robinson(ian) matrix for any scalar $\lambda \in \mathbb{R}$. Hence, we may consider, without loss of generality, nonnegative similarities A. Furthermore, if we denote by $0 = \alpha_0 < \cdots < \alpha_L$ the distinct values taken by the entries of A, then we can build the corresponding nonnegative Robinson(ian) dissimilarity $D = \alpha J_n - A$ with $\alpha > \alpha_L$ (to create a distance matrix with null values only on the diagonal).

Indeed, the dissimilarity matrix D in (3.8) is obtained from the similarity matrix A in (3.8) by setting $D = 8J_7 - A$. Note that the same permutation $\pi = (1, 5, 7, 4, 2, 3, 6)$ is a Robinson ordering for both A and D.

Monotonic rectangular matrices As we now observe, Robinsonian matrices can also be used to capture a class of monotonic matrices. Call a rectangular matrix $B \in \mathbb{R}^{n\times k}$ *left-down monotonic* if its entries are nonincreasing in the rows and nondecreasing in the columns, i.e., if:

$$B_{i+1,j} \geq B_{ij} \geq B_{i,j+1} \quad \text{for all} \quad i \in [n], j \in [k].$$

Then, the problem of checking whether the rows of B can be reordered by a permutation Π_1 and its columns by a permutation Π_2 in such a way that $\Pi_1 B \Pi_2^{\mathsf{T}}$ is left-down monotonic, can be reformulated as an instance of deciding whether an associated matrix A is Robinsonian.
For this select a scalar $\mu > \max\limits_{i,j} B_{ij}$ and consider the following (block) matrix:

$$A = \left[\begin{array}{c|c} \mu J_n & B \\ \hline B^{\mathsf{T}} & \mu J_k \end{array}\right].$$

Then, for permutations Π_1 of $[n]$ and Π_2 of $[k]$, consider their concatenation:

$$\Pi = \left[\begin{array}{c|c} \Pi_1 & 0 \\ \hline 0 & \Pi_2 \end{array}\right],$$

which is a permutation of $[n+k]$. Since:

$$\Pi A \Pi^{\mathsf{T}} = \left[\begin{array}{c|c} \mu J_n & \Pi_1 B \Pi_2^{\mathsf{T}} \\ \hline \Pi_2 B^{\mathsf{T}} \Pi_1^{\mathsf{T}} & \mu J_k \end{array}\right],$$

it follows that $\Pi_1 B \Pi_2^{\mathsf{T}}$ is left-down monotonic if and only if $\Pi A \Pi^{\mathsf{T}}$ is Robinson. On the other hand, if Π is a permutation of $[n+k]$ reordering A as a Robinson matrix, then it is easy to see that Π must induce a permutation Π_1 of $[n]$ and a permutation Π_2 of $[n+1, \ldots, n+k]$.

Therefore, testing whether the rows and columns of B can be reordered (by, respectively, Π_1 and Π_2) to get a left-down monotonic matrix is equivalent to test whether A is Robinsonian. The same reasoning can be extended to recognize *right-up monotonic* matrices (defined in the same fashion as left-down monotone matrices) via Robinsonian dissimilarities. .

3.1.2 Motivation

As already discussed in Chapter 1, Robinsonian matrices play an important role in the seriation problem, as they best achieve its goal of ordering similar objects close to each other.

The Robinsonian structure is a strong property and, even though the data is not Robinsonian, Robinsonian recognition algorithms can be used as core subroutines to design efficient heuristics or approximation algorithms for solving the seriation problem (see, e.g., [30, 56]). As already discussed in Chapter 1, Robinsonian matrices can be also used to build a consistent ranking of items given pairwise comparisons [55]. We will see in Corollary 7.2.6 that Robinsonian matrices also play an important role in the quadratic assignment problem (QAP). Indeed, they represent a class of instances of QAP which can be solved in polynomial time, while general QAP is NP-hard.

Furthermore, Robinsonian similarities matrices are a generalization of 0/1 matrices with C1P, which are used especially in bionformatics, e.g., sequencing DNA applications (see, e.g.,[3]).

We present below applications of Robinsonian matrices to cluster and data analysis. As already discussed in Chapter 1, data analysis is an enormous field, and a detailed discussion is out of scope for this thesis. We present below some basic concepts related to hierarchical and pyramidal clustering without going into details, to underline the importance of seriation and Robinsonian matrices. We refer the reader interested in clustering to the book [53], the reader interested in hierarchical clustering to [90] and the reader interested in pyramidal clustering to the recent work [10].

Hierarchies A *hierarchy* of a ground set E is a collection $\mathcal{H}$ of nonempty subsets of E (called *classes* or *clusters*) with the following properties:

(C1) $E \in \mathcal{H}$.

(C2) for each $x \in E$ then $\{x\} \in \mathcal{H}$.

(C3) for each $H, H' \in \mathcal{H}$, either $H \cap H' = \emptyset$, or $H \subseteq H'$, or $H \supseteq H'$.

In other words, a hierarchy is a collection of nested subsets of E containing the singletons clusters and the universal cluster (i.e., the ground set E). Given a hierarchy $\mathcal{H}$ of E, an *index* on $\mathcal{H}$ is a function $i : \mathcal{H} \to \mathbb{R}^+$ for which the following properties hold:

(D1) $i(H) = 0$ if and only if H is a singleton class

(D2) if $H, H' \in \mathcal{H}$ and $H \subset H'$ then $i(H) < i(H')$

In other words, an index $i(H)$ may be seen as the maximum dissimilarity between any two elements of the cluster H. We refer to the pair $(\mathcal{H}, i)$ as an *indexed hierarchy*. We introduce the concept of *ultrametric*, which is a subclass of metric (see Subsection 2.1), where the triangle inequality *(ii)* is replaced by the condition $D_{ik} \leq \max\{D_{ij}, D_{jk}\}$, also known as strong triangle inequality or ultrametric inequality. Ultrametrics have many applications in *taxonomy* and *phylogenetic* tree construction (see e.g., [87] and references therein). A well known result in the literature is that ultrametrics are in one-to-one correspondence with indexed hierarchies [72, 73]. Indexed hierarchies are produced by hierarchical clustering algorithms, which are widely used tools in the field of clustering and data analysis [90]. They can be visualized through a special binary tree diagram, called a *dendrogram*, where each node corresponds to a cluster of the hierarchy and it is positioned at the height corresponding to its index (see Figure 3.1). The order of the leaves of the dendrogram represents a permutation of the ground set E.

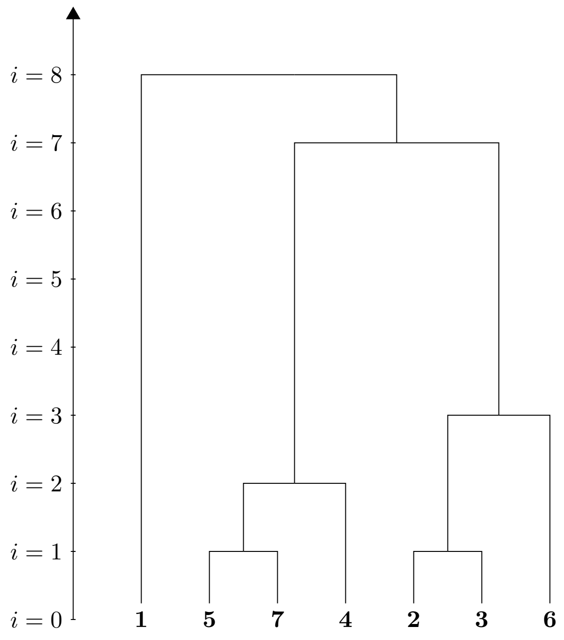


Figure 3.1: A possible dendrogram visualization of the indexed hierarchy $(\mathcal{H}, i)$, where $\mathcal{H} = \{\{1\}, \{5\}, \{7\}, \{4\}, \{2\}, \{3\}, \{6\}, \{5, 7\}, \{2, 3\}, \{4, 5, 7\}, \{2, 3, 6\}, \{2, 3, 4, 5, 6, 7\}, \{1, 2, 3, 4, 5, 6, 7\}\}$ is a hierarchy corresponding to the dissimilarity matrix D_π in (3.8).

In general, a linear order π of the set of objects E is said to be compatible with the hierarchy $\mathcal{H}$ if every cluster of $\mathcal{H}$ is an interval when E is sorted according to π. Then, for each hierarchy $\mathcal{H}$ of E, there exist $2^{|E|} - 1$ possible rearrangements of the leaves of the dendrogram compatibly with $\mathcal{H}$, because each node of the (binary) dendrogram tree can be arbitrarily reordered [10]. Furthermore, given a hierarchy $\mathcal{H}$ of E and a linear π order of E, the dendrogram can uniquely be drawn. In this stream of work, the seriation problem can be thus used in a second stage to obtain a better visualization of the hierarchy returned by a hierarchical clustering algorithm. Indeed, since Robinsonian matrices best achieve the seriation goal to order similar objects close to each other, a Robinson ordering can be used to order the leaves of the dendrogram in such a way that similar elements in the clusters are close to each other [52].

Pyramids A *pyramid* $\mathcal{P}$ (also known as a *pseudo-hierarchy*) is a relaxation of the concept of hierarchy, and it is obtained by keeping conditions (C1) and (C2) and by replacing condition (C3) with the following two conditions:

(C4) for each $P, P' \in \mathcal{P}$, either $P \cap P' = \emptyset$ or $P \cap P' \in \mathcal{P}$.

(C5) there exists a linear order π of E such that every cluster of $\mathcal{P}$ is an interval when E is sorted according to π, i.e., π is compatible with $\mathcal{P}$.

Note that condition (C4) allows overlapping among the classes of $\mathcal{P}$. Pyramids were in fact introduced to cope with the limitation of hierarchies, which cannot be used to represent elements that belong to more than one cluster (e.g., multifunctional proteins [9]). Let i be an index satisfying (D1) and (D2). Then the pair $(\mathcal{P}, i)$ is called an *indexed pyramid*. As pyramids are a generalization of hierarchies, one can also generalize the concept of index. Specifically, a *weak-index* of a pyramid $\mathcal{P}$ is obtained by keeping the condition (D1) and relaxing the condition (D2) with the two following conditions:

(D3) if $P, P' \in \mathcal{P}$ and $P \subset P'$ then $i(P) \leq i(P')$.

(D4) if $P, P' \in \mathcal{P}$, $P \subsetneq P'$ and $i(P) = i(P')$ then $P = \bigcap\{P'' \in \mathcal{P} : P \subsetneq P''\}$.

We refer to the pair $(\mathcal{P}, i)$ as an *indexed weak-pyramid*. Then, analogously to the bijection between ultrametrics and indexed hierarchies, there exists a bijection between pyramids and Robinsonian dissimilarities. Precisely, Robinsonian dissimilarities are in one-to-one correspondence with weakly indexed pyramids [49], while strongly Robinsonian dissimilarities are in one-to-one correspondence with indexed pyramids [50]. Pyramids are used in pyramidal clustering and they can be visualized through a special tree diagram, which is the analog of the for hierarchical clustering, with the difference that each node now has at most two parents (see Figure 3.2).

This aspect reflects the property of overlapping of clusters in pyramids. For applications of pyramidal clustering in classification and data analysis we refer the interested reader to [50, 40, 6, 83, 108]. For more details on Robinsonian matrices and pyramids we refer instead to [45, 89, 11, 10].

3.2 Recognition algorithms

As we have seen in Subsection 3.1.2, in many applications it is important to recognize Robinsonian matrices, i.e., given a proximity matrix, determine whether it is Robinsonian and, if so, find a Robinson ordering of the matrix. In this section

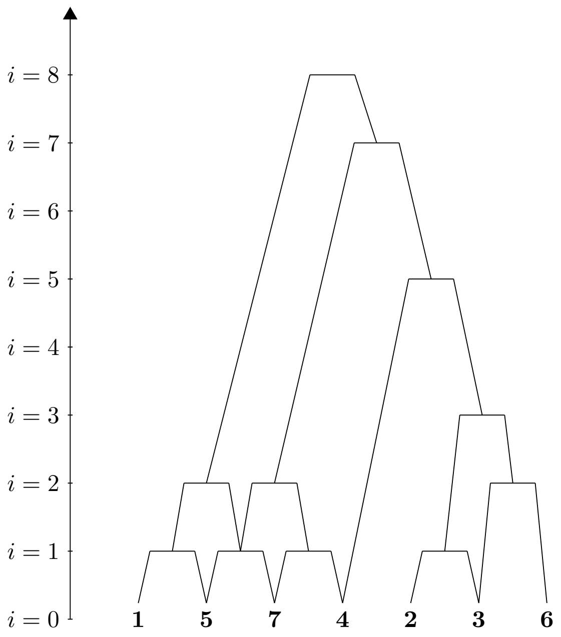


Figure 3.2: A possible visualization of the indexed pyramid $(\mathcal{P}, i)$, where $\mathcal{P} = \{\{1\},\{5\},\{7\},\{4\},\{2\},\{3\},\{6\},\{1,5\},\{5,7\},\{4,7\},\{2,3\},\{3,6\},\{1,5,7\},\{2,3,6\},$ $\{5,4,7\},\{2,3,4,5,6,7\},\{1,2,3,4,5,6,7\}\}$ is a pyramid corresponding to the dissimilarity matrix D_π in (3.8).

we give a short overview of the main recognition algorithms for Robinsonian matrices existing in the literature.

It is straightforward to verify whether a given matrix $A \in \mathcal{S}^n$ is a Robinson similarity, by checking if $A_{xy} \geq \max\{A_{x,(y+1)}, A_{(x-1),y}\}$ holds for each $1 \leq x < y \leq n$, which is equivalent to relation (3.1). Moreover, as we will see soon, one can decide in polynomial time whether A is Robinsonian.

In this section we first characterize a special class of Robinsonian matrices, namely 0/1 Robinsonian matrices. Then we briefly review the main existing recognition algorithms, dividing them in two classes: combinatorial algorithms and spectral algorithms. Since to the best of our knowledge the combinatorial algorithms are in fact not implemented or not available, we will focus more on the description of the spectral algorithm, which will be used as benchmark in Chapter 9 with respect to the recognition algorithms presented in Chapters 5 and 6.

3.2.1 Binary Robinsonian matrices

Any symmetric binary matrix $A \in \{0,1\}^{n\times n}$ can be seen as the (extended) adjacency matrix of a graph $G = (V = [n], E)$ whose edges are the positions of the nonzero off-diagonal entries of A. Then the following holds.

3.2.1 Theorem. *[97] Let $A \in \mathcal{S}^n \cap \{0,1\}^{n\times n}$ be the (extended) adjacency matrix of a graph $G = (V = [n], E)$. Then A is a Robinsonian similarity matrix if and only if G is a unit interval graph.*

Indeed, if A is the extended adjacency matrix of G then A is Robinsonian if and only if it has symmetric C1P (see Definition 2.1.1), which in turn is equivalent to the condition (vi) of Theorem 2.3.3 and thus for G being a unit interval graph. Hence, 0/1 Robinsonian matrices have the block structure given below.

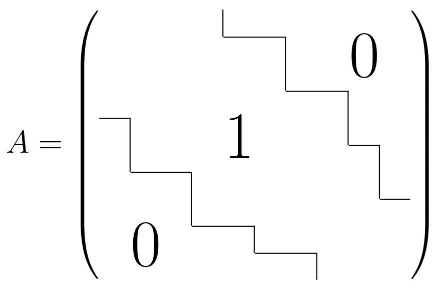


Figure 3.3: The block structure of a 0/1 Robinsonian matrix.

Furthermore, it is easy to see that the 3-vertex condition (iii) in Theorem 2.3.3 coincides with relation (3.1) for 0/1 matrices. This equivalence and the fact that unit interval graphs can be recognized with a Lex-BFS multisweep algorithm [33] will motivate our algorithm discussed in Chapter 5.

0/1 Robinsonian matrices can thus be recognized in linear time in terms of the size of the matrix, using recognition algorithms either for C1P (like the algorithm of Booth and Leuker [14], which is based on PQ-trees), or for unit interval graphs (see Section 2.3).

The link between Robinsonian matrices and C1P can be extended also to non symmetric 0/1 matrices as follows.

3.2.2 Theorem. *[75] Let $C \in \{0,1\}^{m\times n}$ be a matrix and define $A = C^{\mathsf{T}}C$. Let $\Pi \in \{0,1\}^{n\times n}$ be a permutation matrix. Then $C\Pi$ has consecutive ones in its rows if and only if $\Pi^{\mathsf{T}}A\Pi$ is a Robinson similarity matrix.*

Hence, checking if C has C1P reduces to a special instance of Robinsonian matrix recognition. We refer the interested reader to [47] and references therein for details about algorithms for C1P. In Chapter 5 we will present an algorithm to recognize 0/1 Robinsonian matrices based on computing straight enumerations of unit interval graphs with the Lex-BFS algorithm discussed in Chapter 4. This algorithm relies on Theorem 2.3.5 combined with Theorem 3.2.1.

3.2.2 Graph characterizations

In the last decades, different graph characterizations of Robinsonian matrices have appeared in the literature, leading to different polynomial time recognition algorithms. The first bunch of recognition algorithms relies on the characterization of Robinsonian dissimilarities matrices in terms of interval hypergraphs. Given a dissimilarity matrix $D \in \mathcal{S}^n$ and a scalar α, the *threshold graph* $G_\alpha = (V, E_\alpha)$ has edge set $E_\alpha = \{\{x, y\} : D_{xy} \leq \alpha\}$ and, for $x \in V$, the ball $B(x, \alpha) := \{y \in V : D_{xy} \leq \alpha\}$ consists of x and its neighbors in G_α. Let $\mathcal{B}$ denote the collection of all the balls of D (see Table 3.1) and let $H_\mathcal{B}$ denote the corresponding *ball hypergraph*, with vertex set $V = [n]$ and with $\mathcal{B}$ as set of hyperedges (see Table 3.2).

	$\alpha = 0$	$\alpha = 1$	$\alpha = 2$	$\alpha = 3, 4$	$\alpha = 5$	$\alpha = 6$	$\alpha = 7$	$\alpha = 8$
1	$\{1\}$	$\{1,5\}$	$\{1,5,7\}$	$\{1,5,7\}$	$\{1,5,7\}$	$\{1,5,7\}$	$\{1,5,7\}$	V
2	$\{2\}$	$\{2,3\}$	$\{2,3\}$	$\{2,3,6\}$	$\{2,3,4,6\}$	$V \setminus \{1\}$	$V \setminus \{1\}$	V
3	$\{3\}$	$\{2,3\}$	$\{2,3,6\}$	$\{2,3,6\}$	$\{2,3,4,6\}$	$V \setminus \{1,5\}$	$V \setminus \{1\}$	V
4	$\{4\}$	$\{4,7\}$	$\{4,5,7\}$	$\{4,5,7\}$	$V \setminus \{1\}$	$V \setminus \{1\}$	$V \setminus \{1\}$	V
5	$\{5\}$	$\{1,5,7\}$	$\{1,4,5,7\}$	$\{1,4,5,7\}$	$\{1,4,5,7\}$	$V \setminus \{3,6\}$	V	V
6	$\{6\}$	$\{6\}$	$\{3,6\}$	$\{2,3,6\}$	$\{2,3,4,6\}$	$\{2,3,4,6\}$	$V \setminus \{1\}$	V
7	$\{7\}$	$\{4,5,7\}$	$\{1,4,5,7\}$	$\{1,4,5,7\}$	$\{1,4,5,7\}$	$V \setminus \{6\}$	V	V

Table 3.1: Collection $\mathcal{B}$ of the balls of the dissimilarity matrix in (3.8).

$e_1 = \{1\}$	$e_8 = \{1,5\}$	$e_{15} = \{1,4,5,7\}$
$e_2 = \{2\}$	$e_9 = \{2,3\}$	$e_{16} = \{2,3,4,6\}$
$e_3 = \{3\}$	$e_{10} = \{3,6\}$	$e_{17} = \{1,2,4,5,7\}$
$e_4 = \{4\}$	$e_{11} = \{4,7\}$	$e_{18} = \{2,3,4,6,7\}$
$e_5 = \{5\}$	$e_{12} = \{1,5,7\}$	$e_{19} = \{1,2,3,4,5,7\}$
$e_6 = \{6\}$	$e_{13} = \{2,3,6\}$	$e_{20} = \{2,3,4,5,6,7\}$
$e_7 = \{7\}$	$e_{14} = \{4,5,7\}$	$e_{21} = \{1,2,3,4,5,6,7\}$

Table 3.2: Hyperedges of the ball hypergraph $H_\mathcal{B}$ corresponding to the dissimilarity matrix in (3.8).

Most of the existing algorithms are then based on the following result.

3.2.3 Theorem. *[88] $D \in \mathcal{S}^n$ is a Robinsonian dissimilarity if and only if the ball hypergraph $H_\mathcal{B}$ is an interval hypergraph.*

Recall from Section 2.3 that the incidence matrix of an hypergraph is the 0/1 matrix whose rows and columns are labeled, respectively, by the hyperedges and the vertices, with an entry equal to 1 when the hyperedge contains the corresponding

	1	**2**	**3**	**4**	**5**	**6**	**7**
e_1	1	0	0	0	0	0	0
e_2	0	1	0	0	0	0	0
e_3	0	0	1	0	0	0	0
e_4	0	0	0	1	0	0	0
e_5	0	0	0	0	1	0	0
e_6	0	0	0	0	0	1	0
e_7	0	0	0	0	0	0	1
e_8	1	0	0	0	1	0	0
e_9	0	1	1	0	0	0	0
e_{10}	0	0	1	0	0	1	0
e_{11}	0	0	0	1	0	0	1
e_{12}	1	0	0	0	1	0	1
e_{13}	0	1	1	0	0	1	0
e_{14}	0	0	0	1	1	0	1
e_{15}	1	0	0	1	1	0	1
e_{16}	0	1	1	1	0	0	1
e_{17}	1	1	0	1	1	0	1
e_{18}	0	1	1	1	0	1	1
e_{19}	1	1	1	1	1	0	1
e_{20}	0	1	1	1	1	1	1
e_{21}	1	1	1	1	1	1	1

(a) Ball hypergraph unordered.

	1	**5**	**7**	**4**	**2**	**3**	**6**
e_1	1	0	0	0	0	0	0
e_2	0	0	0	0	1	0	0
e_3	0	0	0	0	0	1	0
e_4	0	0	0	1	0	0	0
e_5	0	1	0	0	0	0	0
e_6	0	0	0	0	0	0	1
e_7	0	0	1	0	0	0	0
e_8	1	1	0	0	0	0	0
e_9	0	0	0	0	1	1	0
e_{10}	0	0	0	0	0	1	1
e_{11}	0	0	1	1	0	0	0
e_{12}	1	1	1	0	0	0	0
e_{13}	0	0	0	0	1	1	1
e_{14}	0	1	1	1	0	0	0
e_{15}	1	1	1	1	0	0	0
e_{16}	0	0	0	1	1	1	1
e_{17}	1	1	1	1	1	0	0
e_{18}	0	0	1	1	1	1	1
e_{19}	1	1	1	1	1	1	0
e_{20}	0	1	1	1	1	1	1
e_{21}	1	1	1	1	1	1	1

(b) Ball hypergraph with C1P.

Table 3.3: Incidence matrix of the ball hypergraph $H_{\mathcal{B}}$ of the dissimilarity matrix in (3.8).

vertex. Then, a hypergraph is an interval hypergraph if and only if its incidence matrix has C1P, i.e., there exists a order π of $V = [n]$ such that all the balls of D are intervals with respect to π.

Mirkin and Rodin [88] gave the first polynomial algorithm to recognize Robinsonian matrices, with $O(n^4)$ running time, based on checking whether the ball hypergraph is an interval hypergraph and using the PQ-tree algorithm of [14] to check whether the incidence matrix has C1P (see Table 3.3). Later, Chepoi and Fichet [28] introduced a simpler algorithm that, using a divide-an-conquer strategy and sorting the entries of D, improved the running time to $O(n^3)$. The same sorting preprocessing was used by Seston [106], who improved the algorithm to $O(n^2 \log n)$ by constructing paths in the threshold graphs of D.

Equivalently, one can also build the intersection graph $G_{\mathcal{B}}$ of $\mathcal{B}$, where the balls are the vertices and connecting two vertices if the corresponding balls intersect. Then, Theorem 3.2.3 can be shown to be equivalent to the next statement.

3.2.4 Theorem. *[94] $D \in \mathcal{S}^n$ is a Robinsonian dissimilarity if and only if the intersection graph $G_{\mathcal{B}}$ is an interval graph.*

Recall that the vertex-clique incidence matrix of a graph G is the matrix whose rows are indexed by the vertices and the columns by the maximal cliques of G (see Table 3.4). Then, since a graph is an interval graph if and only if its vertex-clique matrix has C1P (see Theorem 2.3.2), one can check if the vertex-clique matrix of $G_{\mathcal{B}}$ has C1P in order to decide if D is Robinsonian.

$\mathcal{C}_1$	$=$	$\{e_1, e_8, e_{12}, e_{15}, e_{17}, e_{19}, e_{21}\}$
$\mathcal{C}_2$	$=$	$\{e_2, e_9, e_{13}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}\}$
$\mathcal{C}_3$	$=$	$\{e_3, e_9, e_{10}, e_{13}, e_{16}, e_{18}, e_{19}, e_{20}, e_{21}\}$
$\mathcal{C}_4$	$=$	$\{e_4, e_{11}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}\}$
$\mathcal{C}_5$	$=$	$\{e_5, e_8, e_{12}, e_{14}, e_{15}, e_{17}, e_{19}, e_{20}, e_{21}\}$
$\mathcal{C}_6$	$=$	$\{e_6, e_{10}, e_{13}, e_{16}, e_{18}, e_{20}, e_{21}\}$
$\mathcal{C}_7$	$=$	$\{e_7, e_{11}, e_{12}, e_{14}, e_{15}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}\}$

Table 3.4: Maximal cliques of the intersection graph $G_{\mathcal{B}}$ corresponding to the dissimilarity D in (3.8). Since D is Robinsonian, then each clique $\mathcal{C}_i$ is in correspondence with vertex i, for $i \in [7]$ (see [94]).

In this stream of works, very recently, Préa and Fortin [94] presented a more sophisticated $O(n^2)$ algorithm, based on the fact that the maximal cliques of the graph $G_{\mathcal{B}}$ are in one-to-one correspondence with the row/column indices of D. Roughly speaking, they use the algorithm from Booth and Leuker [14] to compute a first PQ-tree which they update throughout the algorithm. Furthermore, they return all the possible Robinson orderings, which can be useful in some practical applications.

All the above characterizations and algorithms referred to Robinsonian dissimilarities, and the corresponding recognition algorithms are based on the characterization of Robinsonian dissimilarities matrices in terms of interval (hyper)graphs. Furthermore, these algorithms use the sophisticated PQ-tree algorithm of [14] and, to the best of our knowledge, are not implemented.

This motivated us to investigate new simpler recognition algorithms which could be conceptually easy and also implementable. In this context, we will use the characterization of Robinsonian similarities matrices in terms of unit interval graphs (see Theorem 5.2.2).

Specifically, let $\alpha_0 < \alpha_1 < \cdots < \alpha_L$ denote the distinct values taken by the entries of A with $\alpha_0 = 0$ (recall that we can assume without loss of generality A to be nonnegative). The graph $G^{(\ell)} = (V, E_\ell)$, whose edges are the pairs $\{x, y\}$ with $A_{xy} \geq \alpha_\ell$, is called the *ℓ-th level graph* of A. Then, $G^{(1)}$ is the support graph of A whose edges are the pairs $\{x, y\}$ with $A_{xy} > 0$. The level graphs represent the analog for similarity matrices of the threshold graphs for dissimilarities, and they can be used to decompose A as a conic combination of 0/1 matrices.

This is summarized in the next lemma, whose easy proof is omitted.

3.2.5 Lemma. *Let $A \in \mathcal{S}^n$ with distinct values $\alpha_0 < \alpha_1 < \cdots < \alpha_L$ and with level graphs $G^{(1)}, \ldots, G^{(L)}$. Then, $A = \alpha_0 J + \sum_{\ell=1}^{L} (\alpha_\ell - \alpha_{\ell-1}) A_{G^{(\ell)}}$. Moreover, A is Robinson if and only if $A_{G^{(\ell)}}$ is Robinson for each $\ell \in [L]$.*

Clearly, if A is a Robinsonian matrix then the adjacency matrices of its level graphs $G^{(\ell)}$ are Robinsonian too, for each $\ell \in [L]$. However, the converse is not true. It is easy to build a small example where A is not Robinsonian although the extended adjacency matrix of each level graph is Robinsonian. Consider, for example, the following matrices:

$$A = \begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 0 \\ 1 & 2 & 2 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix}, \quad A_{G^{(1)}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}, \quad A_{G^{(2)}} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where $A_{G^{(1)}}$ and $A_{G^{(2)}}$ are the extended adjacency matrices of the level graphs of A. Both matrices $A_{G^{(1)}}$ and $A_{G^{(2)}}$ are Robinsonian. Indeed, the matrix $A_{G^{(2)}}$ is Robinson, and thus the identity permutation represents a Robinson ordering for $A_{G^{(2)}}$. Furthermore, $\pi = (2, 1, 3, 4)$ or $\pi = (2, 3, 1, 4)$ (and their reversals) are the only Robinson orderings of $A_{G^{(1)}}$. However, none of these permutations π is a Robinson ordering also of $A_{G^{(2)}}$. Hence A is not Robinsonian.

The difficulty lies in the fact that one needs to find a permutation that reorders *simultaneously* the extended adjacency matrices of all the level graphs as Robinson matrices. Roberts [99] first introduced a characterization of Robinsonian matrices in terms of indifference graphs (i.e., unit interval graphs). Rephrasing his result using the notion of level graphs, he showed that A is Robinsonian if and only if its level graphs have vertex linear orders that are compatible (see [99, Theorem 4.4]). However, he does not give any algorithmic insight on how to find such orders. We will see in Subsection 5.2 how to rephrase the above concept of compatibility of unit interval graphs in terms of their straight enumerations (see Theorem 5.2.2). This will lead to the new recognition algorithm presented in Chapter 5.

3.2.3 Spectral characterization

A completely different approach to recognize Robinsonian similarities was used by Atkins *et al.* [5], who introduced an interesting spectral sequencing algorithm. Given a matrix $A \in \mathcal{S}^n$, its *Laplacian matrix* is the matrix defined by the relation:

$$L_A = \mathrm{Diag}(Ae) - A \in \mathcal{S}^n,$$

where e is the all-ones vector and $\mathrm{Diag}(Ae)$ is the diagonal matrix whose diagonal is given by the vector Ae. If $A \geq 0$ then L_A is positive semidefinite and moreover its smallest eigenvalue is $\lambda_1(L_A) = 0$ (since $L_A e = 0$).

A *Fiedler vector* $y_F \in \mathbb{R}^n$ of A is an eigenvector corresponding to the second smallest eigenvalue $\lambda_2(L_A)$ of L_A, which is also known as the *Fiedler value.* If $\lambda_2(A)$ has multiplicity one, then the Fiedler value is called *simple*, and the Fiedler vector is uniquely determined up to a scalar multiple. The spectral algorithm of Atkins *et al.* [5] relies on the following properties of the Fiedler vector of a Robinson similarity matrix.

3.2.6 Theorem. *[5] If $A \in \mathcal{S}^n$ is a Robinson similarity then it has a monotone Fiedler vector y_F, i.e., satisfying: $y_F(1) \leq \ldots \leq y_F(n)$ or $y_F(n) \leq \ldots \leq y_F(1)$.*

3.2.7 Theorem. *[5] Assume that $A \in \mathcal{S}^n$ is a Robinsonian similarity, that its Fiedler value is simple and that the Fiedler vector y_F has no repeated entries. Let π be the permutation induced by sorting monotonically the values of y_F (in increasing or decreasing order). Then the matrix A_π is a Robinson similarity matrix.*

In other words, the above results show that sorting monotonically the Fiedler vector y_F of a similarity matrix A reorders A as a Robinson similarity. Hence, a simple algorithm to recognize Robinsonian similarity matrices consists in building the corresponding Laplacian matrix L_A and then sort the entries of its second eigenvector for increasing or decreasing values, which leads to a linear order π of $[n]$. Then, in view of Theorem 3.2.7, if A_π is a Robinson similarity then A is Robinsonian and π a Robinson ordering, whereas if A_π is not Robinson then A is not Robinsonian.

The authors in [5] discuss how to deal with the general case when the Fiedler vector does not satisfy the hypotheses in Theorem 3.2.7. Specifically, they prove that if a Robinsonian matrix A is irreducible and if its smallest off-diagonal entry is zero, then the Fiedler value is simple [5, Theorem 4.6].

Since a symmetric matrix A is irreducible if and only if its support graph is connected, one can identify in a preprocessing step the connected components of the support graph of A and deal with each of them independently [5, Lemma 4.2]. After this preprocessing step one can assume without loss of generality, that the Fiedler value of A is simple.

Finally, to cope with repeated values of the Fiedler vector, one applies recursively the spectral algorithm to each submatrix of A induced by the positions corresponding to all entries that are equal to a common value [5, Theorem 4.7].

Below, in Table 3.5, we show the original Fiedler vector of the Laplacian of the similarity matrix A in (3.4) (on the left) and after reordering it for increasing values (on the right). Note that the permutation $\pi = (1, 5, 7, 4, 2, 3, 6)$ obtained by sorting the Fiedler vector y_F is exactly the same Robinson ordering found in (3.4) and (3.8).

The complexity of the spectral algorithm in [5] is given by $O(n(T(n) + \log n))$, where $T(n)$ is the complexity of computing (approximately) eigenvalues of an $n \times n$

$$y_F = \begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \end{array} \begin{pmatrix} -0.6490 \\ 0.3414 \\ 0.3807 \\ 0.0105 \\ -0.2838 \\ 0.4292 \\ -0.2290 \end{pmatrix} \qquad y_F^{\pi} = \begin{array}{c} \mathbf{1} \\ \mathbf{5} \\ \mathbf{7} \\ \mathbf{4} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{6} \end{array} \begin{pmatrix} -0.6490 \\ -0.2838 \\ -0.2290 \\ 0.0105 \\ 0.3414 \\ 0.3807 \\ 0.4292 \end{pmatrix}$$

Table 3.5: The Fiedler vector of the matrix A in (3.4).

symmetric matrix. As for the algorithm in [94], the above spectral algorithm can also be extended to compute all the Robinson orderings of a given Robinsonian matrix using recursively PQ-trees. Given its simplicity, this algorithm is used in some classification applications (see, e.g., [55]) as well as in spectral clustering (see, e.g., [7]).

3.3 Conclusions

In this chapter we have introduced Robinson(ian) similarity and dissimilarity matrices, and we have outlined their importance in the seriation problem presented in Chapter 1 and more generally in classification and data analysis problems, discussing their relation with hierarchical and pyramidal clustering. We then presented the main existing recognition algorithms, which are based on the characterization of Robinsonian matrices in terms of interval graphs and interval hypergraphs. Furthermore, we discussed an interesting spectral algorithm which is used in practice to solve the seriation problem, and consists simply in sorting the second smallest eigenvector of the Laplacian of a similarity matrix.

Based on the concepts introduced in this chapter, in the second part of the thesis (Chapters 5 and 6) we will present two new characterizations of Robinsonian similarities matrices. Specifically, the first characterization (Chapter 5) is an equivalent formulation based on Lemma 3.2.5 in terms of *straight enumerations* of unit interval graphs (see Definition 2.3.4 and Theorem 5.2.2).

The second characterization (Chapter 6) is instead based on new combinatorial properties of Robinsonian matrices in terms of the notion of 'path avoiding a vertex' and of 'end points' of Robinson orderings. Both characterizations differ from most of the existing approaches discussed in Section 3.2 in the sense that they are not directly related to interval (hyper)graphs, but they are inspired by the Lex-BFS algorithm presented in the next chapter (Algorithm 4.3). Each characterization will lead to a new combinatorial recognition algorithm for Robin-

Table 3.6: Summary of existing recognition algorithms for Robinsonian matrices. Given $A \in \mathcal{S}^n$, then: m denotes its number of nonzero entries; L denotes its number of distinct values; $T(n)$ is the time to compute eigenvalues of A.

	Year	**Complexity**	**Subroutine**	**Characterization**
Mirkin & Rodin[88]	1984	$O(n^4)$	PQ-tree	interval hypergraphs
Chepoi & Fichet[28]	1997	$O(n^3)$	none	interval hypergraphs
Atkins et al.[5]	1998	$O(n(T(n) + n \log n))$	eigenvalues computation	Fiedler vector
Seston[106]	2008	$O(n^2 \log n)$	paths in threshold graphs	interval hypergraphs
Préa & Fortin[94]	2014	$O(n^2)$	PQ-tree	interval hypergraphs
Laurent & Seminaroti[77]	2015	$O(L(m + n))$	Lex-BFS	unit interval graphs [1]
Laurent & Seminaroti[79]	2016	$O(n^2 + mn \log n)$	SFS	path avoiding a vertex [2]

sonian matrices, which will be compared in Chapter 9 with the spectral algorithm from Atkins *et al.* [5].

In Table 3.6 we summarize the theoretical complexity of each algorithm presented in this chapter and of the new recognition algorithms which will be presented in Chapters 5 and 6.
For each algorithm, we outline the characterization of Robinsonian matrices which is exploited and the main (external) subroutine or task required.

[1]see Chapter 5.
[2]see Chapter 6.

4

Lexicographic Breadth-First Search

In this chapter we discuss Lexicographic Breadth-First search (Lex-BFS), which is a special graph traversal algorithm. In Section 4.1 we introduce graph traversal algorithms and we underline the importance of Lex-BFS for the recognition of several classes of graphs and of Robinsonian matrices. Then, in Section 4.2 we present in detail how Lex-BFS and its variant Lex-BFS$_+$ work. Furthermore, we discuss a linear time implementation of Lex-BFS (and Lex-BFS$_+$) using the partition refinement paradigm introduced in Subsection 2.2.3, and we show its application in multisweep algorithms. Finally, in Section 4.3 we discuss some structural properties of Lex-BFS when applied to special graph classes, and we conclude the chapter in Section 4.4 with directions for possible future work.

4.1 Motivation

Lexicographic Breadth-First Search, abbreviated Lex-BFS [1], is a graph traversal algorithm developed by Rose *et al.* [102] for the recognition of chordal graphs. A *graph traversal* algorithm (also called *graph search* algorithm) is a fundamental search paradigm, aiming to traverse all the vertices and the edges of a given graph.

[1]In some works Lexicographic Breadth-First Search is abbreviated by LBFS. However, since LBFS is also used to denote an industrial level switch, in our manuscript we will use the unambiguous abbreviation Lex-BFS.

We say that a vertex is *visited* if it is traversed during the algorithm; a vertex is *explored* if it is encountered during the algorithm, i.e., if it is adjacent to some already visited vertex; a vertex is *unvisited* if it is not visited. Roughly speaking, a generic graph search algorithm works as follows. At the beginning, every vertex is initialized as unvisited. Every time a vertex is traversed, it is marked as visited and its neighbors are marked as explored. The algorithm then iteratively visits the unvisited vertices of the graph, until no unvisited vertex exists anymore. At the end, it returns a linear order of the vertices representing the order in which the vertices have been traversed. Throughout the algorithm, unvisited vertices are maintained in a *priority queue*, i.e., a data structure which defines the priorities in visiting the vertices. If two vertices have the same priority, we say that we have *ties* and we break them according to their order in the queue. Different maintenance of the queue of unvisited vertices leads to different graph traversal algorithms.

The most famous graph traversal algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS). Specifically, BFS explores first the neighbors of formerly visited vertices, by storing explored vertices in a priority queue with a first-in, first-out (FIFO) strategy. In other words, 'oldest' explored vertices are visited first. By contrast, DFS explores first the neighbors of recently visited vertices, by storing the unvisited vertices in a priority queue with a last-in, first-out strategy (LIFO). In other words, 'newest' explored vertices are visited first. Both algorithms can be implemented in linear time in the size of the graph, and they have many applications in graph theory: BFS is used, e.g., to solve the shortest path problem and to compute the maximum flow in a flow network; DFS is used, e.g., to test planarity of graphs and to compute strongly connected components and topological orderings of directed graphs. For an exhaustive overview of BFS and DFS we refer the reader, e.g., to [32].

Lex-BFS is a variant of BFS, where vertices are explored by giving preference to those vertices whose neighbors have been visited earliest [102]. As BFS and DFS, also Lex-BFS can be implemented in linear time (see Subsection 4.2.3) and it plays an important role in the design of efficient recognition algorithms for special graph classes (e.g., chordal, interval and unit interval graphs) [33].

Lex-BFS will play a fundamental role in this thesis for two reasons. First, it can be directly applied to the recognition of Robinsonian matrices. In fact, Lex-BFS can be used to recognize 0/1 matrices with C1P [61], and thus also Robinsonian matrices (see Section 3.2). Furthermore, we will discuss in Chapter 5 a new recognition algorithm for Robinsonian matrices entirely based on Lex-BFS. Second, Lex-BFS represents the main inspiration for the Similarity-First Search algorithm discussed in Chapter 6, which can be seen as a generalization of Lex-BFS applied to weighted graphs, and which will be used as well to recognize Robinsonian matrices. Hence, a good understanding of Lex-BFS will facilitate the discussion of the new recognition algorithms in Chapters 5 and 6.

4.2 The algorithm

In this section we discuss in detail the Lex-BFS algorithm. First, we present a pseudocode of the algorithm (see Subsection 4.2.1) and its variant, denoted by Lex-BFS$_+$ (see Subsection 4.2.2), illustrating how the algorithms concretely work on a small example. Then, we show how to implement Lex-BFS (and Lex-BFS$_+$) in linear time using the partition refinement paradigm presented in Subsection 2.2.3 (see Subsection 4.2.3). Finally, we show how Lex-BFS can be used in a multisweep framework, where repeated Lex-BFS orderings are computed (see Subsection 4.2.4). The description of Lex-BFS is mainly based on the papers [61, 33, 38], where the Lex-BFS ordering represents the order in which the vertices in the graph have been visited (although in some related works (e.g., [15]), the vertices are actually returned in the reversed order).

4.2.1 Lex-BFS

Lex-BFS is a variant of the classic BFS algorithm applied to an undirected graph $G = (V, E)$ where, at each iteration, the vertex whose neighborhood has been visited earliest is traversed. The output is a linear order σ of the vertices V of G, called *Lex-BFS-ordering*, where $x <_\sigma y$ means that x has been visited before y. The algorithm works as follows. Each vertex has a label, which is a word of the alphabet $\{0, \ldots, |V-1|\}$ and it is initialized empty (denoted here by $\varnothing$). Starting from an arbitrary vertex, at each iteration i the algorithm chooses a vertex p to visit (called *pivot*) which is marked as visited and placed at position i in σ. Then, it appends to the label of each unvisited neighbor $x \in N(p)$ of p the letter $|V| - i$. The idea is then that the new vertex to visit is chosen among the vertices with the lexicographic largest label (see Subsection 2.2.2), from which the name Lexicographic Breadth-First Search.

Algorithm 4.1: Lex-BFS(G)

```
input: a graph $G = (V, E)$
output: a Lex-BFS ordering $\sigma$ of $V$

1 foreach $x \in V$ do
2     $label(x) = \varnothing$
3 for $i = 1, \ldots, |V|$ do
4     pick any unvisited vertex $p$ with lexicographical largest label
5     $\sigma(p) = i$ and mark $p$ as visited
6     foreach unvisited vertex $x$ in $N(p)$ do
7         append $|V| - i$ to $label(v)$
8 return $\sigma$
```

Hence, the priority queue of Lex-BFS is defined by ordering the unvisited vertices for decreasing lexicographic labels. The subset of unvisited vertices with the lexicographical largest label at each iteration, denoted here by S, represents the set of *tied* vertices in the graph search algorithm, and it is referred in the literature as *slice*. Then, we have ties if $|S| > 1$, in which case in Lex-BFS (Algorithm 4.1) we break them arbitrarily (line 4). We will see in the next section a variant of Lex-BFS where ties are broken using an additional linear order given in input. Note that, by construction, the vertices in the slice appear consecutively in σ. Furthermore, if the graph is disconnected then the vertices in each connected component are visited consecutively. In fact, it is easy to see that, if at some point during the algorithm each unvisited vertex has empty label, then the set of visited vertices is disconnected from the set of unvisited vertices.

We show below an example to show how Algorithm 4.1 works (Table 4.1) when applied to the (unit interval) graph in Figure 4.1. We have ties at the first iteration, where all the vertices have empty labels and are contained in the so-called *universal slice*, at the second iteration between vertices $2, 3, 5, 6$, and at the third iteration between vertices 3 and 5.

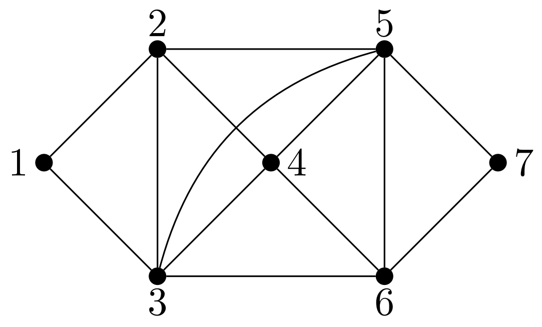


Figure 4.1: A unit interval graph.

Table 4.1: Values of the labels of each step of Lex-BFS (Algorithm 4.1) when applied to the graph in Figure 4.1.

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
4	∅						
2	∅	6					
3	∅	6	65				
5	∅	6	65	654			
6	∅	6	6	64	643		
1	∅	∅	5	54	54	54	
7	∅	∅	∅	∅	3	32	32

Recall that the Lex-BFS ordering denotes how the vertices in the graphs have been traversed. The following result characterizes Lex-BFS orderings by a 4-points condition, and it is valid for any arbitrary graph.

4.2.1 Theorem. *[102] A linear order σ of the vertices of a graph $G = (V, E)$ is a Lex-BFS ordering if and only if, for all vertices $a, b, c \in V$ such that $c <_\sigma b <_\sigma a$, $\{a, c\} \in E$ and $\{b, c\} \notin E$, then there exists a vertex $d \in V$ such that $d <_\sigma c$, $\{d, b\} \in E$ and $\{d, a\} \notin E$.*

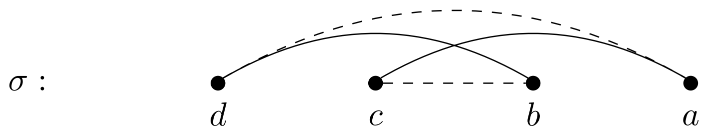


Figure 4.2: Lex-BFS ordering 4-points characterization of Theorem 4.2.1.

4.2.2 Lex-BFS$_+$

We present here a variant of Lex-BFS where ties are broken using an additional linear order σ given in input. First, we introduce the concept of 'good Lex-BFS'. A vertex z is *good* (or an *end-vertex*) if there exists some Lex-BFS ending at z. A Lex-BFS is *good* if every slice starts with a vertex that is good for the slice, i.e., there is a Lex-BFS ordering of the subgraph induced by the slice ending at that vertex.

Then, a well famous variant of Lex-BFS producing a good Lex-BFS is the one investigated by [107] and denoted by Lex-BFS$_+$, where the ties in the slice at line 4 in Algorithm 4.1 are broken choosing the vertex in the slice appearing last in σ. We denote by σ_+ the linear order returned by Lex-BFS$_+$. Hence, at the first iteration, the first vertex in σ_+ is actually the last vertex in σ.

There exist other variants of Lex-BFS in the literature. We refer the interested reader to [34] for more details. As we will see soon, Lex-BFS$_+$ plays an important role when defining multisweep Lex-BFS algorithms to recognize special graph classes, where the linear order σ given as input is actually the linear order produced by a previous Lex-BFS on the same set of vertices.

4.2.3 Partition refinement implementation

It is well known that, given a graph $G = (V, E)$, Lex-BFS can be implemented in linear time in the size of the graph, i.e., in $O(|V| + |E|)$ time (see, e.g., [37] and references therein). In this thesis we will follow the linear time implementation of Habib *et al.* [61], which uses the data structure based on the partition refinement paradigm presented in Subsection 2.2.3.

The motivation for this choice is that partition refinement will be the main routine used in the new recognition algorithms presented in Chapters 5 and 6.

Algorithm 4.2: Lex-BFS$_+(G,\sigma)$

```
input: a graph $G = (V, E)$ and a linear ordering $\sigma$ of $V$
output: a Lex-BFS ordering $\sigma_+$ of $V$

1 foreach $x \in V$ do
2     $label(x) = \varnothing$
3 for $i = 1, \dots, |V|$ do
4     let $S$ be the set of unvisited vertices with lexicographical largest label
5     let $p$ be the vertex in $S$ appearing last in $\sigma$
6     $\sigma(p) = i$ and mark $p$ as visited
7     foreach unvisited vertex $x$ in $N(p)$ do
8         append $|V| - i$ to $label(x)$
9 return $\sigma_+$
```

Table 4.2: Values of the labels of each step of Lex-BFS$_+$ (Algorithm 4.2) when applied to the graph in Figure 4.1 with $\sigma = (4, 2, 3, 5, 6, 1, 7)$.

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
7	$\varnothing$						
6	$\varnothing$	6					
5	$\varnothing$	6	65				
3	$\varnothing$	$\varnothing$	5	54			
4	$\varnothing$	$\varnothing$	5	54	543		
2	$\varnothing$	$\varnothing$	$\varnothing$	4	43	432	
1	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	3	3	31

Therefore, visualizing Lex-BFS as a partition refinement problem will be useful for their better understanding.

We thus introduce the equivalent Lex-BFS algorithm based on partition refinement (Algorithm 4.3) presented in [61], and we will show that it has linear time complexity. The partition refinement paradigm works as follows. Recall from Section 4.1 that any generic graph search algorithm maintains a priority queue of the unvisited vertices throughout the algorithm, which defines the priority to visit them. Then, at each iteration of Lex-BFS we maintain a priority queue, which is an ordered partition $\phi = (B_1, \dots, B_k)$ of the unvisited vertices ordered for decreasing labels. Each class of ϕ is called a *block* or a *cell*, and it defines a priority among the vertices, in the sense that if $x \in B_i, y \in B_j$ with $i < j$, then $label(x) > label(y)$ and thus $x <_\sigma y$, i.e., x will be visited before y. In the beginning of the algorithm, ϕ is initialized as a unique block corresponding to the whole ground set V (i.e., the universal slice).

The idea is then to choose as next vertex to visit (i.e., the pivot) the first vertex in the first block B_1 of ϕ, which represents the slice of the current iteration. Then, every time the pivot p is chosen, it is removed by its block and the ordered partition $\phi = (B_1 \setminus \{p\}, \dots, B_k)$ is refined according to the neighborhood $N(p)$ of p, i.e., each block B_j of ϕ is split in two blocks: the intersection $B_j \cap N(p)$ and the difference $B_j \setminus N(p)$, keeping only nonempty blocks. Equivalently, the (unvisited) vertices adjacent to p are placed in a new block in ϕ positioned before the original block containing them. This partition refinement procedure leads to a new ordered partition ϕ, whose first vertex will be chosen as pivot in the next iteration. The procedure is repeated until the priority queue ϕ is empty, i.e., all vertices have been visited.

Algorithm 4.3: Lex-BFS(G)

```
input: a graph $G = (V, E)$
output: a Lex-BFS ordering $\sigma$ of $V$

1  $\phi = (V)$
2  $i = 0$
3  while $\phi = (B_1, \dots, B_k) \neq \emptyset$ do
4      let $p$ be the first vertex in $B_1$
5      remove $p$ from $B_1$
6      $\sigma(p) = i$
7      $i = i + 1$
8      let $N(p)$ be the neighborhood of $p$
9      foreach class $B_j$ in $\phi$ do
10         if $B_j \cap N(p) \neq \emptyset$ and $|B_j \cap N(p)| \neq |B_j|$ then
11             replace $B_j$ by $(B_j \cap N(p), B_j \setminus N(p))$ in $\phi$
12 return $\sigma$
```

As we will see soon, for the implementation of Lex-BFS multisweep algorithms in Subsection 4.2.4, it will be convenient to assume that the vertices have always an initial ordering. Since in the classic Lex-BFS (Algorithm 4.1) the ties are broken arbitrarily, this choice can be done without loss of generality. To show concretely how Algorithm 4.3 works, we compute in Figures 4.3a and 4.3b the partition refinement procedures corresponding to the Lex-BFS orderings given in Tables 4.1 and 4.2 respectively.

We prove that Algorithm 4.3 can indeed be implemented in linear time in the size of the input graph. For the sake of clarity, we give below the complete proof following the argument in [61].

4.2.2 Theorem. *The Lex-BFS algorithm (Algorithm 4.3) applied to an (undirected) graph $G = (V, E)$ runs in $O(|V| + |E|)$ time.*

Proof. We assume that the graph is stored in an adjacency list and that we are given an initial order τ of V (e.g., the natural numbering of $[n]$). Furthermore, the vertices and their neighborhoods are ordered according to τ (in increasing order). This assumption will be useful for the discussion of the implementation of Lex-BFS$_+$ with the partition refinement paradigm.

To maintain the priority among the unvisited vertices, the queue ϕ is stored in a doubly linked list, whose elements are the classes $B_1, \ldots, B_k$. Moreover each vertex has a pointer to the class B_j containing it as well as a pointer to its position in the class, which are updated throughout the algorithm. This data structure permits constant time insertion and deletion of a vertex in ϕ.

Initially, the queue ϕ has only one class, namely the full set V (i.e., the universal slice). At a generic iteration i of Algorithm 4.3, we need to perform two operations: choose the next pivot $p = p_i$ to visit and update the queue ϕ of unvisited vertices.

1) The new pivot p is the first vertex (with respect of the ordering τ) in the first block B_1 of ϕ, and with the above described data structure it is easy to see that selecting the pivot p and removing it from ϕ can be implemented in $O(1)$ time.

2) The update of the queue ϕ is obtained by refining ϕ by $N(p)$. Specifically, we use an additional counter n_j for each block B_j which will denote the size of the intersection of $B_j \cap N(p)$ and it is initialized equal to zero. Then, for each unvisited vertex $x \in N(p)$, we remove it from its current position in B_j and we place it in position $n_j + 1$ in B_j, which can be done in $O(1)$ time as each vertex has a pointer to the position in the class containing it. Furthermore, we increase n_j by one and we keep track of the blocks whose intersection $B_j \cap N(p)$ is nonempty, i.e., the ones with $n_j > 0$. Hence, we can split each block B_j in $B_j \cap N(p)$ and $B_j \setminus N(p)$ by simply removing the first $n_j = |B_j \cap N(p)|$ elements of B_j and pushing them in a new block B'_j which we insert immediately before B_j in ϕ. Note that since V is ordered according to τ, the initial order τ in the new block B'_j is preserved. Hence, throughout the algorithm, the vertices in a common block of ϕ remain ordered according to τ. Once a vertex is relocated in ϕ, its pointers to the corresponding block and position in ϕ are updated accordingly. Given the above data structure, in order to perform all these operations we just need to pass through the neighborhood of the pivot p, which can be performed in $O(N(p))$ as the graph is represented with an adjacency list.

To conclude, the first task is repeated $|V|$ times (as each vertex is visited in the traversal algorithm exactly once), while the second task is repeated for $O(\sum_{i=1}^{p} N(p_i)) = O(|E|)$, leading to an overall complexity of $O(|V| + |E|)$. $\square$

Using the same data structure as in the proof of Theorem 4.2.2, we can show easily that Lex-BFS$_+$ (Algorithm 4.2) can be implemented as well in linear time using (the obvious variant of) Algorithm 4.3. In fact, the only difference between Lex-BFS and Lex-BFS$_+$ is the tie-breaking rule. Specifically, in Lex-BFS$_+$ in case of ties we choose as next pivot the vertex in the slice appearing last in the given order σ. We now show that this choice can be done in constant time, not affecting the complexity of Algorithm 4.3.

Recall that we assumed V to be initially ordered according to a linear order τ. We now select $\tau = \overline{\sigma}$, i.e., the reversal of σ. Then, since we showed that the initial order τ is always preserved in the classes of ϕ throughout the algorithm, we ensure that the first vertex in each slice S is exactly the vertex of S appearing first in τ, i.e., last in σ. Hence, the only thing we need to discuss is the complexity of reordering the adjacency list A according to τ. This can be done in $O(|V| + |E|)$ time as follows. We build a new adjacency list A' where the vertices are ordered according to τ: starting from the vertex appearing first in τ, for each vertex x in τ and for each $y \in N(x)$, we push $\tau(x)$ back in the list of A' corresponding to the neighbors of y. At the end of the process, each neighborhood in A' is then sorted according to τ. Since we explore each vertex and edge of A exactly once, the complexity is still $O(|V| + |E|)$.

4.2.4 Multisweep algorithms

Lex-BFS is widely applied in *multisweep* frameworks for recognizing many graph classes (see [38] and references therein). Roughly speaking, a multisweep algorithm is obtained by repeating several Lex-BFS's (or its variants) in a row. Each round is called a *sweep*, and it returns a linear order of the vertices which is then used in the next sweep to break ties in the slices encountered during the Lex-BFS algorithm.

Algorithm 4.4: *multisweep*$(G, \mathcal{C})$

```
input: a graph $G = (V, E)$ and a boolean condition $\mathcal{C}$
output: a linear order $\pi$ of the vertices $V$ of $G$ satisfying $\mathcal{C}$

1 $\sigma_0$ =Lex-BFS($G$)
2 $i = 0$
3 while $\mathcal{C}$ is FALSE do
4     $i = i + 1$
5     $\sigma_i$ =Lex-BFS$_+$($G, \sigma_{i-1}$)
6 return: $\sigma_i$
```

The multisweep algorithm terminates if some pre-established condition $\mathcal{C}$ is achieved (e.g., a maximum number of sweeps). In view of the complexity results in Section 4.2.3, it follows that any multisweep algorithm with k sweeps applied to a graph $G = (V, E)$ can be implemented in $O(k(|V| + |E|))$ time.

In our thesis, we are interested in the multisweep algorithm (Algorithm 4.4) where a first Lex-BFS sweep (Algorithm 4.1) is computed and then repeated Lex-BFS$_+$ sweeps (Algorithm 4.2) are performed. We will see in Section 4.3 some important combinatorial applications of Algorithm 4.4. Furthermore, this algorithm will be the inspiration for the new SFS multisweep algorithm to recognize Robinsonian matrices introduced in Chapter 6. We give below in Figure 4.3 an example of the first three sweeps of the multisweep algorithm applied to the graph in Figure 4.1.

$$\begin{array}{lll}
\boxed{4\ 2\ 3\ 5\ 6\ 1\ 7} & \boxed{7\ 1\ 6\ 5\ 3\ 2\ 4} & \boxed{1\ 2\ 4\ 3\ 5\ 6\ 7} \\
\mathbf{4}\ \boxed{2\ 3\ 5\ 6}\ \boxed{1\ 7} & \mathbf{7}\ \boxed{6\ 5}\ \boxed{1\ 3\ 2\ 4} & \mathbf{1}\ \boxed{2\ 3}\ \boxed{4\ 5\ 6\ 7} \\
4\ \mathbf{2}\ \boxed{3\ 5}\ \boxed{6}\ \boxed{1}\ \boxed{7} & 7\ \mathbf{6}\ \boxed{5}\ \boxed{3\ 4}\ \boxed{1\ 2} & 1\ \mathbf{2}\ \boxed{3}\ \boxed{4\ 5}\ \boxed{6\ 7} \\
4\ 2\ \mathbf{3}\ \boxed{5}\ \boxed{6}\ \boxed{1}\ \boxed{7} & 7\ 6\ \mathbf{5}\ \boxed{3\ 4}\ \boxed{2}\ \boxed{1} & 1\ 2\ \mathbf{3}\ \boxed{4\ 5}\ \boxed{6}\ \boxed{7} \\
4\ 2\ 3\ \mathbf{5}\ \boxed{6}\ \boxed{1}\ \boxed{7} & 7\ 6\ 5\ \mathbf{3}\ \boxed{4}\ \boxed{2}\ \boxed{1} & 1\ 2\ 3\ \mathbf{4}\ \boxed{5}\ \boxed{6}\ \boxed{7} \\
4\ 2\ 3\ 5\ \mathbf{6}\ \boxed{1}\ \boxed{7} & 7\ 6\ 5\ 3\ \mathbf{4}\ \boxed{2}\ \boxed{1} & 1\ 2\ 3\ 4\ \mathbf{5}\ \boxed{6}\ \boxed{7} \\
4\ 2\ 3\ 5\ 6\ \mathbf{1}\ \boxed{7} & 7\ 6\ 5\ 3\ 4\ \mathbf{2}\ \boxed{1} & 1\ 2\ 3\ 4\ 5\ \mathbf{6}\ \boxed{7}
\end{array}$$

(a) first sweep σ_0. (b) second sweep σ_1. (c) third sweep σ_2.

Figure 4.3: Iterations of Lex-BFS multisweep algorithm with partition refinement paradigm (Algorithm 4.4) applied to the graph in Figure 4.1. We indicate in bold the pivot which is chosen at the current iteration.

4.3 Structural properties

In this section we present some important properties for the Lex-BFS structure of chordal, interval and unit interval graphs. Most of these results can be found in [34] and [38], and they will be rephrased in the discussion of the SFS algorithm in Chapter 6. We introduce some notation.

Recall from Section 2.2 that a vertex z is *simplicial* if its neighbors are pairwise adjacent. Furthermore, a vertex z *intercepts* a path P if z is adjacent to at least one vertex in P; otherwise, z is said to *miss* P. Two vertices x, y are *unrelated* with respect to z if there exists a path P from x to z and a path Q from y to z such that y misses P and x misses Q. A vertex z is then said to be *admissible* if there do not exist two distinct vertices in G that are unrelated with respect to z. If a vertex is both simplicial and admissible, then it is called *valid*.

4.3.1 Chordal and interval graphs

We give some useful results when Lex-BFS (Algorithm 4.1) and Lex-BFS$_+$ (Algorithm 4.2) are applied to chordal and interval graphs.

Chordal graphs [102] Recall that a graph is a chordal graph if and only if it has a perfect elimination ordering (see Section 2.3). A famous result is that if G is chordal, then any Lex-BFS ordering is a perfect elimination ordering (this was in fact the motivation for Rose and Tarjan [102] to introduce the Lex-BFS algorithm). Therefore, one can test whether a graph is chordal by simply running one Lex-BFS sweep in Algorithm 4.4 and checking if it is a perfect elimination order, which can be done in linear time. The main results leading to this conclusion are the following.

4.3.1 Theorem. ***(The* $\mathbf{P_3}$ *Rule)****[102] Let σ be a Lex-BFS ordering of a chordal graph $G = (V, E)$. Let $x, y, z \in V$ with $\{x, z\}, \{y, z\} \in E$ and such that $x <_\sigma z$ and $y <_\sigma z$. Then $\{x, y\} \in E$.*

4.3.2 Theorem. ***(The Chordal Lex-BFS Theorem)****[38] Let $G = (V, E)$ be a chordal graph and let S be a slice of an arbitrary Lex-BFS ordering τ of V. Furthermore, let σ be an arbitrary Lex-BFS ordering of V. Then $\sigma[S]$ is a Lex-BFS ordering of $G[S]$.*

Interval graphs [34],[38] Interval graphs can be recognized using six sweeps of Lex-BFS in Algorithm 4.4. Specifically, in addition to a first Lex-BFS sweep (Algorithm 4.1), four additional Lex-BFS$_+$ sweeps (Algorithm 4.2) and a final Lex-BFS$_*$ [2] sweep are needed.

We give below some useful results which will be extended in Chapter 6 when defining the SFS algorithm to recognize Robinsonian matrices. Recall from Section 4.2.2 that a vertex is good if it is the last vertex of some Lex-BFS ordering of G.

4.3.3 Lemma. *[38] Let $G = (V, E)$ be an interval graph. Then a vertex is good if and only if it is valid (i.e., simplicial and admissible).*

The the following result shows a 'flipping' property of the end points of two consecutive Lex-BFS sweeps.

[2] Lex-BFS$_*$ is another variant of Lex-BFS, breaking ties according to the Lex-BFS orderings of two previous sweeps. We do not discuss this variant in the thesis, but we refer the interested reader to [38] for more details.

4.3.4 Theorem. *[38] Let $G = (V, E)$ be an interval graph and let a and b be good vertices of G. If σ is an arbitrary Lex-BFS of G starting at a and ending at b, then Lex-BFS$_+$(G,σ) is a good Lex-BFS starting at b and ending at a.*

4.3.2 Unit interval graphs

We finally discuss some properties of Lex-BFS when applied to unit interval graphs, which are a subclass of chordal graphs and interval graphs. Unit interval graphs can be recognized in three sweeps of Lex-BFS in Algorithm 4.4. Specifically, we run a first Lex-BFS sweep (Algorithm 4.1), denoted by σ, and two additional Lex-BFS$_+$ sweeps (Algorithm 4.2), denoted respectively by σ_+ and σ_{++}. Then G is a unit interval graph if and only if σ_{++} satisfies the 'neighborhood condition' *(iv)* in Theorem 2.3.3 (see [33]). To show the correctness of the above algorithm, Corneil [33] outlined some important structural results of two consecutive good Lex-BFS orderings on unit interval graphs.

Since Lex-BFS is a special BFS, one can define the BFS-layered structure $L_0, \dots, L_k$ for a given graph G as follows. If BFS starts at vertex a, then $L_0 = \{a\}$, which is called the *root*, and a vertex x belongs to the layer L_i if it has a distance of i from the root L_0, i.e., if there exists a path of length i from x to L_0 but no shorter path exists. In our case, the root is the first vertex of a Lex-BFS ordering. Hence, we denote by L_i^+ and L_i^{++} the layer structures rooted, respectively, at the first vertex of σ_+ and the first vertex of σ_{++} (see Figure 4.4). In short, we say that L_i^+ is the layer structure of σ_+, meaning that it is in fact the layer structure rooted at the first vertex of σ_+.

Suppose now that σ_+ starts with a and end with b. Then σ_{++} starts with b and ends with a (Theorem 4.3.4). Hence, $L_0^+ = \{a\}$ and $L_0^{++} = \{b\}$. Let k be the distance between a and b; we denote by P the set of vertices that are on a shortest path from a to b (for example, in Figure 4.4 we have $P = V \setminus \{4\}$). Then, for unit interval graphs the following facts hold:

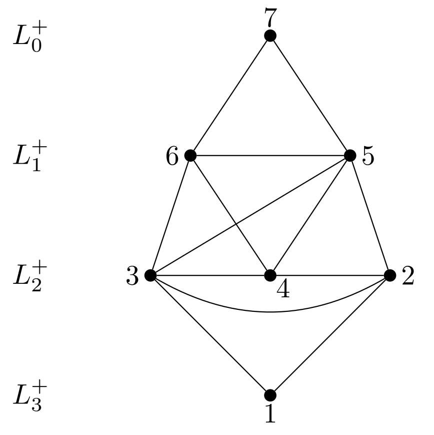
4.3.5 Lemma. *[35] For each $0 \leq i \leq k$, L_i^+, L_i^{++} is a clique of G, and*

$$x \in L_i^{++} \Rightarrow x \in \begin{cases} L_{k-i}^+ & \textit{if } x \in P, \\ L_{k-i+1}^+ & \textit{if } x \notin P. \end{cases}$$

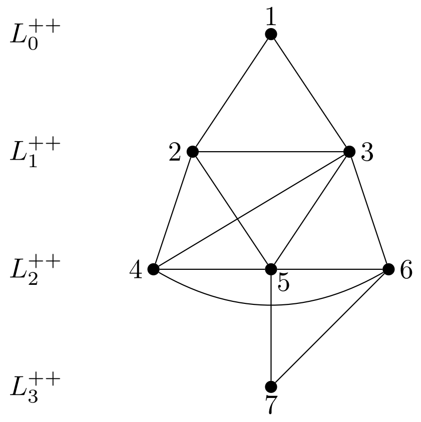
In other words, the above result 'extends' the flipping result of Theorem 4.3.4 also to the other vertices of the graph (not only the end points). Note that in the example in Figure 4.3 the third sweep $\sigma_{++} = \sigma_2$ satisfies the 'neighborhood condition'. Another way to see that G is a unit interval graph is to consider its extended adjacency matrix A ordered according to $\sigma_{++} = \sigma_2$ as shown below:

$$A = \begin{array}{c|ccccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \\ \hline \mathbf{1} & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \mathbf{2} & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ \mathbf{3} & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \mathbf{4} & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \mathbf{5} & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{6} & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{7} & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array} \tag{4.1}$$

Since A has consecutive ones in rows and columns, this implies that G is indeed a unit interval graph (see Section 2.2 and Subsection 3.2.1).



(a) BFS layers of σ_+.



(b) BFS layers of σ_{++}.

Figure 4.4: BFS layers of the second and third sweeps $\sigma_+ = \sigma_1$ and $\sigma_{++} = \sigma_2$ of the multisweep algorithm (Algorithm 4.4) applied to the graph in Figure 4.1.

4.4 Conclusions

In this chapter we discussed Lexicographic Breadth-First search (Lex-BFS), a variant of Breadth-First Search (BFS) which is used in multisweep algorithms for the recognition of several classes of graphs. Specifically, we have seen that chordal graphs can be recognized in one Lex-BFS sweep, interval graphs can be recognized in six sweeps and unit interval graphs in three sweeps. Furthermore, other graph classes not discussed in this thesis can be recognized in a finite number of Lex-BFS sweeps. Specifically, cographs can be recognized in three sweeps [16] and co-comparability graphs can be recognized in n sweeps [51].

Lex-BFS will play a predominant role in this thesis, as the concepts and results discussed here will be fundamental for the understanding of the Robinsonian recognition algorithms presented in Chapters 5 and 6. Specifically, in Chapter 5 we will answer an open question posed by M. Habib *et al.* at the PRIMA Conference in Shanghai in June 2013. Namely, the authors asked whether it is possible to use Lex-BFS+ to recognize Robinsonian matrices (see [39]). In this context, Lex-BFS will be used to compute straight enumerations of unit interval graphs (see Theorem 5.2.2).

Then, in Chapter 6 we will introduce a generalization of Lex-BFS to weighted graphs, named Similarity-First Search (SFS), and we will introduce a multi-sweep SFS algorithm to recognize Robinsonian matrices. In order to prove the correctness of the algorithm, we will also introduce an analogous concept of BFS layers, which were exploited for unit interval graphs in Subsections 4.3.2, namely 'similarity layers'. Since SFS reduces to Lex-BFS when applied to unweighted graphs and since Robinsonian matrices coincide with the class of (adjacency matrices of) unit interval graphs (Theorem 3.2.1), we will provide an alternative proof of correctness for the 3-sweep recognition algorithm for unit interval graphs of Corneil in [33], exploiting the structure of such 'similarity layers' (see Theorem 6.5.2).

Part II

Recognition algorithms for Robinsonian matrices

5 Lex-BFS based algorithm

In this chapter we introduce a new combinatorial algorithm to recognize Robinsonian matrices based on straight enumerations of unit interval graphs. In Section 5.1 we give a short overview of the recognition algorithm and the motivation for our work. In Section 5.2 we recall the characterization of Robinsonian matrices in terms of straight enumerations of unit interval graphs, which will be the base for the recursive algorithm discussed in this chapter. In Section 5.3 we present the main subroutines constituting our recursive algorithm, which we discuss in Section 5.4. In Section 5.5 we show how to return all Robinson orderings of a given Robinsonian similarity matrix. In Section 5.6 we illustrate the algorithm on an example. The final Section 5.7 contains some questions for possible future work. This chapter is based on our work [77].

5.1 Introduction

In Subsection 3.2.2 we have seen that a similarity matrix A is Robinsonian if and only if it can be decomposed as a conic combination of (adjacency matrices of) compatible unit interval graphs (Lemma 3.2.5). Recall that for a graph $G = (V, E)$, *straight enumerations* are special orderings of the classes of the ‘undistinguishability’ equivalence relation on V (see Definition 2.3.4), and that G is a unit interval graph precisely when it admits a straight enumeration (Theorem 2.3.5). In this chapter we will exploit this characterization to define a new (recursive) recognition algorithm for Robinsonian similarities matrices.

Our approach differs from the existing ones in the sense that it is not directly related to interval (hyper)graphs, but it relies only on unit interval graphs (which are a simpler graph class than interval graphs) and on their straight enumerations. Furthermore, our algorithm does not rely on any sophisticated external algorithm such as the Booth and Leuker algorithm for C1P. In fact, the most difficult task carried out by our algorithm is the Lex-BFS algorithm and the variant Lex-BFS$_+$ introduced in [107] to compute straight enumerations (both described in Chapter 4). Our algorithm uses a divide-and-conquer strategy with a merging step, tailored to efficiently exploit the possible sparsity structure of the given similarity matrix A. Assuming that the matrix A is given as an adjacency list of an undirected weighted graph, our algorithm runs in $O(d(m+n))$ time, where n is the size of A, m is the number of nonzero entries of A and d is the depth of the recursion tree computed by the algorithm, which is upper bounded by the number L of distinct nonzero entries of A (see Theorem 5.4.4). Furthermore, we can return all Robinson orderings of A using the PQ-tree data structure introduced in Subsection 2.2.4, on which we perform only a few simple operations (see Section 5.5).

Our algorithm uncovers an interesting link between straight enumerations of unit interval graphs and Robinsonian matrices which, to the best of our knowledge, has not been made before. Moreover it provides an answer to an open question posed by M. Habib at the PRIMA Conference in Shanghai in June 2013, who asked whether it is possible to use Lex-BFS$_+$ to recognize Robinsonian matrices [39]. Alternatively one could check whether the incidence matrix M of the ball hypergraph of A has C1P (Theorem 3.2.3), using the Lex-BFS based algorithm of [61] in $O(r+c+f)$ time if M is $r \times c$ with f ones. As $r \leq nL$, $c = n$ and $f \leq Lm$, the overall time complexity is $O(L(n+m))$. Interestingly, this approach is not mentioned by Habib. In comparison, an advantage of our approach is that it exploits the sparsity structure of the matrix A, as d can be smaller than L.

5.2 Preliminaries

In this section we will reintroduce some notations and concepts already presented in Chapter 2. Given a linear order $\pi = (x_1, \dots, x_n)$ of $[n]$, then $\overline{\pi} = (x_n, \dots, x_1)$ denotes its *reversal*. For $U \subseteq [n]$, $\pi[U]$ denotes the restriction of π to U. If π_1 and π_2 are two linear orders on disjoint subsets V_1 and V_2, then $\pi = (\pi_1, \pi_2)$ denotes their concatenation, which is a linear order on $V_1 \cup V_2$.

Recall that the blocks of a graph are the (disjoint) subsets of vertices with the same closed neighborhood (see Section 2.3). Then, a *straight enumeration* of a unit interval graph $G = (V = [n], E)$ is an ordered partition $\phi = (B_1, \dots, B_p)$ of the blocks of G such that, for any block B_i, the block B_i and the blocks B_j adjacent to it are consecutive in the linear order. The blocks B_1 and B_p are called the *end blocks* of ϕ and B_i (with $1 < i < p$) are its *inner blocks* (see Definition 2.3.4). The *reversal* of ϕ is the ordered partition $\overline{\phi} = (B_p, \dots, B_1)$. For $U \subseteq V$, $\phi[U] = (B_1 \cap U, \dots, B_p \cap U)$ denotes the *restriction* of the weak linear order ϕ to U. Note that in this chapter we may use the term block also to denote the class of an ordered partition which does not necessarily represents a straight enumeration.

We say that two weak linear orders ψ and ϕ of $V = [n]$ are *compatible* if they admit a compatible linear order π, i.e., there do not exist elements $x, y \in V$ such that $x <_\psi y$ and $y <_\phi x$. Then their *common refinement* is the weak linear order $\Phi = \psi \wedge \phi$ on V defined by $x =_\Phi y$ if $x =_\psi y$, $x =_\phi y$, and by $x <_\Phi y$ if $x \leq_\psi y$, $x \leq_\phi y$ with at least one strict inequality. In what follows, the weak linear orders ψ and ϕ will correspond (roughly speaking) to the straight enumerations of the level graphs of A.

Given a matrix $A \in \mathcal{S}^n$, for $U \subseteq V$, $A[U] = (A_{ij})_{i,j \in U}$ is the principal submatrix of A indexed by U. For a permutation π of V, $A_\pi = (A_{\pi(i),\pi(j)})_{i,j=1}^n$ is the matrix obtained by symmetrically permuting the rows and columns of A according to π. As already underlined in Chapter 3, also in this chapter we will deal exclusively with Robinson(ian) similarities. Hence, when speaking of a Robinson(ian) matrix, we mean a Robinson(ian) similarity matrix. We may assume without loss of generality that the given similarity matrix is nonnegative.

Recall that 0/1 Robinsonian matrices coincide with the class of (adjacency matrices of) unit interval graphs (Theorem 3.2.1). Furthermore, in view of Theorem 2.3.5, a graph G is a unit interval graph if and only if it has a straight enumeration. Therefore, a first natural step is to characterize 0/1 Robinsonian matrices in terms of straight enumerations of unit interval graphs. The next result is simple but will play a central role in our algorithm for recognizing Robinsonian similarities.

5.2.1 Theorem. *Let $G = (V = [n], E)$ be a graph and let A_G be the corresponding (extended) adjacency matrix. Then, a linear order π of V is a Robinson ordering of A_G if and only if there exists a straight enumeration of G whose corresponding weak linear order ψ is compatible with π, i.e., satisfies:*

$$\forall x, y \in V \quad \text{with} \quad x \neq_\psi y \qquad x <_\pi y \quad \Longleftrightarrow \quad x <_\psi y. \tag{5.1}$$

Proof. Assume that π is a linear order of V that reorders A_G as a Robinson matrix. Then it is easy to see that the 3-vertex condition *(iii)* in Theorem 2.3.3 holds for π and that each block of G is an interval w.r.t. π. Therefore the order π induces an order ψ of the blocks: $B_1 <_\psi \dots <_\psi B_p$, with $B_i <_\psi B_j$ if and only if $x <_\pi y$ for all $x \in B_i$ and $y \in B_j$. In other words, ψ is compatible with π by construction. Moreover, ψ defines a straight enumeration of G. Indeed, if $B_i <_\psi B_j <_\psi B_k$ and B_i, B_k are adjacent then B_j is adjacent to B_i and B_k, since this property follows directly from the 3-vertex condition for π.

Conversely, assume that $B_1 <_\psi \ldots <_\psi B_p$ is a straight enumeration of G and let π be a linear order of V which is compatible with ψ, i.e., satisfies (5.1). We show that π reorders A_G as a Robinson matrix. That is, we show that if $x <_\pi y <_\pi z$, then $(A_G)_{xz} \le \min\{(A_G)_{xy}, (A_G)_{yz}\}$ or, equivalently, that $\{x,z\} \in E$ implies $\{x,y\}, \{y,z\} \in E$. If x, z belong to the same block B_i then $y \in B_i$ (using (5.1)) and thus $\{x,y\}, \{y,z\} \in E$ since B_i is a clique. Assume now that $x \in B_i$, $z \in B_k$ and $\{x,z\} \in E$. Then, $B_i <_\psi B_k$ and B_i, B_k are adjacent blocks and thus $B_i \cup B_k$ is a clique. If $y \in B_i$ then y is adjacent to x and z (since $B_i \cup B_k$ is a clique). Analogously if $y \in B_k$. Suppose now that $y \in B_j$. Using (5.1), we have that $B_i <_\psi B_j <_\psi B_k$. As ψ is a straight enumeration with B_i,B_k adjacent then B_j is adjacent to B_i and to B_k and thus y is adjacent to x and z. $\square$

Hence, in order to find the permutations reordering a given 0/1 matrix A as a Robinson matrix, it suffices to find all the possible straight enumerations of the corresponding graph G. As is shown e.g., in [35, 43], this is a simple task and can be done in linear time. This is coherent with the fact that C1P can be checked in linear time (see Subsection 3.2.1).

We now consider a general (non necessarily 0/1) matrix A. Let $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_L$ denote the distinct values taken by the entries of A. Recall that we denoted by $G^{(\ell)} = (V, E_\ell)$ the *ℓ-th level graph* of A, i.e., the graph whose edges are the pairs $\{x,y\}$ with $A_{xy} \ge \alpha_\ell$. As stated in Theorem 3.2.5, the level graphs can be used to decompose A as a conic combination of 0/1 matrices and, as already observed by Roberts [99], A is Robinson precisely when these 0/1 matrices are Robinson.

Combining the links between reordering 0/1 Robinsonian matrices and straight enumerations of unit interval graphs (Theorem 5.2.1) together with the decomposition result of Lemma 3.2.5, we obtain the following new characterization of Robinsonian matrices.

5.2.2 Theorem. *Let $A \in \mathcal{S}^n$ with level graphs $G^{(1)}, \ldots, G^{(L)}$. Then:*

(i) A is a Robinsonian matrix if and only if there exist straight enumerations of its level graphs $G^{(1)}$, ..., $G^{(L)}$ whose corresponding weak linear orders $\psi_1, \ldots, \psi_L$ are pairwise compatible.

(ii) A linear order π of V reorders A as a Robinson matrix if and only if there exist pairwise compatible straight enumerations of $G^{(1)}, \ldots, G^{(L)}$ whose corresponding common refinement is compatible with π.

Proof. Observe first that if assertion (ii) holds then (i) follows directly using the result of Lemma 2.2.2. We now prove (ii). Assume that A is Robinsonian and let π a Robinson ordering of A. Then A_π is Robinson and thus, by lemma 3.2.5, each permuted matrix $(A_{G^{(\ell)}})_\pi$ is a Robinson matrix. Then, applying Theorem 5.2.1, for each $\ell \in [L]$, there exists a straight enumeration of $G^{(\ell)}$ whose corresponding weak linear ordering ψ_ℓ is compatible with π. We can thus conclude that the common refinement of $\psi_1, \ldots, \psi_L$ is compatible with π in view of Lemma 2.2.2. Conversely, assume that there exist straight enumerations of $G^{(1)}, \ldots, G^{(L)}$ whose corresponding weak linear orders $\psi_1, \ldots, \psi_L$ are pairwise compatible and their common refinement is compatible with π. Then, by Theorem 5.2.1, π reorders simultaneously each $A_{G^{(\ell)}}$ as a Robinson matrix and thus A_π is Robinson, which shows that A is Robinsonian. $\square$

5.3 Subroutines

We describe here the main subroutines which will be used in the recursive algorithm for recognizing whether a given symmetric nonnegative matrix A is Robinsonian. The algorithm is based on Theorem 5.2.2. Recall that a straight enumeration is nothing but a special ordered partition of the vertices of a graph (see Definition 2.3.4), and two straight enumerations are compatible if there exists a linear order compatible with the ordered partitions induced by both straight enumerations. Hence, the main idea for the recognition algorithm is to find straight enumerations of the level graphs of A that are pairwise compatible and to compute their common refinement. The matrix A is not Robinsonian precisely when these objects cannot be found.

One of the main tasks in the algorithm is to find (if it exists) a straight enumeration of a graph G which is compatible with a given weak linear order ψ of V. Roughly speaking, G will correspond to a level graph $G^{(\ell)}$ of A (in fact, to a connected component of it), while ψ will correspond to the common refinement of the previous level graphs $G^{(1)}, \ldots, G^{(\ell-1)}$. Hence, looking for a straight enumeration of G compatible with ψ will correspond to looking for a straight enumeration of $G^{(\ell)}$ compatible with previously selected straight enumerations of the previous level graphs $G^{(1)}, \ldots, G^{(\ell-1)}$.

There are three main subroutines in our algorithm, each discussed in an independent subsection. In Subsection 5.3.1 we describe *CO-Lex-BFS* (see Algorithm 5.1), a variation of Lex-BFS, which finds and orders the connected components of the level graphs. In Subsection 5.3.2 we describe *Straight_enumeration* (see Algorithm 5.2), which computes the straight enumeration of a connected graph as in [33]. Finally, in Subsection 5.3.3 we present *Refine* (see Algorithm 5.3), a variation of partition refinement (Algorithm 2.1), which finds the common refinement of two weak linear orders.

5.3.1 Component ordering

Our first subroutine is *CO-Lex-BFS* (where CO stands for 'Component Ordering'), presented in Algorithm 5.1. The algorithm is an extension of Lex-BFS (Algorithm 4.1) to detect connected components of a graph. Namely, given a graph $G = (V, E)$ and a weak linear order ψ on V, it detects the connected components of G and orders them in a compatible way with respect to ψ. According to Lemma 5.3.1 below, this is possible if G admits a straight enumeration compatible with ψ.

5.3.1 Lemma. *Consider a graph $G = (V, E)$ and a weak linear order ψ of V. If G has a straight enumeration ϕ compatible with ψ then there exists an ordering $V_1, \ldots, V_c$ of the connected components of G which is compatible with ψ, i.e., such that $V_1 \leq_\psi \ldots \leq_\psi V_c$.*

Proof. If $V_1, \ldots, V_c$ is the ordering of the components of G which is induced by the straight enumeration ϕ, i.e., $V_1 <_\phi \ldots <_\phi V_c$, then $V_1 \leq_\psi \ldots \leq_\psi V_c$ as ϕ is compatible with ψ. $\square$

Roughly speaking, the algorithm uses Lex-BFS to detect the connected components of a graph G and build throughout an ordering of the components which is compatible with a given weak linear order ψ. To do so, every time a new connected component V_ω is found, the algorithm computes the intersection of V_ω with the blocks of ψ (in fact, only with the first block $B_\omega^{\min}$ and last block $B_\omega^{\max}$ of ψ meeting V_ω), and check if V_ω can be ordered along the already detected components compatibly with ψ.

In order to detect connected components using Lex-BFS, we proceed as follows. As in the classic Lex-BFS (Algorithm 4.1), at each iteration we are given an ordered partition representing the queue of unvisited vertices. The first block of the queue is called *slice*, denoted by S, and represents the subset of unvisited vertices with largest lexicographic label. Then, we choose as new pivot p (i.e., the next vertex to visit) a vertex in the slice S arbitrarily. In the normal Lex-BFS we would now update the queue of unvisited vertices according to the neighborhood $N(p)$ of the pivot p. In *CO-Lex-BFS* we have an additional step, which is based on the following observations. When the vertex p in the set S at line 7 of Algorithm 5.1 has label $\varnothing$, it means that p is not contained in the current component V_ω. In view of the Lex-BFS property (see Theorem 4.2.1), this in turns implies that the unvisited vertices and the visited vertices are disconnected. Hence, a new connected component is found and initialized with p. Every time a connected component V_ω is detected we, check if it can be ordered along the already detected components in a compatible way with ψ. Let $B_\omega^{\min}$ and $B_\omega^{\max}$ denote respectively the first and the last block of ψ intersecting the connected component V_ω. We distinguish two cases:

1. if V_ω meets more than one block of ψ (i.e., if $B_\omega^{\min} <_\psi B_\omega^{\max}$), we check if all the inner blocks between $B_\omega^{\min}$ and $B_\omega^{\max}$ are contained in V_ω. If this is not the case, then the algorithm stops. Moreover the algorithm also stops if both V_ω and $V_{\omega-1}$ meet exactly the same two blocks, i.e., $B_\omega^{\min} = B_{\omega-1}^{\min}$ and $B_\omega^{\max} = B_{\omega-1}^{\max}$. In both cases it is indeed not possible to order the components in a compatible way with ψ.

2. if V_ω meets only one block B_k of ψ (i.e., $V_\omega \subseteq B_k$) and if moreover this block B_k is the first block of the previous connected component $V_{\omega-1}$ (i.e., $B_k = B_{\omega-1}^{\min}$), then we swap $V_{\omega-1}$ and V_ω in order to make the ordering of the components compatible with ψ. The ordering σ is updated by setting, for each $v \in V_{\omega-1}$ its new ordering as $\sigma(v) + |V_\omega|$ and for each $v \in V_\omega$ as $\sigma(v) - |V_{\omega-1}|$. Observe however that if we are in the case when both V_ω and $V_{\omega-1}$ are contained in B_k, then we do not need to do this swap, i.e., the two components V_ω and $V_{\omega-1}$ can be ordered arbitrarily.

Algorithm 5.1: *CO-Lex-BFS*(G, ψ)

```
input: a graph $G = (V, E)$, a weak linear order $\psi = (B_1, \dots, B_p)$ of $V$
output: a linear order $\sigma$ of $V$ and a linear order $(V_1, \dots, V_c)$ of the
        connected components of $G$ compatible with $\psi$ and $\sigma$, or STOP
        (no such linear order of the components exists)

mark all the vertices as unvisited
$\omega = 1$
$V_\omega, B_\omega^{\min}, B_\omega^{\max} = \emptyset$
foreach $v \in V$ do
    $label(v) = \varnothing$
for $i = 1, \dots, |V|$ do
    let $S$ be the set of unvisited vertices with lexicographically largest label
    pick arbitrarily a vertex $p$ in $S$ and mark it as visited
    $\sigma(p) = |V| + 1 - i$
    if $label(p) = \varnothing$ then
        if $V_\omega \subseteq B_{\omega-1}^{\min}$ then
            swap $V_\omega$ and $V_{\omega-1}$ and modify $\sigma$ accordingly
        else
            if $B_\omega^{\min} <_\psi B_{\omega-1}^{\max}$ or if there exists a block $B$ of $\psi$ such that
            $B \nsubseteq V_\omega$ and $B_\omega^{\min} <_\psi B <_\psi B_\omega^{\max}$ then
                stop (no ordering of components compatible with $\psi$ exists)
        $\omega = \omega + 1$
        $V_\omega = \emptyset$
    $V_\omega = V_\omega \cup \{p\}$
    $B_\omega^{\min}$ is the first block in $\psi$ which meets $V_\omega$
    $B_\omega^{\max}$ is the last block in $\psi$ which meets $V_\omega$
    foreach unvisited vertex $w$ in $N(p)$ do
        append $|V| - i$ to $label(w)$
return $(V_1, \dots, V_c)$ and $\sigma$
```

The next lemma shows the correctness of Algorithm 5.1.

5.3.2 Lemma. *Let $G = (V, E)$ be a graph and let ψ be a weak linear order of V.*

(i) If Algorithm 5.1 successfully terminates then the returned order $V_1, \dots, V_c$ of the components satisfies $V_1 \leq_\psi \;\dots\; \leq_\psi V_c$.

(ii) If Algorithm 5.1 stops then no ordering of the components exists that is compatible with ψ.

Proof. (i) Assume first that Algorithm 5.1 successfully terminates and returns the linear ordering $V_1, \dots, V_c$ of the components. Suppose for contradiction that $V_{\omega-1} \not\leq_\psi V_\omega$ for some $\omega \in [c]$. Then there exist $x \in V_{\omega-1}$ and $y \in V_\omega$ such that $y <_\psi x$. Let z be the first vertex selected in the component $V_{\omega-1}$. Then, $z \leq_\psi y$ (for if not the algorithm would have selected y before z when opening the component $V_{\omega-1}$). Let $\psi = (B_1, \dots, B_p)$ and let denote by B_ω^{min} and B_ω^{max} respectively the first and last block of ψ meeting V_ω ($B_{\omega-1}^{min}$ and $B_{\omega-1}^{max}$ are analogously defined). Say $x \in B_j$, $y \in B_i$ so that $i < j$, and $z \in B_r$. As $z \leq_\psi y$, we have $B_r \leq_\psi B_i$. Suppose first that $B_r <_\psi B_i$. Then, B_i is an inner block between $B_{\omega-1}^{\min}$ and $B_{\omega-1}^{\max}$ which is not contained in $V_{\omega-1}$ (since $y \in B_i$), yielding a contradiction since the algorithm would have stopped when dealing with the component $V_{\omega-1}$. Suppose now that $B_r = B_i$. If $\psi[V_\omega]$ has only one block B, then $B \subseteq B_i = B_{\omega-1}^{\min}$ and then the algorithm would have swapped V_ω and $V_{\omega-1}$. Hence $\psi[V_\omega]$ has at least two blocks and $B_\omega^{\min} \leq_\psi B_i <_\psi B_j \leq_\psi B_{\omega-1}^{\max}$, which is again a contradiction since the algorithm would have stopped.
(ii) Assume now that the algorithm stops when dealing with the component V_ω. Then $\psi[V_\omega]$ has at least two blocks. Suppose first that the algorithm stops because $B_\omega^{\min} <_\psi B_{\omega-1}^{\max}$. Then clearly one cannot have $V_{\omega-1} <_\psi V_\omega$. We show that we also cannot have $V_\omega <_\psi V_{\omega-1}$. For this assume for contradiction that $V_\omega <_\psi V_{\omega-1}$. Let y be the first selected vertex in V_ω and let x be the first vertex selected in $V_{\omega-1}$. Then, $y \in B_\omega^{\min}$, $x \leq_\psi y$ (for if not the algorithm would have considered the component V_ω before $V_{\omega-1}$), and thus $B_{\omega-1}^{\min} \leq_\psi B_\omega^{\min}$. If $B_{\omega-1}^{\min} <_\psi B_\omega^{\min}$ then the algorithm would have stopped earlier when examining $V_{\omega-1}$, since $B_{\omega-1}^{\min} <_\psi B_\omega^{\min} <_\psi B_{\omega-1}^{\max}$ and $B_\omega^{\min} \not\subseteq V_{\omega-1}$. Hence, we have $B_{\omega-1}^{\min} = B_\omega^{\min}$ and, as $\psi[V_\omega]$ has at least two blocks, there exists a vertex $z \in B_\omega^{\max}$ such that $x <_\psi z$, which contradicts $V_\omega <_\psi V_{\omega-1}$. Suppose now that the algorithm stops because $B_\omega^{\min} <_\psi B <_\psi B_\omega^{\max}$ and $B \not\subseteq V_\omega$. Let $x \in B_\omega^{\min}, y \in B_\omega^{\max}$ and $z \in B \setminus V_\omega$, and say $z \in V_{\omega'}$. Then we cannot have $V_{\omega'} <_\psi V_\omega$ since $x <_\psi z$, and we also cannot have $V_\omega <_\psi V_{\omega'}$ since $z <_\psi y$. Hence the two components V_ω and $V_{\omega'}$ cannot be ordered compatibly with ψ and this concludes the proof. □

5.3.2 Straight enumeration

Once the connected components of G are ordered, we need to compute a straight enumeration of each connected component $G[V_\omega]$. We do this with the routine

Straight_enumeration applied to $(G[V_\omega], \sigma_\omega)$, where σ_ω is a suitable given order of V_ω (namely, $\sigma_\omega = \sigma[V_\omega]$, where σ is the vertex order returned by *CO-Lex-BFS*(G, ψ)). This routine is essentially the 3-sweep unit interval graph recognition algorithm of Corneil [33] (see Subsection 4.3.2). The only difference of *Straight_enumeration*$(G[V_\omega], \sigma_\omega)$ with respect to Corneil's algorithm is that we save the first sweep, because we use the order σ_ω returned by *CO-Lex-BFS*. We now describe the routine *Straight_enumeration* which is based on the algorithms of [35, §3] and [33, §2]. Below, $\deg_G(v)$ denotes the degree of the vertex v in G.

Algorithm 5.2: *Straight_enumeration*(G, σ)

```
input: a connected graph $G = (V, E)$ and a linear order $\sigma$ of $V$
output: a straight enumeration $\phi$ of $G$, or STOP ($G$ is not a unit interval
        graph)

1  $\sigma_+ = \text{Lex-BFS}_+(G, \sigma)$
2  $\sigma_{++} = \text{Lex-BFS}_+(G, \sigma_+)$
3  $i = 0$                                        (index of the blocks of $\psi$)
4  $L = R = 0$                  (dummy variables to record the current block $B_i$)
5  for $v = 1, \ldots, |V|$ do
6      $lmn(v) = \min\{u : u \in N[v]\}$        (leftmost vertex adjacent to $v$ in $\sigma_{++}$)
7      $rmn(v) = \max\{u : u \in N[v]\}$    (rightmost vertex adjacent to $v$ in $\sigma_{++}$)
8      if $rmn(v) - lmn(v) \neq \deg_G(v)$ then
9          stop                                  ($G$ is not a unit interval graph)
10     else
11         if $lmn(v) = L$ and $rmn(v) = R$ then
12             $C_i = C_i \cup \{v\}$.
13         else
14             $L = lmn(v)$
15             $R = rmn(v)$
16             $i = i + 1$
17             $C_i = \{v\}$
18 return $\phi = (C_1, \ldots, C_q)$
```

Basically, after the last sweep of Lex-BFS, for each vertex v we define the leftmost vertex $lmn(v)$ and the rightmost vertex $rmn(v)$, according to σ_{++}, that are adjacent to v. Checking whether $rmn(v) - lmn(v) = \deg_G(v)$ corresponds exactly to checking whether the neighborhood condition holds for node v. The vertices with the same leftmost and rightmost vertex are then undistinguishable vertices, and they form a block of G. The order of the blocks follows the vertex order σ_{++}. For the correctness of the algorithm we refer the interested reader to [33].

Since the straight enumerations of the level graphs might not be unique, it is important to choose, among all the possible straight enumerations, the ones that lead to a common refinement (if it exists).

In fact, if G is a connected unit interval graph, its straight enumeration ϕ is unique up to reversal (see Theorem 2.3.5). On the other hand, if G is not connected then any possible ordering of the connected components induces a straight enumeration, obtained by concatenating straight enumerations of its connected components. This freedom in choosing the straight enumerations of the components is crucial in order to return *all* the Robinson orderings of A, and it is taken care of in Section 5.5 using PQ-trees. However, for now we are interested in finding *one* common refinement, and once one of the two above cases occurs, the arbitrary choice made does not affect the correctness of the algorithm.

As we will see in Section 5.4, the choice of a straight enumeration of each subgraph G_ω compatible with ψ reduces to correctly orient the ordered partition returned by the subroutine *Straight_enumeration*$(G_\omega, \sigma[V_\omega])$.

5.3.3 Refinements of weak linear orders

Given two weak linear orders ψ and ϕ on V, our second subroutine *Refine* in Algorithm 5.3 computes their common refinement $\Phi = \psi \wedge \phi$ (if it exists). We show below its correctness.

Algorithm 5.3: *Refine*(ψ, ϕ)

```
   input: two weak linear orders $\psi = (B_1, \dots, B_p)$ and $\phi = (C_1, \dots, C_q)$ of $V$
   output: their common refinement $\Phi = \psi \wedge \phi$, or $\Phi = \emptyset$ ($\psi$ and $\phi$ are not
           compatible)
1  $B^{\max}$ is the last block of $\psi$ meeting $C_1$
2  $\Phi = \emptyset$
3  if there exists a block $B$ of $\psi$ such that $B <_\psi B^{\max}$ and $B \not\subseteq C_1$ then
4      return $\emptyset$                              ($\psi$ and $\phi$ are not compatible)
5  else
6      $W = V \setminus C_1$
7      $\Phi = (\psi[C_1], Refine(\psi[W], \phi[W]))$
8  if $\Phi$ is a weak linear order of $V$ then
9      return $\Phi$
10 else
11     return $\emptyset$                              ($\psi$ and $\phi$ are not compatible)
```

5.3.3 Lemma. *If Algorithm 5.3 returns a weak linear order Φ of V, then Φ is the common refinement of ψ and ϕ. If Algorithm 5.3 returns $\Phi = \emptyset$, then ψ and ϕ are not compatible.*

Proof. The proof is by induction on the number q of blocks of ϕ. If $q = 1$ then $\phi = (V)$ is clearly compatible with ψ and the algorithm returns $\Phi = \psi$ as desired. Assume now $q \geq 2$. Let $W = V \setminus C_1$. Then we can apply the induction assumption to $\phi[W]$ which has $q-1$ blocks. Assume first that the algorithm returns Φ which is a weak linear order of V. We show that $\Phi = \psi \wedge \phi$, i.e., that the following holds for all $x, y \in V$:

$$\begin{aligned} &x =_\Phi y \Longleftrightarrow x =_\psi y \text{ and } x =_\phi y, \\ &x <_\Phi y \Longleftrightarrow x \leq_\psi y \text{ and } x \leq_\phi y \text{ with at least one strict inequality.} \end{aligned} \tag{5.2}$$

If $x, y \in C_1$ then $x =_\phi y$ and (5.2) holds since $\Phi[C_1] = \psi[C_1]$. If $x, y \in V \setminus C_1$, then (5.2) holds by the induction assumption. Suppose now $x \in C_1$ and $y \in V \setminus C_1$. Then $x <_\phi y$ and $x <_\Phi y$. We show that $x \leq_\psi y$ holds. For this let B_i (resp., B_j) be the block of ψ containing x (resp., y). Then $B_i \leq_\psi B^{\max}$ since B_1 meets C_1 as $x \in C_1$. Moreover, $B^{\max} \leq_\psi B_j$, which implies $x \leq_\psi y$. Indeed, if one would have $B_j <_\psi B^{\max}$, then we would have $\Phi = \emptyset$ (line 4 in Algorithm 5.3), since $B_j \not\subseteq C_1$ as $y \in B_j \setminus C_1$, and thus Φ would not be a weak linear order of V. Assume now that the returned Φ is not a weak linear order of V. If $\Phi = \emptyset$ (line 4 in Algorithm 5.3), then there is a block $B <_\psi B^{\max}$ such that $B \not\subseteq C_1$, and we can pick elements $x \in B \setminus C_1$ and $y \in C_1 \cap B^{\max}$ so that $y <_\phi x$ and $x <_\psi y$, which shows that ψ and ϕ are not compatible. If Φ is a weak linear order of a subset $U \subset V$ (line 11 in Algorithm 5.3), then it means that the weak linear order returned by the recursive routine *Refine*$(\psi[W], \phi[W])$ is not a weak linear order of W (but of a subset) and thus, by the induction assumption, $\psi[W]$ and $\phi[W]$ are not compatible and thus ψ and ϕ neither. This concludes the proof. □

5.4 The algorithm

We can now describe our main algorithm *Robinson*(A, ψ). Given a nonnegative matrix $A \in \mathcal{S}^n$ and a weak linear order ψ of $V = [n]$, it either returns a weak linear order Φ of V compatible with ψ and with straight enumerations of the level graphs of A, or it indicates that such Φ does not exist. The idea behind the algorithm is the following. We use the subroutines *CO-Lex-BFS* and *Straight_enumeration* to order the components and compute the straight enumerations of the level graphs of A, and we refine them using the subroutine *Refine*. However, instead of refining the level graphs one by one on the full set V, we use a recursive algorithm based on a divide-and-conquer strategy, which refines smaller and smaller subgraphs of the level graphs obtained by restricting to the connected components and thus working independently with the corresponding principal submatrices of A. In this way we work with smaller subproblems and one may also skip some level graphs (as some principal submatrices of A may have fewer distinct nonzero entries). This recursive algorithm is Algorithm 5.4 below.

Algorithm 5.4: *Robinson*(A, ψ)

```
input: a nonnegative matrix $A \in \mathcal{S}^n$ and a weak linear order $\psi$ of $V = [n]$
output: a weak linear order $\Phi$ compatible with $\psi$ and with straight
        enumerations of all the level graphs of $A$, or STOP (such an order
        $\Phi$ does not exist)

1  $G$ is the support of A
2  $CO\text{-}Lex\text{-}BFS(G, \psi)$ returns a linear order $(V_1, \ldots, V_c)$ of the connected
   components of $G$ compatible with $\psi$ (if it exists) and a vertex order $\sigma$
3  $\Phi = \emptyset$
4  for $\omega = 1, \ldots, c$ do
5      $\phi_\omega = Straight\_enumeration(G[V_\omega], \sigma[V_\omega])$  (if $G[V_\omega]$ is a unit int. graph)
6      if $\Phi_\omega = Refine(\psi[V_\omega], \phi_\omega) = \emptyset$ then
7          if $\Phi_\omega = Refine(\psi[V_\omega], \overline{\phi_\omega}) = \emptyset$ then
8              stop    (no straight enumeration compatible with $\psi[V_\omega]$ exists)
9      $a'_{\min}$ is the smallest nonzero entry of $A[V_\omega]$
10     $A'[V_\omega]$ is obtained from $A[V_\omega]$ by setting entries with value $a'_{\min}$ to zero
11     if $A'[V_\omega]$ is diagonal then
12         $\Phi = (\Phi, \Phi_\omega)$
13     else
14         $\Phi = (\Phi, Robinson(A'[V_\omega], \Phi_\omega))$
15 return: $\Phi$
```

The algorithm *Robinson*(A, ψ) works as follows. We are given as input a symmetric nonnegative matrix $A \in \mathcal{S}^n$ and a weak linear order ψ of $V = [n]$. Let G be the support of A. First, we find the connected components of G and we order them in a compatible way with ψ. If this is not possible, then we stop as there do not exist straight enumerations of the level graphs of A compatible with ψ (Theorem 5.3.2). Otherwise, we initialize the weak linear order Φ, which at the end of the algorithm will represent a common refinement of the straight enumerations of the level graphs of A. In order to find Φ, we divide the problem over the connected components of G. The idea is then to work independently on each connected component V_ω and to find its (unique) straight enumeration ϕ_ω and the common refinement Φ_ω of $\psi[V_\omega]$ and ϕ_ω.

For each component V_ω, we compute the straight enumeration ϕ_ω of $G[V_\omega]$ if it exists, else we stop (Theorem 2.3.5). Since ϕ_ω is unique up to reversal, we check if either ϕ_ω or $\overline{\phi_\omega}$ is compatible with $\psi[V_\omega]$. Specifically, we first compute the common refinement Φ_ω of $\psi[V_\omega]$ and ϕ_ω. If it is nonempty we continue (Theorem 5.3.3), while if it is is empty we compute the common refinement Φ_ω of $\psi[V_\omega]$ and $\overline{\phi_\omega}$. If such a common refinement is nonempty we continue (Theorem 5.3.3), while if it is again empty this time we stop, as no straight enumeration of G_ω compatible with $\psi[V_\omega]$ exists. Finally, we set to zero the smallest nonzero entries of $A[V_\omega]$, obtaining the new matrix $A'[V_\omega]$ (whose nonzero entries take fewer distinct values than the matrix $A[V_\omega]$). Now, if the matrix $A'[V_\omega]$ is diagonal, then we concatenate Φ_ω after $\Phi_{\omega-1}$ in Φ. Otherwise, we make a recursive call, where the the input of the recursive routine is the matrix $A'[V_\omega]$ and Φ_ω. If the algorithm successfully terminates, then the concatenation $(\phi_1, \dots, \phi_c)$ will represent a straight enumeration of G, and $\Phi = (\Phi_1, \dots, \Phi_c)$ will represent the common refinement of this straight enumeration with the given weak linear order ψ and with the level graphs of A.

The final algorithm is Algorithm 5.5 below. Roughly speaking, every time we make a recursive call, we are basically passing to the next level graph of A. Hence, each recursive call can be visualized as the node of a recursion tree, whose root is defined by the first recursion in Algorithm 5.5, and whose leaves (i.e. the pruned nodes) are the subproblems whose corresponding submatrices are diagonal.

```
Algorithm 5.5: Robinsonian($A$)
  input: a nonnegative matrix $A \in \mathcal{S}^n$
  output: a permutation $\pi$ such that $A_\pi$ is Robinson or stating that $A$ is
          not Robinsonian

1 $\psi = (V)$
2 if Robinson($A, \psi$) stops then
3     "A is NOT Robinsonian"
4 else
5     $\Phi$=Robinson($A, \psi$)
6     return: a linear order $\pi$ of $V$ compatible with $\Phi$;
```

5.4.1 Correctness

The correctness of Algorithm 5.5 follows directly from the correctness of Algorithm 5.4, which is shown by the next theorem. Indeed, assume that Algorithm 5.4 is correct. Then, if Algorithm 5.5 terminates then it computes a weak linear order Φ compatible with straight enumerations of the level graphs of A and thus the returned order π orders A as a Robinson matrix in view of Theorem 5.2.2 (ii). On the other hand, if Algorithm 5.5 stops then Algorithm 5.4 stops with the input $(A, \psi = (V))$. Then no weak linear order Φ exists which is compatible with straight enumerations of the level graphs of A and thus, in view of Theorem 5.2.2 (i), A is not Robinsonian.

5.4.1 Theorem. *Consider a weak linear order ψ of $V = [n]$ and a nonnegative matrix $A \in \mathcal{S}^n$ ordered compatibly with ψ.*

(i) If Algorithm 5.4 terminates, then there exist straight enumerations $\phi^{(1)}, \ldots, \phi^{(L)}$ of the level graphs $G^{(1)}, \ldots, G^{(L)}$ of A such that the returned weak linear order Φ is compatible with each of them and with ψ.

(ii) If Algorithm 5.4 stops then there do not exist straight enumerations of the level graphs of A that are pairwise compatible and compatible with ψ.

Proof. The proof is by induction on the number L of distinct nonzero entries of the matrix A. We first consider the base case $L = 1$, i.e., when A is (up to scaling) 0/1 valued. We first show (i) and assume that the algorithm terminates succesfully and returns Φ. Then G is the support of A, *CO-Lex-BFS*(G, ψ) orders the components of G as $V_1 \leq_\psi \ldots \leq_\psi V_c$, and $\Phi = (\Phi_1, \ldots, \Phi_c)$ where each $\Phi_\omega = \Phi[V_\omega]$ is build as the common refinement of $\psi[V_\omega]$ and a straight enumeration of $G[V_\omega]$ (either ϕ_ω or $\overline{\phi_\omega}$). Hence G has a straight enumeration ϕ and the returned Φ is compatible with ϕ and ψ.

We now show (ii) and assume that Algorithm 5.4 stops. If it stops when applying *CO-Lex-BFS*(G, ψ), then no order of the components of G exists that is compatible with ψ and thus no straight enumeration of G exists that is compatible with ψ (Lemma 5.3.1). If the algorithm stops when applying *Straight_enumeration* to $G[V_\omega]$ then no straight enumeration of $G[V_\omega]$ exists. Else, if the algorithm stops at line 8 in Algorithm 5.4, then $\psi[V_\omega]$ is not compatible with neither ϕ_ω or $\overline{\phi_\omega}$. Because $G[V_\omega]$ is connected, ϕ_ω and $\overline{\phi_\omega}$ are its unique straight enumerations (see Theorem 2.3.5) and therefore no straight enumeration of $G[V_\omega]$ is compatible with $\psi[V_\omega]$. In both cases, no straight enumeration of G exists that is compatible with ψ.

We now assume that Theorem 5.4.1 holds for any matrix whose entries take at most $L - 1$ distinct nonzero values. We show that the theorem holds when considering A whose nonzero entries take L distinct values. We follow the same lines as the above proof for the case $L = 1$, except that we use recursion for some components. First, assume that the algorithm terminates and returns Φ. Then, $\Phi = (\tilde{\Phi}_1, \ldots, \tilde{\Phi}_c)$ after ordering the components compatibly with ψ as $V_1 \leq_\psi \ldots \leq_\psi V_c$, constructing the common refinement Φ_ω of $\psi[V_\omega]$ and a straight enumeration (say) ϕ_ω of $G[V_\omega]$, and having $\tilde{\Phi}_\omega = \textit{Robinson}(A'[V_\omega], \Phi_\omega)$, where $A'[V_\omega]$ is obtained from $A[V_\omega]$ by setting to 0 its entries with smallest nonzero value. By the induction assumption, $\tilde{\Phi}_\omega$ is compatible with straight enumerations of the level graphs of the matrix $A'[V_\omega]$ and with Φ_ω. As $\tilde{\Phi}_\omega$ is compatible with Φ_ω, which refines both $\psi[V_\omega]$ and ϕ_ω, it follows that $\tilde{\Phi}_\omega$ is compatible with $\psi[V_\omega]$ and ϕ_ω. Therefore, $\tilde{\Phi}_\omega$ is compatible with straight enumerations of all the level graphs of $A[V_\omega]$ and thus $\Phi = (\tilde{\Phi}_1, \ldots, \tilde{\Phi}_c)$ is compatible with ψ and all level graphs of A, as desired.

Assume now that the algorithm stops. If the algorithm stops at *CO-Lex-BFS*(G,ψ), then no linear order of the connected components of G exists that is compatible with ψ and then no straight enumeration of G exists that is compatible with ψ (Lemma 5.3.1), giving the desired conclusion. If the algorithm stops at line 8, then a connected component V_ω is found for which $\psi[V_\omega]$ is not compatible with any straight enumeration of $G[V_\omega]$, giving again the desired conclusion.

Assume now that the algorithm stops at line 14, i.e., there is a component V_ω for which the algorithm terminates when applying *Robinson*$(A'[V_\omega],\Phi_\omega)$. Then, by the induction assumption, we know that:

$$\text{no straight enumerations of the level graphs of } A'[V_\omega] \text{ exist that are pariwise compatible and compatible with } \Phi_\omega, \qquad (*)$$

where Φ_ω is the common refinement of $\psi[V_\omega]$ and a straight enumeration (say) ϕ_ω of $G[V_\omega]$. Assume, for the sake of contradiction, that there exist straight enumerations $\varphi^{(1)},\ldots,\varphi^{(L)}$ of the level graphs $G^{(1)}=G,\ldots,G^{(L)}$ of A, that are pairwise compatible and compatible with ψ. In particular, $\varphi^{(1)}[V_\omega]$ is a straight enumeration of $G[V_\omega]$ compatible with $\psi[V_\omega]$. If $\varphi^{(1)}[V_\omega]=\phi_\omega$, then the restrictions $\varphi^{(\ell)}$ ($\ell\geq 2$) yield straight enumerations of the level graphs of $A'[V_\omega]$ that are pairwise compatible and compatible with ϕ_ω and $\psi[V_\omega]$, and thus with their refinement $\Phi_\omega=\psi[V_\omega]\wedge\phi_\omega$, contradicting (*). Hence, $\varphi^{(1)}[V_\omega]=\overline{\phi}_\omega$, so that $\psi[V_\omega]$ is compatible with both ϕ_ω and its reversal $\overline{\phi}_\omega$. This implies that $\psi[V_\omega]=(V_\omega)$. But then the reversals $\overline{\varphi}^{(2)}[V_\omega],\ldots,\overline{\varphi}^{(L)}[V_\omega]$ provide straight enumerations of the level graphs of $A'[V_\omega]$ that are pairwise compatible and compatible with $\overline{\varphi}^{(1)}[V_\omega]=\phi_\omega=\Phi_\omega$. This contradicts again (*) and concludes the proof. $\square$

5.4.2 Complexity analysis

We now study the complexity of our main algorithm. First we discuss the complexity of the two subroutines *CO-Lex-BFS* and *Refine* in Algorithms 5.1 and 5.3 and then we derive the complexity of the final Algorithm 5.5. In the rest of the section, we let m denote the number of nonzero (upper diagonal) entries of A, so that m is the number of edges of the support graph $G=G^{(1)}$ and $m=|E_1|\geq|E_2|\geq\ldots\geq|E_L|$ for the level graphs of A. We assume that A is a nonnegative symetric matrix, which is given as an adjacency list of an undirected weighted graph, where each vertex $x\in V$ is linked to the list of vertex/weight pairs corresponding to the neighbors y of x in G with nonzero entry A_{xy}.

A simple but important observation that we will repeatedly use is that, if we consider a linear order τ compatible with ψ, then the blocks of ψ are intervals of the order τ and thus one can check whether a given set $C\subseteq V$ is contained in a block B of ψ in $O(|C|)$ operations (simply by comparing each element of C to the end points of the interval B). Furthermore, the size of any block of ψ is simply given by the difference between its extremities (plus one).

5.4.2 Lemma. *Algorithm 5.1 runs in $O(|V|+|E|)$ time.*

Proof. As shown in Subsection 4.2.3, Lex-BFS can be implemented in linear time $O(|V|+|E|)$ (see Theorem 4.2.2). Hence, in our implementation of Algorithm 5.1 we will follow the same linear time implementation (see Algorithm 4.3). Recall that the blocks of ψ are intervals in τ, which is a linear order compatible with ψ. We maintain a doubly linked list, where each node of the list represents a connected component V_ω of G and it has a pointer to the connected component $V_{\omega-1}$ ordered immediately before V_ω and to the connected component $V_{\omega+1}$ ordered immediately after V_ω. Then, swapping two connected components can be done simply by swapping the left and right pointers of the corresponding connected components in the doubly linked list. Furthermore, each node in this list contains the set of vertices in V_ω, the first block $B_\omega^{\min}$ and the last block $B_\omega^{\max}$ in ψ meeting V_ω. These two blocks $B_\omega^{\min}$ and $B_\omega^{\max}$ can be found in $O(|V_\omega|)$ as follows. First one finds the smallest element $v_{\min}$ (resp. the largest element $v_{\max}$) of V_ω in the order τ, which can be done in $O(|V_\omega|)$. Then, $B_\omega^{\min}$ is the block of ψ containing $v_{\min}$, which can be found in $O(|V_\omega|)$. Analogously for $B_\omega^{\max}$, which is the block of ψ containing $v_{\max}$. Checking whether V_ω is contained in the block $B_{\omega-1}^{\min}$ can be done in $O(|V_\omega|)$ (since $B_{\omega-1}^{\min}$ is an interval). In order to check whether all the inner blocks between $B_\omega^{\min}$ and $B_\omega^{\max}$ are contained in V_ω we proceed as follows. Let B_ω be the union of these inner blocks, which is an interval of τ. First we compute the sets $V_\omega \cap B_\omega^{\min}$ and $V_\omega \cap B_\omega^{\max}$, which can be done in $O(|V_\omega|)$. Then we need to check whether $B_\omega \subseteq V_\omega$ or, equivalently, whether the two sets $V_\omega \setminus (B_\omega^{\min} \cup B_\omega^{\max})$ and B_ω are equal. For this we check first whether $V_\omega \setminus (B_\omega^{\min} \cup B_\omega^{\max})$ is contained in B_ω (in time $O(|V_\omega|)$) and then whether these two sets have the same cardinality, which can be done in $O(|V_\omega|)$. Hence, the complexity of this task is $O(\sum_\omega |V_\omega|) = O(|V|)$. Therefore we can conclude that the overall complexity of Algorithm 5.1 is $O(|V|+|E|)$. □

5.4.3 Lemma. *Algorithm 5.3 runs in $O(|V|)$ time.*

Proof. We show the lemma using induction on the number q of blocks of ϕ. Recall that the blocks of ψ are intervals in τ, which is a linear order compatible with ψ. If $q = 1$ the result is clear since the algorithm returns $\Phi = \phi$ without any work. Assume $q \geq 2$. The first task is to compute the last block $B^{\max}$ of ψ meeting C_1. For this, as in the proof of the previous lemma, one finds the largest element $v_{\max}$ of C_1 in the order τ and one returns the block of ψ containing $v_{\max}$, which can be done in $O(|C_1|)$. Then let B be the union of the blocks preceding $B^{\max}$. In order to check whether $B \subseteq C_1$ or, equivalently, whether $C_1 \setminus B^{\max} = B$, we proceed as in the previous lemma: we first check whether $C_1 \setminus B^{\max} \subseteq B$ and then whether $|C_1 \setminus B^{\max}| = |B|$, which can be done in $O(|C_1|)$. Hence, the running time is $O(|C_1|)$ for this task which, together with the running time $O(|V \setminus C_1|)$ for the recursive application of *Refine* to the restrictions of ψ, ϕ to the set $V \setminus C_1$, gives an overall running time $O(|V|)$. □

We can now complete the complexity analysis of our algorithm.

5.4.4 Theorem. *Algorithm 5.5 applied to a nonnegative $n \times n$ symmetric matrix A with m nonzero entries recognizes whether A is a Robinsonian matrix in $O(d(|V| + |E|))$ time, where d is the depth of the recursion tree created by Algorithm 5.5. Moreover, $d \leq L$, where L is the number of distinct nonzero entries of A.*

Proof. We show the result using induction on the depth d of the recursion tree. In Algorithm 5.5 we are given a matrix A and its support graph G, and we set $\psi = (V)$. First we run the routine *CO-Lex-BFS*(G, ψ) in $O(|V| + |E|)$ time, in order to find and order the components of G. For each component V_ω, the following tasks are performed. We compute a straight enumeration ϕ_ω of $G[V_\omega]$, in time $O(|V_\omega| + m_\omega)$ where m_ω is the number of edges of $G[V_\omega]$. The reversal $\overline{\phi}_\omega$ can be computed in $O(|V_\omega|)$ by simply reversing the ordered partition ϕ_ω, which is stored in a double linked list. Hence, we apply the routine *Refine* to $\psi[V_\omega]$ and ϕ_ω (or $\overline{\phi}_\omega$), which can be done in $O(|V_\omega|)$ time. Then we build the new matrix $A'[V_\omega]$ and checks whether it is diagonal, in time $O(m_\omega)$. Finally, by the induction assumption, the recursion step *Robinson*$(A'[V_\omega], \Phi_\omega)$ is carried out in time $O(d_\omega(|V_\omega| + m_\omega))$, where d_ω denotes the depth of the corresponding recursion tree. As $d_\omega \leq d - 1$ for each ω, after summing up, we find that the overall complexity is $O(d(|V| + |E|))$.

The last claim: $d \leq L$ is clear since the number of distinct nonzero entries of the current matrix decreases by at least 1 at each recursion node. □

As we will see in the example in the Section 5.6, the depth d of the recursion tree can be smaller than the number L of distinct nonzero entries. Indeed in this example we have $d = 7$ while $L = 10$. Nevertheless, the experiments in Chapter 9 show that in many instances the depth of the recursion is exactly equal to L.

5.5 Finding all Robinson orderings

In general, there might exist several permutations reordering a given matrix A as a Robinson matrix. We show here how to return all Robinson orderings of a given matrix A, using the PQ-tree data structure of [14] (see Subsection 2.2.4).

The main observation is that a straight enumeration $\psi = (B_1, \ldots, B_p)$ of a graph $G = (V, E)$ corresponds in a unique way to a PQ-tree $\mathcal{T}$ as follows. If G is connected, then the root of $\mathcal{T}$ is a Q-node, denoted γ, and it has children $\beta_1, \ldots, \beta_p$ (in that order). For $i \in [p]$, the node β_i is a P-node corresponding to the block B_i and its children are the elements of the set B_i, which are the leaves of the subtree $\mathcal{T}_{\beta_j}$. If a block B_i is a singleton then no node β_i appears and the element of B_i is directly a child of the root γ (see the example in Figure 5.1).

$$A_G = \begin{array}{c|cccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \hline \mathbf{1} & 1 & 1 & 1 & 1 & 0 & 0 \\ \mathbf{2} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{3} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{4} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{5} & 0 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{6} & 0 & 1 & 1 & 1 & 1 & 1 \end{array}$$

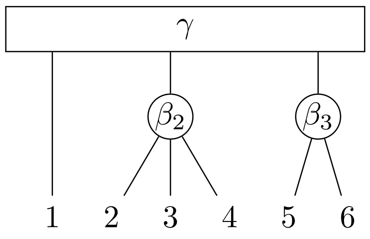


Figure 5.1: A connected graph G and the PQ-tree corresponding to its straight enumeration

If G is not connected, let $V_1, \dots, V_c$ be its connected components. For each connected component $G[V_\omega]$, $\mathcal{T}_\omega$ is its PQ-tree (with root γ_ω) as indicated above. Then, the full PQ-tree $\mathcal{T}$ is obtained by inserting a new P-node α as ancestor, whose children are the subtrees $\mathcal{T}_1, \dots, \mathcal{T}_c$ (see Figure 5.2).

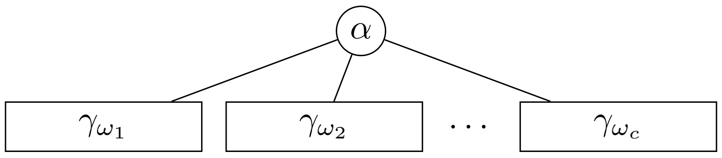


Figure 5.2: The PQ-tree corresponding to the straight enumeration of a disconnected graph

We now indicate how to modify Algorithms 5.4 and 5.5 in order to return a PQ-tree $\mathcal{T}$ encoding all the permutations ordering A as a Robinson matrix.

We modify Algorithm 5.4 by taking as input, beside the matrix A and the weak linear order ψ, also a node α (see Algorithm 5.6). Then, the output is a PQ-tree $\mathcal{T}_\alpha$ rooted in α, representing all the possible weak linear orders compatible with ψ and with straight enumerations of all the level graphs of A. It works as follows.

Let G be the support of A. The idea is to recursively build a tree $\mathcal{T}_\omega$ for each connected component V_ω of G and then to merge these trees according to the order of the components found by the routine *CO-Lex-BFS*(G, ψ). To carry out this merging step we classify the components into the following three groups:

1. Θ, which consists of all $\omega \in [c]$ for which the connected component V_ω meets at least two blocks of ψ.
2. Λ, which consists of all $\omega \in [c]$ for which the component V_ω is contained in some block B_i, which contains no other component.
3. $\Omega = \cup_{i=1}^{p} \Omega_i$, where Ω_i consists of all $\omega \in [c]$ for which the component V_ω is contained in the block B_i, which contains at least two components.

Every time we analyze a new connected component $\omega \in [c]$ in Algorithm 5.4, we create a Q-node γ_ω. After the common refinement Φ_ω (of $\psi[V_\omega]$ and the straight enumeration ϕ_ω of $G[V_\omega]$ or its reversal) has been computed, we have two possibilities. If $A'[V_\omega]$ is diagonal, then we build the tree $\mathcal{T}_\omega$ rooted in γ_ω and whose children are P-nodes corresponding to the blocks of Φ_ω (and prune the recursion tree at this node). Otherwise, we build the tree $\mathcal{T}_\omega$ recursively as output of *Robinson*$(A'[V_\omega], \Phi_\omega, \gamma_\omega)$.
After all the connected components have been analyzed, we insert the trees $\mathcal{T}_\omega$ in the final tree $\mathcal{T}_\alpha$ in the order they appear according to the routine *CO-Lex-BFS*(G, ψ). The root node is α and is given as input. For each component V_ω, we do the following operation to insert $\mathcal{T}_\omega$ in $\mathcal{T}_\alpha$, depending on the type of the component V_ω:

1. If $\omega \in \Theta$, then ϕ_ω (or $\overline{\phi}_\omega$) is the only straight enumeration compatible with $\psi[V_\omega]$. Then we delete the node γ_ω and the children of γ_ω become children of α (in the same order).

2. If $\omega \in \Lambda$, then both ϕ_ω and its reversal $\overline{\phi}_\omega$ are compatible with $\psi[V_\omega]$. Then γ_ω becomes a child of α.

3. If $\omega \in \Omega_i$ for some $i \in [p]$, then both ϕ_ω and $\overline{\phi}_\omega$ are compatible with $\psi[V_\omega]$ and the same holds for any $\omega' \in \Omega_i$. Moreover, arbitrary permuting any two connected components $V_\omega, V_{\omega'}$ with $\omega, \omega' \in \Omega_i$ will lead to a compatible straight enumeration. Then we insert a new node β_i which is a P-node and becomes a child of α and, for each $\omega' \in \Omega_i$, $\gamma_{\omega'}$ becomes a child of β_i.

Finally, we modify Algorithm 5.5 by just giving a new node $\alpha = \emptyset$ (i.e. undefined) as input to the first recursive call (see Algorithm 5.7). The overall complexity of the algorithm after the above mentioned modifications is the same as for Algorithm 5.5. Indeed, determining the type of the connected components can be done in linear time, by just using the information about the initial and final blocks $B_\omega^{\min}$ and $B_\omega^{\max}$ already provided in Algorithm 5.1. Furthermore, the operations on the PQ-tree are basic operations that do not increase the overall complexity of the algorithm.

Algorithm 5.6: *Robinson*(A, ψ, α)

```
input: a nonnegative matrix $A \in \mathcal{S}^n$, a weak linear order $\psi$ of $V = [n]$ and
       a node $\alpha$
output: A PQ-tree $\mathcal{T}_\alpha$ representing all the possible weak linear order $\Phi$
        compatible with $\psi$ and with straight enumerations of all the level
        graphs of $A$, or STOP (such a tree does not exist)

1  $G$ is the support of A
2  $CO\text{-}Lex\text{-}BFS(G, \psi)$ returns a linear order $(V_1, \ldots, V_c)$ of the connected
   components of $G$ compatible with $\psi$ (if it exists) and a vertex order $\sigma$
3  group the connected components (c.c.) $V_\omega$ ($\omega \in [c]$) of $G$ as follows:
4  $\Theta$ : all $\omega$ for which $V_\omega$ meets at least two blocks of $\psi$
5  $\Lambda$ : all $\omega$ for which $V_\omega$ is contained in a block $B_i$ containing no other c.c.
6  for $i \in [p]$, $\Omega_i$: all $\omega$ for which $V_\omega \subseteq B_i$ and $B_i$ contains at least two c.c.
7  $\Phi = \emptyset$
8  for $\omega = 1, \ldots, c$ do
9      create a Q-node $\gamma_\omega$
10     $\phi_\omega = Straight\_enumeration(G[V_\omega], \sigma[V_\omega])$  (if $G[V_\omega]$ is a unit int. graph)
11     if $\Phi_\omega = Refine(\psi[V_\omega], \phi_\omega) = \emptyset$ then
12         if $\Phi_\omega = Refine(\psi[V_\omega], \overline{\phi_\omega}) = \emptyset$ then
13             stop      (no straight enumeration compatible with $\psi[V_\omega]$ exists)
14     $a'_{\min}$ is the smallest nonzero entry of $A[V_\omega]$
15     $A'[V_\omega]$ is obtained from $A[V_\omega]$ by setting entries with value $a'_{\min}$ to zero
16     if $A'[V_\omega]$ is diagonal then
17         create a PQ-tree $\mathcal{T}_\omega$ rooted in $\gamma_\omega$ and whose children are P-nodes
           corresponding to the blocks of $\Phi_\omega$
18     else
19         $\mathcal{T}_\omega = Robinson(A'[V_\omega], \Phi_\omega, \gamma_\omega)$
20 $\mathcal{T}_\alpha$ is the PQ-tree rooted in $\alpha$, build as follows:
21 $\omega = 1$
22 while $\omega \leq c$ do
23     if $\omega \in \Theta$ then
24         the children of $\gamma_\omega$ become children of $\alpha$ and remove $\gamma_\omega$; $\omega = \omega + 1$
25     else
26         if $\omega \in \Lambda$ then
27             set $\mathcal{T}_\omega$ as child of $\alpha$ (if $\alpha = \emptyset$, then set $\alpha = \gamma_\omega$); $\omega = \omega + 1$
28         else
29             let $\Omega_i$ s.t. $\omega \in \Omega_i$; create a P-node $\beta_i$ and set it as child of $\alpha$ (if
               $\alpha = \emptyset$, then set $\alpha = \beta_i$)
30             foreach $\omega' \in \Omega_i$ do
31                 set $\gamma_{\omega'}$ as children of $\beta_i$
32             $\omega = \omega + |\Omega_j|$
33 return: $\mathcal{T}_\alpha$ or STOP
```

Algorithm 5.7: *Robinsonian*(A)

```
input: a nonnegative matrix $A \in \mathcal{S}^n$
output: a PQ-tree $\mathcal{T}$ that encodes all the permutations $\pi$ such that $A_\pi$ is
        a Robinson matrix or stating that $A$ is not Robinsonian

1 $\psi = (V)$
2 $\alpha = \emptyset$
3 $G$ is the support of A
4 $\mathcal{T}$=Robinson$(A, \psi, \alpha)$
5 if the leaves of $\mathcal{T}$ are $n$ then
6     return: $\mathcal{T}$
7 else
8     "A is NOT Robinsonian"
```

5.6 Example

We show on a concrete example how the algorithm works. We consider the same matrix A as the one used in the example in Section 5 of [94]. However, since [94] handles Robinsonian dissimilarities, we first transform it into a similarity matrix and thus we use instead the matrix $a_L J - A$, where $a_L = 11$ denotes the largest entry in the matrix A. If we rename such a new matrix as A, it looks as follows:

$$A = \begin{array}{r|ccccccccccccccccccc|}
 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \mathbf{12} & \mathbf{13} & \mathbf{14} & \mathbf{15} & \mathbf{16} & \mathbf{17} & \mathbf{18} & \mathbf{19} \\
\mathbf{1} & 11 & 2 & 9 & 0 & 5 & 0 & 5 & 5 & 2 & 0 & 5 & 0 & 5 & 6 & 0 & 0 & 2 & 0 & 5 \\
\mathbf{2} & & 11 & 2 & 0 & 9 & 0 & 8 & 5 & 10 & 0 & 5 & 0 & 5 & 2 & 0 & 0 & 10 & 0 & 8 \\
\mathbf{3} & & & 11 & 0 & 5 & 0 & 5 & 5 & 2 & 0 & 5 & 0 & 5 & 10 & 0 & 0 & 2 & 0 & 5 \\
\mathbf{4} & & & & 11 & 0 & 3 & 0 & 0 & 0 & 3 & 0 & 3 & 0 & 0 & 10 & 3 & 0 & 9 & 0 \\
\mathbf{5} & & & & & 11 & 0 & 8 & 7 & 9 & 0 & 7 & 0 & 7 & 5 & 0 & 0 & 9 & 0 & 10 \\
\mathbf{6} & & & & & & 11 & 0 & 0 & 0 & 10 & 0 & 6 & 0 & 0 & 5 & 8 & 0 & 5 & 0 \\
\mathbf{7} & & & & & & & 11 & 7 & 8 & 0 & 7 & 0 & 7 & 5 & 0 & 0 & 8 & 0 & 9 \\
\mathbf{8} & & & & & & & & 11 & 6 & 0 & 10 & 0 & 8 & 7 & 0 & 0 & 6 & 0 & 7 \\
\mathbf{9} & & & & & & & & & 11 & 0 & 6 & 0 & 5 & 2 & 0 & 0 & 10 & 0 & 8 \\
\mathbf{10} & & & & & & & & & & 11 & 0 & 6 & 0 & 0 & 4 & 9 & 0 & 5 & 0 \\
\mathbf{11} & & & & & & & & & & & 11 & 0 & 9 & 7 & 0 & 0 & 6 & 0 & 7 \\
\mathbf{12} & & & & & & & & & & & & 11 & 0 & 0 & 9 & 6 & 0 & 10 & 0 \\
\mathbf{13} & & & & & & & & & & & & & 11 & 7 & 0 & 0 & 5 & 0 & 7 \\
\mathbf{14} & & & & & & & & & & & & & & 11 & 0 & 0 & 2 & 0 & 5 \\
\mathbf{15} & & & & & & & & & & & & & & & 11 & 4 & 0 & 10 & 0 \\
\mathbf{16} & & & & & & & & & & & & & & & & 11 & 0 & 4 & 0 \\
\mathbf{17} & & & & & & & & & & & & & & & & & 11 & 0 & 8 \\
\mathbf{18} & & & & & & & & & & & & & & & & & & 11 & 0 \\
\mathbf{19} & & & & & & & & & & & & & & & & & & & 11
\end{array} \tag{5.3}$$

Here the bold labels denote the original numbering of the elements. The recursion tree computed by Algorithm 5.5 is shown in Figure 5.3 at page 87. The weak linear order at each node represents the weak linear order ψ given as input to the recursion node, while the number on the edge between two nodes denotes the minimum value in the current matrix A, which is set to zero before making a new recursion call (in this way, the reader may reconstruct the input given at each recursion node).

Root node
We set $\psi = (V)$ and invoke Algorithm 5.4. Then, Algorithm 5.1 would find two connected components:

$$V_1 = \{1, 2, 3, 5, 7, 8, 9, 11, 13, 14, 17, 19\},$$
$$V_2 = \{4, 6, 10, 12, 15, 16, 18\}.$$

Hence, we can split the problem into two subproblems, where we deal with each connected component independently.

1.0 Connected component V_1, level 0
The submatrix $A[V_1]$ is a clique, and thus we have $\psi[V_1] = \phi = (V_1)$. The smallest nonzero value of the submatrix is $a'_{\text{min}} = 2$. Hence, we compute $A'[V_1]$ by setting to zero the entries of the submatrix with value equal to a'_{min}, and we make a recursive call, setting $\psi = \Phi$. To simplify notation, we shall rename $A'[V_1]$ as $A[V_1]$ after every iteration.

1.1 Connected component V_1, level 1
The matrix in input at the current recursion node is shown below.

$$A[V_1] = \begin{array}{c|cccccccccccc}
 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{5} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{11} & \mathbf{13} & \mathbf{14} & \mathbf{17} & \mathbf{19} \\
\hline
\mathbf{1} & 11 & 0 & 9 & 5 & 5 & 5 & 0 & 5 & 5 & 6 & 0 & 5 \\
\mathbf{2} & & 11 & 0 & 9 & 8 & 5 & 10 & 5 & 5 & 0 & 10 & 8 \\
\mathbf{3} & & & 11 & 5 & 5 & 5 & 0 & 5 & 5 & 10 & 0 & 5 \\
\mathbf{5} & & & & 11 & 8 & 7 & 9 & 7 & 7 & 5 & 9 & 10 \\
\mathbf{7} & & & & & 11 & 7 & 8 & 7 & 7 & 5 & 8 & 9 \\
\mathbf{8} & & & & & & 11 & 6 & 10 & 8 & 7 & 6 & 7 \\
\mathbf{9} & & & & & & & 11 & 6 & 5 & 0 & 10 & 9 \\
\mathbf{11} & & & & & & & & 11 & 9 & 7 & 6 & 7 \\
\mathbf{13} & & & & & & & & & 11 & 7 & 5 & 7 \\
\mathbf{14} & & & & & & & & & & 11 & 0 & 5 \\
\mathbf{17} & & & & & & & & & & & 11 & 8 \\
\mathbf{19} & & & & & & & & & & & & 11
\end{array} \qquad (5.4)$$

If we invoke Algorithm 5.2 we get the following straight enumeration:

$$\phi = (\{1, 3, 14\}, \{5, 7, 8, 11, 13, 19\}, \{2, 9, 17\}).$$

Note that since $\psi[V_1]$ has only one block, we do not need to compute the partition refinement in Algorithm 5.3, and the common refinement is simply $\Phi = \phi$. The smallest nonzero value of the matrix in (5.4) is $a'_{\text{min}} = 5$. Hence, we compute $A'[V_1]$ by setting to zero the entries of the matrix in (5.4) with value equal to a'_{min}. Then, we make a recursion call, and we set $\psi = \Phi$.

1.2 Connected component V_1, level 2
The input matrix $A[V_1]$ is obtained by setting to zero the entries of the matrix in (5.4) with value at most 5. The support of this matrix is still connected, and its straight enumeration is:

$$\phi = (\{1,3\},\{14\},\{13\},\{8,11\},\{5,7,19\},\{9,17\},\{2\}).$$

If we invoke Algorithm 5.3, it is easy to see that the common refinement Φ of ψ and ϕ is exactly ϕ. The smallest nonzero value of $A[V_1]$ is now $a'_{\min} = 6$. Hence, we compute $A'[V_1]$ and we make a recursion call, setting $\psi = \Phi$ and renaming $A'[V_1]$ as $A[V_1]$.

1.3 Connected component V_1, level 3
The input matrix $A[V_1]$ is obtained by setting to zero the entries of the matrix in (5.4) with value at most 6. The support of this matrix is still connected, and its straight enumeration is:

$$\phi = (\{1\},\{3\},\{14\},\{13,8,11\},\{5,7,19\},\{9,17,2\}).$$

The common refinement with ψ is then given by:

$$\Phi = (\{1\},\{3\},\{14\},\{13\},\{8,11\},\{5,7,19\},\{9,17\},\{2\}).$$

The smallest nonzero value of $A[V_1]$ is now $a'_{\min} = 7$. Hence, we compute $A'[V_1]$ and we make a recursion call, setting $\psi = \Phi$.

1.4 Connected component V_1, level 4
The input matrix $A[V_1]$ is obtained by setting to zero the entries of the matrix in (5.4) with value at most 7. The support of this matrix is not connected, and thus Algorithm 5.1 will detect the following connected components:

$$\begin{aligned} V_{11} &= \{1,3,14\}, \\ V_{12} &= \{13,8,11\}, \\ V_{13} &= \{5,7,19,9,17,2\}. \end{aligned}$$

1.4.1 Connected component V_{11}, level 4
We have that $\psi[V_{11}] = (\{1\},\{3\},\{14\})$. Because all its block are singletons, it is a linear order and thus it cannot be refined more. Since Φ is a Robinson ordering for $A[V_{11}]$, we do not apply recursion and pass to the next connected component.

1.4.2 Connected component V_{12}, level 4
We have that $\psi[V_{12}] = (\{13\},\{8,11\})$. The submatrix $A[V_12]$ is a clique, and thus $\phi = (V_{12})$. Hence, the common refinement is $\Phi = (\{13\},\{8,11\})$. The smallest nonzero value of $A[V_12]$ is now $a'_{\min} = 8$. Hence, we compute $A'[V_{12}]$ and we make a recursion call, setting $\psi = \Phi$.

1.5.2 Connected component V_{12}, level 5
The input matrix $A[V_{12}]$ is obtained by setting to zero the entries of the matrix in (5.4) restricted to V_{12} with value at most 8. The straight enumeration is $\phi = (\{13\},\{11\},\{8\})$, and thus the common refinement is $\Phi = (\{13\},\{11\},\{8\})$. Because all its block are singletons, Φ is a linear order and thus it cannot be refined more. Furthermore, Φ is Robinson for $A[V_{12}]$, and thus we can prune the recursion node.

1.4.3 Connected component V_{13}, level 4
We have that $\psi[V_{13}] = (\{5,7,19\},\{9,17\},\{2\})$. The submatrix $A[V_{13}]$ is a clique, and thus $\phi = (V_{13})$. Hence, the common refinement is $\Phi = (\{5,7,19\},\{9,17\},\{2\})$. The smallest nonzero value of $A[V_{13}]$ is $a'_{\min} = 8$. Hence, we compute $A'[V_{13}]$ and we make a recursion call, setting $\psi = \Phi$.

1.5.3 Connected component V_{13}, level 5
The input matrix $A[V_{13}]$ is obtained by setting to zero the entries of the matrix in (5.4) restricted to V_{13} with value at most 8. If we invoke Algorithm 5.2, we get the straight enumeration $\phi = (\{2\},\{17,9\},\{5\},\{19\},\{7\})$, which is not compatible with ψ. Hence we compute $\overline{\phi} = (\{7\},\{19\},\{5\},\{9,17\},\{2\})$, which is compatible with ψ, and thus we have that $\Phi = \overline{\phi}$. The smallest nonzero value of $A[V_{13}]$ is $a'_{\min} = 9$. We update $A'[V_1]$, and we make a recursive call.

1.6.3 Connected component V_{13}, level 6
The new input matrix is then given by the submatrix in (5.4) restricted to V_{13} by setting to zero the entries with value at most 9. The support of this matrix is not connected, and thus Algorithm 5.1 will detect the following connected components:

$$\begin{aligned} V_{131} &= \{7\}, \\ V_{132} &= \{19,5\}, \\ V_{133} &= \{9,17,2\}. \end{aligned}$$

We then split the problem over the connected components. The first one has only one vertex, and therefore we can update Φ and pass to the next connected components. The second and the third ones are matrices with all off-diagonal entries equal, and thus we can stop. This was the last recursion node open of the first subtree.

Therefore, we get that the final common refinement of the level graphs of the matrix in (5.4) is:

$$\Phi_1 = (\{1\},\{3\},\{14\},\{13\},\{11\},\{8\},\{7\},\{19\},\{5\},\{9,17\},\{2\})$$

and the PQ-tree $\mathcal{T}_1$ computed by the algorithm is shown in Figure 5.4 at page 88.

2.0 Connected component V_2, level 0
The submatrix $A[V_2]$ is a clique, and thus we have $\psi[V_2] = \phi = (V_2)$. The smallest

nonzero value of the submatrix is $a'_{\min} = 3$. Hence, we compute $A'[V_2]$ by setting to zero the entries of the submatrix with value equal to $a'_{\min}$, and we make a recursive call, setting $\psi = \Phi$. Again, to simplify notation, we shall rename $A'[V_2]$ as $A[V_2]$ after every iteration.

2.1 Connected component V_2, level 1
The matrix in input at the current recursion node is shown below.

$$A[V_2] = \begin{array}{c} \\ \mathbf{4} \\ \mathbf{6} \\ \mathbf{10} \\ \mathbf{12} \\ \mathbf{15} \\ \mathbf{16} \\ \mathbf{18} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathbf{4} & \mathbf{6} & \mathbf{10} & \mathbf{12} & \mathbf{15} & \mathbf{16} & \mathbf{18} \end{array} \\ \begin{pmatrix} 11 & 0 & 0 & 0 & 10 & 0 & 9 \\ & 11 & 10 & 6 & 5 & 8 & 5 \\ & & 11 & 6 & 4 & 9 & 5 \\ & & & 11 & 9 & 6 & 10 \\ & & & & 11 & 4 & 10 \\ & & & & & 11 & 4 \\ & & & & & & 11 \end{pmatrix} \end{array} \tag{5.5}$$

If we invoke Algorithm 5.2, we get the following straight enumeration:

$$\phi = (\{4\}, \{15, 18\}, \{6, 10, 12, 16\}).$$

Note that since $\psi[V_2]$ has only one block, we do not have to compute the partition refinement in Algorithm 5.3, and then the common refinement is simply $\Phi = \phi$. The smallest nonzero value of the matrix in (5.5) is $a'_{\min} = 4$. Hence, we compute $A'[V_2]$ by setting to zero the entries of the matrix in (5.5) with value equal to $a'_{\min}$, we set $\psi = \Phi$ and we make a recursion call.

2.2 Connected component V_2, level 2
The input matrix $A[V_2]$ is obtained by setting to zero the entries of the matrix in (5.5) with value at most 4. The support of this matrix is still connected, and its straight enumeration is:

$$\phi = (\{4\}, \{15\}, \{18\}, \{6, 12\}, \{10\}, \{16\}).$$

If we invoke Algorithm 5.3, it is easy to see that the common refinement is $\Phi = \phi$ The smallest nonzero value of $A[V_2]$ is now $a'_{\min} = 5$. Hence, we compute $A'[V_2]$, we set $\psi = \Phi$ and we make a recursion call.

2.3 Connected component V_2, level 3
The input matrix $A[V_2]$ is obtained by setting to zero the entries of the matrix in (5.5) with value at most 5. The support of this matrix is still connected, and its straight enumeration is:

$$\phi = (\{4, 15\}, \{18\}, \{12\}, \{6, 10, 16\}).$$

The common refinement is then simply:

$$\Phi = (\{4\}, \{15\}, \{18\}, \{12\}, \{6\}, \{10\}, \{16\}).$$

Because the common refinement consists of all singletons, it cannot be refined anymore. Furthermore, Φ is a Robinson ordering of $A[V_2]$. Hence, we can stop and prune the recursion tree.

The final common refinement of level graphs of $A[V_2]$ is:

$$\Phi_2 = (\{4\}, \{15\}, \{18\}, \{12\}, \{6\}, \{10\}, \{16\})$$

and the PQ-tree $\mathcal{T}_2$ computed by the algorithm is shown in Figure 5.5 at page 88.

Finally, we can build the PQ-tree representing the permutation reordering A as a Robinson matrix. Since both V_1 and V_2 are contained in the same block of ψ (which at the beginning is $\psi = ([n])$), then we create a P-node (named α since it is the ancestor) whose children are the subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$. The final PQ-tree is shown in Figure 5.6 at page 88, and is equivalent to the one returned by [94].

5.7 Conclusions and future work

In this chapter we introduced a new combinatorial algorithm to recognize Robinsonian matrices, based on a new characterization of Robinsonian matrices in terms of straight enumerations of unit interval graphs. The algorithm is simple, rather intuitive and relies only on basic routines like Lex-BFS and partition refinement, and it is well suited for sparse matrices.

The complexity depends on the depth d of the recursion tree. As we will comment in Chapter 9, in the computational experiments we found matrices for which the depth is tight and strictly bigger than n, i.e., $d = L > n$. Nevertheless, for practical purpose, a possible way to bound the depth is to find criteria to prune recursion nodes. One possibility would be, when a submatrix is found for which the current weak linear order consists only of singletons, to check whether the corresponding permuted matrix is Robinson. Analyzing the complexity implications will be the subject of future work.

Another possible way to improve the complexity might be to compute the straight enumeration of the first level graph and then update it dynamically (in constant time, using an appropriate data structure) without having to compute every time the whole straight enumeration of the next level graphs; this would need to extend the dynamic approach of [67], which considers the case of single edge deletions, to the deletion of sets of edges.

Other possible future work includes investigating how the algorithm could be used to design heuristics or approximation algorithm when A is not Robinsonian, for example by using (linear) certifying algorithms as in [66] to detect the edges and the nodes of the level graphs which create obstructions to being a unit interval graph.

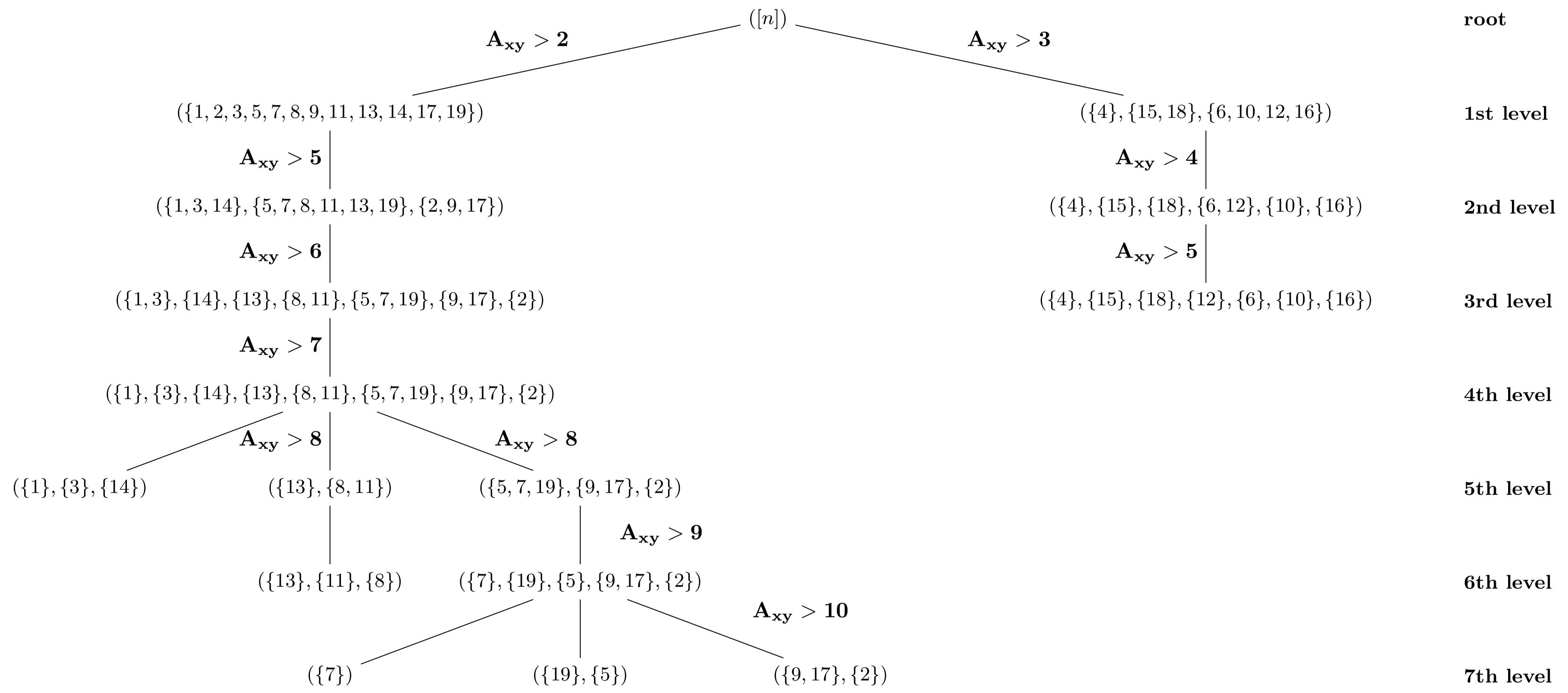


Figure 5.3: Recursion tree. The weak linear order at each node represent the weak linear order ψ given as input to the recursion node, while the number on the edge between two nodes denotes the minimum value set to zero in the original matrix (5.3) before making a new recursion call.

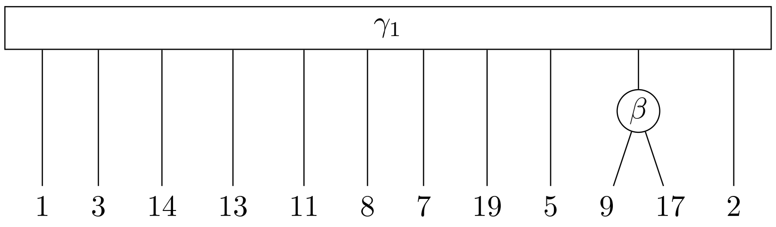


Figure 5.4: The PQ-tree corresponding to the common refinement of level graphs of $A[V_1]$.

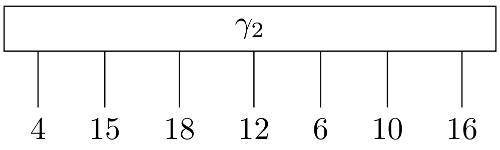


Figure 5.5: The PQ-tree corresponding to the common refinement of level graphs of $A[V_2]$.

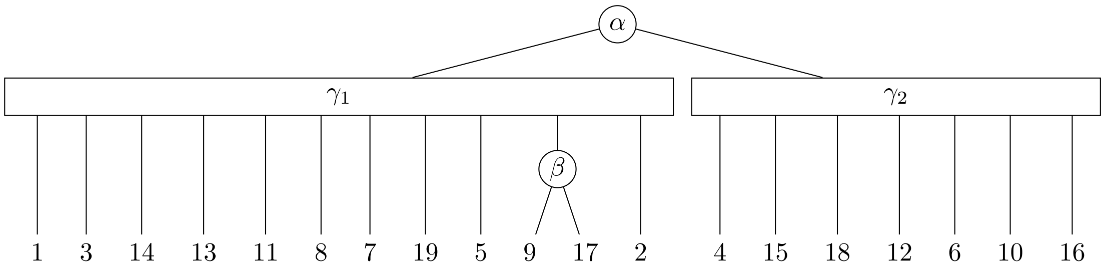


Figure 5.6: The PQ-tree corresponding to the permutations reordering A as a Robinson matrix.

6

Similarity-First Search

In this chapter we present a new combinatorial recognition algorithm for Robinsonian matrices. As main ingredient, we introduce a new algorithm, called *Similarity-First Search* (SFS), which is an extension of the Lex-BFS algorithm discussed in Chapter 4 to weighted graphs. In Section 6.1 we give a short overview of the recognition algorithm and we briefly discuss its relevance in the literature. In Section 6.2 we introduce some preliminaries and we give basic facts about Robinsonian matrices and Robinson orderings. Section 6.3 is devoted to the SFS algorithm. In Section 6.4 we discuss the variant SFS$_+$. In Section 6.5 we present the multisweep algorithm to recognize Robinsonian matrices and we prove its correctness. Finally, in Section 6.6 we conclude the chapter with some possible direction for future work. The content of the chapter is based on our work [79].

6.1 Introduction

In Chapter 5 we have discussed a Robinsonian recognition algorithm entirely based on the Lex-BFS algorithm presented in Chapter 4. In this chapter we introduce instead a new recognition algorithm based on the novel algorithm *Similarity-First Search* (SFS), which represents a generalization of the classical Lex-BFS algorithm to weighted graphs. Intuitively, the SFS algorithm traverses vertices of a weighted graph in such a way that most similar vertices (i.e., corresponding to largest edge weights) are visited first, while still respecting the priorities imposed by previously visited vertices. When applied to an unweighted graph (or equivalently to a 0/1 matrix), the SFS algorithm reduces to Lex-BFS. As for Lex-BFS, the SFS algorithm is entirely based on a unique simple task, namely partition refinement (see Subsection 2.2.3).

We will use the SFS algorithm to define our new Robinsonian recognition algorithm. Specifically, we introduce a multisweep algorithm in the same fashion as for Lex-BFS (see Algorithm 4.4). Each sweep uses the order returned by the previous sweep to break ties in the (weighted) graph search. Our main result is that our multisweep algorithm can recognize after at most $n-1$ sweeps whether a given $n \times n$ matrix A is Robinsonian. Namely we will show that the last sweep is a Robinson ordering of A if and only if the matrix A is Robinsonian. Assuming that the matrix A is nonnegative and given as an adjacency list of an undirected weighted graph with m nonzero entries, our algorithm runs in $O(n^2 + mn \log n)$ time.

As we have already seen in Chapter 4, multisweep algorithms are well studied approaches to recognize classes of (unweighted) graphs and there exist many results on multisweep algorithms which are based on Lex-BFS. Nevertheless, to the best of our knowledge, this is the first work introducing and studying explicitly the properties of a multisweep search algorithm for weighted graphs. The only related idea that we could find is about replacing BFS with Dijkstra's algorithm, which is only briefly mentioned in [39].

The relevance of this work is twofold. First, we reduce the Robinsonian recognition problem to a single extremely simple and basic operation, namely to partition refinement. Hence, even though from a theoretical point of view the algorithm is computationally slower than the optimal one presented in [94], its simplicity makes it easy to implement and thus hopefully will encourage the use and the study of Robinsonian matrices in more practical problems.

Second, we introduce a new (weighted) graph search, which we believe is of independent interest. As we will discuss more in detail at the end of the chapter, this could potentially be used for the recognition of other structured matrices or just as basic operation in the broad field of 'Similarity Search'. In addition, we introduce some new concepts extending analogous notions in graphs, and we develop some combinatorial tools for the study of Robinsonian matrices that we use to analyze our new multisweep algorithm. These concepts could be used to further characterize the Robinsonian structure, e.g., defining a certificate for non-Robinsonian matrices. As an example, we give combinatorial characterizations for the end points (aka anchors) of Robinson orderings.

6.2 Preliminaries

In this section we refresh some notation already introduced in Chapter 2 and we give basic facts about Robinsonian matrices and Robinson orderings. In Subsection 6.2.1 we introduce the concepts of 'path avoiding a vertex' and 'valid vertex'. In Subsection 6.2.2 we give a combinatorial characterization for end points of Robinson orderings (also named 'anchors') and for 'opposite anchors', which will play a crucial role in the rest of the chapter.

As before, we represent a permutation π as a sequence $(x_1, \dots, x_n)$ with $x_1 <_\pi \dots <_\pi x_n$. Hence, $\overline{\pi} = (x_n, x_{n-1}, \dots, x_1)$ denotes the reversed linear order of π. For $U \subseteq V$, $\pi[U]$ denotes the linear order of U obtained by restricting π to U. If π_1 and π_2 are two linear orders on disjoint subsets V_1 and V_2, then $\pi = (\pi_1, \pi_2)$ denotes their concatenation, which is a linear order on $V_1 \cup V_2$. Recall also that an ordered partition $\psi = (B_1, \dots, B_k)$ of V induces a weak linear order on V.

Given a matrix $A \in \mathcal{S}^n$, for $U \subseteq V$, $A[U] = (A_{ij})_{i,j \in U}$ is the principal submatrix of A indexed by U, and for a permutation π of V, $A_\pi = (A_{\pi(i),\pi(j)})_{i,j=1}^n$ is the matrix obtained by symmetrically permuting the rows and columns of A according to π. As already mentioned in Chapter 3, also in this chapter we will deal exclusively with Robinson(ian) similarities. Hence, when speaking of a Robinson(ian) matrix, we mean a Robinson(ian) similarity matrix. Furthermore, as in Chapter 5, it will be convenient to view symmetric matrices as weighted graphs. Namely, any nonnegative symmetric matrix $A \in \mathcal{S}^n$ corresponds to the weighted graph $G = (V = [n], E)$ whose edges are the pairs $\{x, y\}$ with $A_{xy} > 0$, with edge weights A_{xy}. Again, the assumption of nonnegativity can be made without loss of generality and is for convenience only. Accordingly we will often refer to the elements of $V = [n]$ indexing A as vertices (or nodes). For $x \in V$, $N(x) = \{y \in V \setminus \{x\} : A_{xy} > 0\}$ denotes the neighborhood of x in G.

6.2.1 Basics facts

In order to fully understand the motivation for our work, we recall some results about 0/1 Robinsonian matrices already discussed in Chapter 3. Recall, from Theorem 3.2.1, that a symmetric matrix $A \in \{0,1\}^{n \times n}$ is a Robinsonian similarity if and only if it is the (extended) adjacency matrix of a unit interval graph $G = (V = [n], E)$ whose edges are the positions of the nonzero entries of A.

Among the several equivalent characterizations for unit interval graphs, the 3-vertex condition *(iii)* in Theorem 2.3.3 coincides with relation (3.1) for 0/1 matrices. This equivalence and the fact that unit interval graphs can be recognized with a Lex-BFS multisweep algorithm (see Subsection 4.3.2) motivated us to find an extension of Lex-BFS to weighted graphs and to use it to obtain a (simple) multisweep recognition algorithm for Robinsonian matrices.

In what follows we will extend some graph concepts to the general setting of weighted graphs (Robinsonian matrices). Throughout the chapter, we will point out links between our results and some corresponding known results for Lex-BFS applied to graphs. We will refer to the results presented in Chapter 4, which are mostly based on the work in [38].

We now introduce some notions and simple facts about Robinsonian matrices and orderings. Consider a matrix $A \in \mathcal{S}^n$. Given distinct elements $x, y, z \in V$, the triple (x, y, z) is said to be *Robinson* if it satisfies (3.1), i.e., if $A_{xz} \leq \min\{A_{xy}, A_{yz}\}$. Given a set $S \subseteq V$ and $x \in V \setminus S$, we say that x is *homogeneous* with respect to S if $A_{xy} = A_{xz}$ for all $y, z \in S$ (which is an extension of the corresponding notion for graphs, see, e.g., [38]). The following is an easy necessary condition for the Robinson property.

6.2.1 Lemma. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity. Assume that there exists a Robinson ordering π such that $x <_\pi z <_\pi y$. Then $A_{uz} \geq \min\{A_{ux}, A_{uy}\}$ for all $u \neq x, y, z \in [n]$.*

Proof. Indeed, $u <_\pi z$ implies $u <_\pi z <_\pi y$ and thus $A_{uz} \geq A_{uy}$, and $z <_\pi u$ implies $x <_\pi z <_\pi u$ and thus $A_{uz} \geq A_{ux}$. □

We now make a simple observation on how three elements $x, y, z \in V$ may appear in a Robinson ordering π of A depending on their similarities. Namely, if we have that $A_{xz} > \min\{A_{xy}, A_{yz}\}$ then, either y comes before both x and z in π, or y comes after both x and z in π. In other words, if x and z are more similar to each other than to y, then y cannot be ordered between x and z in any Robinson ordering π. Moreover, if $A_{xz} < \min\{A_{xy}, A_{yz}\}$ then, either $x <_\pi y <_\pi z$, or $z <_\pi y <_\pi x$. In other words, if x and z are more similar to y than to each other, then y must be ordered between x and z in any Robinson ordering π.

This observation motivates the following notion of 'path avoiding a vertex', which will play a central role in our discussion. Note that this notion is closely related to the notion of 'path missing a vertex' for Lex-BFS in [38] (see Section 4.3), although it is not equivalent to it when applied to a 0/1 matrix.

6.2.2 Definition. (**Path avoiding a vertex**) Given distinct elements $x, y, z \in V$, a *path from x to z avoiding y* is a sequence $(x = v_0, v_1, \dots, v_{k-1}, v_k = z)$ of elements of V where each triple (v_i, y, v_{i+1}) is not Robinson, i.e.,

$$A_{v_i v_{i+1}} > \min\{A_{yv_i}, A_{yv_{i+1}}\}, \quad \forall\, i = 0, 1, \dots, k-1.$$

The following simple but useful property holds.

6.2.3 Lemma. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix. If there exists a path from x to z avoiding y, then y cannot lie between x and z in any Robinson ordering π of A.*

Proof. Let $(x = v_0, v_1, \dots, v_{k-1}, v_k = z)$ be a path from x to z avoiding y. Then, by definition, we have $A_{v_i v_{i+1}} > \min\{A_{yv_i}, A_{yv_{i+1}}\}$ for all $i = 0, 1, \dots, k-1$, and thus y cannot appear between v_i and v_{i+1} in any Robinson ordering π. Hence y cannot lie between x and z in any Robinson ordering π. □

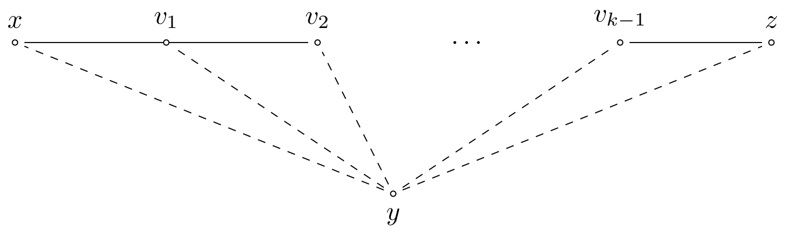


Figure 6.1: A path from x to z avoiding y: each continuous line indicates a value which is strictly larger than the minimum of the two adjacent dotted lines.

We now introduce the notion of 'valid vertex' which we will use in the next section to characterize end points of Robinson orderings.

6.2.4 Definition. (**Valid vertex**) Given a matrix $A \in \mathcal{S}^n$, an element $z \in V$ is said to be *valid* if, for any distinct elements $u, v \in V \setminus \{z\}$, there do not exist both a path from u to z avoiding v and a path from v to z avoiding u.

Observe that, if $z \in V$ is a valid vertex of a matrix A and $S \subseteq V$ is a subset containing z, then z is also a valid vertex of $A[S]$. It is easy to see that, for a 0/1 matrix, the above definition of valid vertex coincides with the notion of valid vertex for Lex-BFS [38] (see Section 4.3).

Consider, for example, the following matrix (already ordered in a Robinson form):

$$A = \begin{array}{c} \\ \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \\ \mathbf{e} \\ \mathbf{f} \\ \mathbf{g} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} & \mathbf{f} & \mathbf{g} \end{array} \\ \begin{pmatrix} * & 7 & 6 & 0 & 0 & 0 & 0 \\ & * & 7 & 3 & 2 & 1 & 1 \\ & & * & 7 & 2 & 2 & 1 \\ & & & * & 3 & 3 & 3 \\ & & & & * & 7 & 5 \\ & & & & & * & 6 \\ & & & & & & * \end{pmatrix} \end{array}$$

Then the vertex d is not valid. Indeed, for the two vertices a and g, there exist a path from a to d avoiding g and a path from g to d avoiding a; namely the path (d, b, a) avoids g and the path (d, b, g) avoids a (see Figure 6.2).

6.2.2 Characterization of anchors

In this subsection we introduce the notion of '(opposite) anchors' of a Robinsonian matrix and then we give characterizations in terms of valid vertices. The notion of anchor was used for unit interval graphs in [35] and it is the analog of the notion of end-vertex for interval graphs [38] (see Secton 4.3).

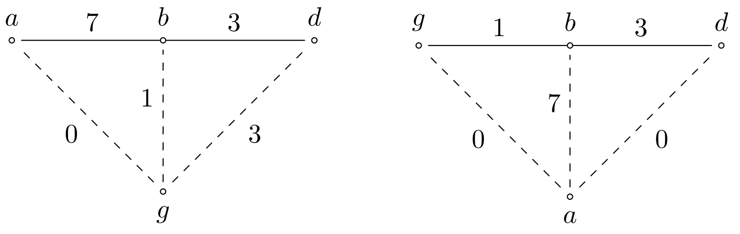


Figure 6.2: Element d is not valid.

6.2.5 Definition. (**Anchor**) Given a Robinsonian similarity $A \in \mathcal{S}^n$, a vertex $a \in [n]$ is called an *anchor* of A if there exists a Robinson ordering π of A whose last vertex is a. Moreover, two distinct vertices a, b are called *opposite anchors* of A if there exists a Robinson ordering π of A with a as first vertex and b as last vertex.

Hence, an anchor is an end point of a Robinson ordering. Clearly, every Robinsonian matrix has at least one pair of opposite anchors. It is not difficult to see that every anchor must be valid. We now show that conversely every valid vertex is an anchor. This is the analog of [34, Lemma 2] for Lex-BFS over interval graphs (see Theorem 4.3.3).

6.2.6 Theorem. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix. Then a vertex $z \in V$ is an anchor of A if and only if it is valid.*

Proof. ($\Rightarrow$) Assume z is an anchor of A and let π be a Robinson ordering of A with z as last element. Suppose for contradiction that, for some elements $u, v \in V$, there exist both a path P from u to z avoiding v and a path Q from v to z avoiding u. Using Lemma 6.2.3 and the path P, we obtain that that v lies before u or after z in π, and using the path Q we obtain that u lies before v or after z in π. As z is the last element of π, we must have $v <_\pi u$ in the first case and $u <_\pi v$ in the second case, which is impossible.

($\Leftarrow$) Conversely, assume that z is valid; we show that z is an anchor of A. The proof is by induction on the size n of the matrix A. The result holds clearly when $n = 2$. So we now assume $n \geq 3$ and that the result holds for any Robinsonian matrix of order at most $n-1$. We need to construct a Robinson ordering π' of A with z as last vertex. For this we consider a Robinson ordering π of A. We let x denote its first element and y denote its last element. If $z = x$ or $z = y$, then we would be done. Hence we may assume $x <_\pi z <_\pi y$. For any $v <_\pi z$, we denote by $P_\pi(v, z)$ the path from v to z consisting of the sequence of vertices appearing consecutively between v and z in π.

We now define the following two sets:

$$\mathcal{B} = \{v <_\pi z : P_\pi(v, z) \text{ avoids } y\}, \quad \mathcal{C} = \{v <_\pi z : v \notin \mathcal{B}\}. \tag{6.1}$$

Next we show their following properties, which will be useful to conclude the proof.

6.2.7 Claim. *The following holds:*

(i) For any $v \in \mathcal{B}$, $A_{vy} = A_{yz}$.

(ii) If $v \in \mathcal{B}$ and $v <_\pi u <_\pi z$, then $u \in \mathcal{B}$.

(iii) Any element $v \in \mathcal{C}$ is homogeneous with respect to $V \setminus \mathcal{C}$, i.e., $A_{vw} = A_{vw'}$ for all $w, w' \in V \setminus \mathcal{C}$.

Proof. (i) As $v <_\pi z <_\pi y$, then $A_{vy} \leq A_{yz}$. We show that equality holds. Suppose not, i.e., $A_{vy} < A_{yz}$. Then $Q = (y, z)$ is a path from y to z avoiding v. Since $v \in \mathcal{B}$, $P = P_\pi(v, z)$ is a path from v to z avoiding y, and thus the existence of the paths P, Q contradicts the assumption that z is valid. Hence we must have $A_{vy} = A_{yz}$.

(ii) If $v \in \mathcal{B}$ then $P_\pi(v, z)$ avoids y and thus the subpath $P_\pi(u, z)$ also avoids y, which implies $u \in \mathcal{B}$.

(iii) Let $u \in \mathcal{B}$ denote the element of $\mathcal{B}$ appearing first in the Robinson ordering π. Then, for any $v \in \mathcal{C}$, $v <_\pi u <_\pi y$ and thus $A_{vy} \leq A_{vu}$ by definition of Robinson ordering. Hence, in order to show that v is homogeneous with respect to $V \setminus \mathcal{C}$, it suffices to show that $A_{vu} = A_{vy}$ (as, using the Robinson ordering property, this would in turn imply that $A_{vw} = A_{vw'}$ for all $w, w' \in V \setminus \mathcal{C}$). Suppose for contradiction that there exists $v \in \mathcal{C}$ such that $A_{vu} \neq A_{vy}$, and let v denote the element of $\mathcal{C}$ appearing last in π with $A_{vu} \neq A_{vy}$.

Then $A_{vu} > A_{vy}$ and the path (v, u) avoids y. Since $P_\pi(u, z)$ is a path from u to z avoiding y (because $u \in \mathcal{B}$), then the path $P = \{v\} \cup P_\pi(u, z)$ (obtained by concatenating (v, u) and $P_\pi(u, z)$) is a path from v to z avoiding y. This implies that v and u cannot be consecutive in π, as otherwise we would have $v \in \mathcal{B}$, contradicting the fact that $v \in \mathcal{C}$. Hence, there exists $v' \in \mathcal{C}$ such that $v <_\pi v' <_\pi u$. By the maximality assumption on v, it follows that $A_{v'u} = A_{v'y}$.

As z is valid and $P = \{v\} \cup P_\pi(u, z)$ is a path from v to z avoiding y, it follows that no path from y to z can avoid v. In particular, the path (y, z) does not avoid v and thus it must be $A_{yz} \leq \min\{A_{vy}, A_{vz}\}$. Recall that we assumed $A_{vu} > A_{vy}$. As $v <_\pi v' <_\pi u <_\pi z <_\pi y$, combining the above inequalities with the inequalities coming from the Robinson ordering π, we obtain $A_{v'y} \leq A_{yz} \leq A_{vy} < A_{vu} \leq A_{v'u}$, which contradicts the equality $A_{v'u} = A_{v'y}$. □

We now turn to the set of vertices coming after z in π. Symmetrically with respect to z, we can define the analogues of the sets $\mathcal{C}, \mathcal{B}$ defined in (6.1), which we denote by $\mathcal{C}', \mathcal{B}'$. For this replace π by its reverse ordering $\overline{\pi}$ and y by x (the first element of π and thus the last element of $\overline{\pi}$), i.e., set

$$\mathcal{B}' = \{v >_\pi z : P_\pi(z, v) \text{ avoids } x\}, \quad \mathcal{C}' = \{v >_\pi z : v \notin \mathcal{B}'\}.$$

To recap, we have that $\pi = (\mathcal{C}, \mathcal{B}, z, \mathcal{B}', \mathcal{C}')$. Recall that x and y are respectively the first and the last vertex in π. Note that it cannot be that $\mathcal{C} = \mathcal{C}' = \emptyset$, as this would imply that $x \in \mathcal{B}$ and $y \in \mathcal{B}'$, and thus this would contradict the fact that z is valid (using the definition of the two sets $\mathcal{B}$ and $\mathcal{B}'$). Therefore, we may assume (without loss of generality) that $\mathcal{C} \neq \emptyset$. Let v be the vertex of $\mathcal{C}$ appearing last in the Robinson ordering π. By Claim 6.2.7 (iii), v is homogeneous with respect to the set $S = V \setminus \mathcal{C}$, i.e., all entries A_{vw} take the same value for any $w \in S$.

Consider the matrix $A[S]$, the principal submatrix of A with rows and columns in S. As $|S| \leq n-1$ and z is valid (also with respect to $A[S]$), we can conclude using the induction assumption that z is an anchor of $A[S]$. Hence, there exists a Robinson ordering σ of $A[S]$ admitting z as last element.

Now, consider the linear order $\pi' = (\pi[\mathcal{C}], \sigma)$ of V obtained by concatenating first the order π restricted to $\mathcal{C} = V \setminus S$ and second the linear order σ of S. Using the fact that every vertex in $\mathcal{C}$ is homogeneous to all elements of S, we can conclude that the new linear order π' is a Robinson ordering of the matrix A. As z is the last element of π', this shows that z is an anchor of A and thus concludes the proof. $\square$

The above proof can be extended to characterize pairs of opposite anchors.

6.2.8 Theorem. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix. Two distinct vertices $z_1, z_2 \in [n]$ are opposite anchors of A if and only if they are both valid and there does not exist a path from z_1 to z_2 avoiding any other vertex.*

Proof. ($\Rightarrow$) Assume that z_1 and z_2 are opposite anchors. Then they are both anchors and thus, in view of Theorem 6.2.6, they are both valid. Let π a Robinson ordering starting with z_1 and ending with z_2. Suppose, for the sake of contradiction, that there exists a vertex x and a path from z_1 to z_2 avoiding x. Then, by Lemma 6.2.3, x cannot lie in π between z_1 and z_2, yielding a contradiction.

($\Leftarrow$) Assume that z_1 and z_2 are valid and that there does not exist a path from z_1 to z_2 avoiding any other vertex. We show that they are opposite anchors. Consider a Robinson ordering π of A whose first element is z_1 and call y its last element. If $y = z_2$ then we are done. Hence, we may assume that $z_1 <_\pi z_2 <_\pi y$. As in the proof of Theorem 6.2.6, for any $v <_\pi z_2$, we denote by $P_\pi(v, z_2)$ the path from v to z_2 consisting of the sequence of vertices appearing consecutively between v and z_2 in π. Then, we can define the sets as in (6.1) in the proof of Theorem 6.2.6, where z is replaced by z_2, i.e.,:

$$\mathcal{B} = \{v <_\pi z_2 : P_\pi(v, z_2) \text{ avoids } y\}, \quad \mathcal{C} = \{v <_\pi z_2 : v \notin \mathcal{B}\}.$$

By assumption, $z_1 \notin \mathcal{B}$, else $P_\pi(z_1, z_2)$ would avoid y, contradicting the nonexistence of a path from z_1 to z_2 avoiding any other vertex. Therefore $z_1 \in \mathcal{C}$ and thus $\mathcal{C} \neq \emptyset$. Let $S = V \setminus \mathcal{C}$. Using the same reasoning as in the proof of Theorem 6.2.6, we can now conclude that one can find a Robinson ordering σ of $A[S]$, where S

contains all the elements coming after the last element of $\mathcal{C}$ in π. The new linear order $\pi' = (\pi[\mathcal{C}], \sigma)$ of V obtained by concatenating first the order π restricted to $\mathcal{C} = V \setminus S$ and second the linear order σ of S is then a Robinson ordering of A whose first element is z_1 and whose last element is z_2, concluding the proof. $\square$

6.3 The SFS algorithm

In this section we introduce our new Similarity-First Search (SFS) algorithm. We first describe the algorithm in detail in Subsection 6.3.1 and provide a 3-point characterization of SFS orderings in Subsection 6.3.2. Then in Subsection 6.3.3 we discuss some properties of SFS orderings of Robinsonian matrices. Specifically, we introduce the fundamental 'Path Avoiding Lemma' (Lemma 6.3.6) which will be used repeatedly throughout the chapter. Finally, in Subsection 6.3.4 we introduce the notion of 'good SFS ordering' and we show fundamental properties of end points of (good) SFS orderings, namely that they are (opposite) anchors of Robinsonian matrices.

6.3.1 Description

The SFS algorithm is a generalization of Lex-BFS for weighted graphs. As we will remark later, when applied to a 0/1 matrix, the SFS algorithm coincides with Lex-BFS. Roughly speaking, the basic idea is to traverse a weighted graph by visiting first vertices which are similar to each other (i.e., corresponding to an edge with largest weight) but respecting the priorities imposed by previously visited vertices. As for the algorithm presented in Chapter 5, also the SFS algorithm is heavily based on the partition refinement procedure describe in Subsection 2.2.3.

In fact, in our new SFS algorithm, we basically operate a sequence of partition refinements steps as in Algorithm 4.3. However, instead of splitting each class into two subsets, we will split into several subsets. Specifically, given two ordered partitions ϕ and ψ, the output will be a new ordered partition which, roughly speaking, is obtained by splitting each class of ϕ into its intersections with the classes of ψ. The formal definition is as follows.

6.3.1 Definition. (**Refine**) Let $\phi = (B_1, \dots, B_r)$ and $\psi = (C_1, \dots, C_s)$ be two ordered partitions of a set V and a subset $W \subseteq V$, respectively. *Refining ϕ by ψ* creates the new ordered partition of V, denoted by *Refine*(ϕ, ψ), obtained by replacing in ϕ each class B_i by the ordered sequence of classes $(B_i \cap C_1, \dots, B_i \cap C_s, B_i \setminus (C_1 \cup \dots \cup C_s) = B_i \setminus W)$ and keeping only nonempty classes.

We will use this partition refinement operation in the case when the partition ψ is obtained by partitioning for decreasing values the elements of the neighborhood $N(p)$ of a given element p, according to the following definition.

6.3.2 Definition. (**Similarity partition**) Consider $A \in \mathcal{S}^n$ nonnegative and an element $p \in [n]$. Let $a_1 > \ldots > a_s > 0$ be the distinct values taken by the entries A_{px} of A for $x \in N(p) = \{y \in [n] : A_{py} > 0\}$ and, for $i \in [s]$, set $C_i = \{x \in N(p) : A_{px} = a_i\}$. Then we define $\psi_p = (C_1, \ldots, C_s)$, which we call the *similarity partition* of $N(p)$ with respect to p.

Note that the above concept of Similarity partition depends on how we define $N(p)$. We will discuss in Chapter 8 an alternative algorithm, called ϵ-SFS, where a different $N(p)$ is considered.

We can now describe the SFS algorithm. The input is a nonnegative matrix $A \in \mathcal{S}^n$ and the output is an ordering σ of the set $V = [n]$, that we call a *SFS ordering* of A. As in any general graph search algorithm (see Section 4.1), the central idea of the SFS algorithm is that, at each iteration, a special vertex (called the *pivot*) is chosen among the subset of unvisited vertices (i.e., the subset of vertices that have not been a pivot in prior iterations). Such vertices are ordered in a queue which defines the priorities for visiting them. Intuitively, the pivot is chosen as the most similar to the visited vertices, but respecting the visiting priorities imposed by previously visited vertices.

Algorithm 6.1: *SFS*(A)

```
input: a nonnegative matrix $A \in \mathcal{S}^n$
output: a linear order $\sigma$ of $[n]$

1  $\phi = (V)$
2  for $i = 1, \ldots, n$ do
3      let $S$ be the first class of $\phi$
4      choose $p$ arbitrarily in $S$
5      $\sigma(p) = i$
6      remove $p$ from $\phi$
7      let $N(p)$ be the set of vertices $y \in \phi$ with $A_{py} > 0$
8      let $\psi_p$ be the similarity partition of $N(p)$ with respect to $p$
9      $\phi$ =Refine $(\phi, \psi_p)$
10 return: $\sigma$
```

We now discuss in detail how the algorithm works. In the beginning, all vertices in V are unvisited, i.e., the queue ϕ of unvisited vertices is initialized with the unique class V.

At the iteration i, we are given an element p_{i-1} (which is the pivot chosen at iteration $i-1$) and a queue $\phi(p_{i-1}) = (B_1, \ldots, B_r)$, which is an ordered partition of the set of unvisited vertices. There are two main tasks to perform: the first task is to select the new pivot p_i, and the second task is to update the queue $\phi(p_{i-1})$ in order to obtain the new queue $\phi(p_i)$.

The first task is carried out as follows. As in the standard Lex-BFS, we denote by S the *slice* induced by p_{i-1} (i.e., the last visited vertex), which consists of the vertices among which to choose the next pivot p_i. The slice S coincides exactly with the first class B_1 of $\phi(p_{i-1})$. We distinguish two cases depending on the size of the slice S. If $|S| = 1$, then the new pivot p_i is the unique element of the slice S. If $|S| > 1$, we say that we have *ties* and, in the general version of the SFS algorithm, we break them arbitrarily. We will see in Section 6.4 a variant of SFS (denoted by SFS$_+$) where such ties are broken using a linear order given as additional input to the algorithm. Once the new pivot p_i is chosen, we mark it as visited (i.e., we remove it from the queue $\phi(p_{i-1})$) and we set $\sigma(p_i) = i$ (i.e., we let p_i appear at position i in σ).

The second task is the update of the queue $\phi(p_{i-1})$, which can be done as follows. Intuitively, we update $\phi(p_{i-1})$ according to the similarities of p_i with respect to the unvisited vertices and compatibly with the queue order. Specifically, first we compute the similarity partition $\psi_{p_i} = (C_1, \ldots, C_s)$ of the neighborhood $N(p_i)$ of p_i among the unvisited vertices (see Definition 6.3.2). Second, we refine the ordered partition $\phi(p_{i-1}) \ \setminus \ p_i = (B_1 \setminus \{p_i\}, B_2, \ldots, B_r)$ by the ordered partition ψ_{p_i} (see Definition 6.3.1). The resulting ordered partition is the ordered partition $\phi(p_i)$.

Note that if the matrix has only 0/1 entries then the similarity partition ψ_{p_i} has only one class, equal to the neighborhood of p_i intersection the unvisited vertices. Hence, the refinement procedure defined in Definition 6.3.1 simply reduces to the partition refinement operation defined in [61] for Lex-BFS (see Subsection 2.2.3). This is why Lex-BFS is actually a special case of SFS applied to 0/1 matrices.

Note also that, by construction, each class of the queue $\phi(p_i)$ is an interval of σ (i.e., the elements of the class are consecutive in σ). Furthermore, each of the visited vertices $p_1, \ldots, p_i$ is homogeneous to every class of the queue $\phi(p_i)$. For a concrete example of how the SFS algorithm works, we refer the reader to the example in Subsection 6.5.5.

6.3.2 Characterization of SFS orderings

In this section we characterize the linear orders returned by the SFS algorithm in terms of a 3-point condition. This characterization applies to any (not necessarily Robinsonian) matrix and it is the analog of Theorem 4.2.1.

6.3.3 Theorem. *Given a matrix $A \in \mathcal{S}^n$, an ordering σ of $[n]$ is a SFS ordering of A if and only if the following condition holds:*

$$\begin{array}{l}\textit{For all } x, y, z \in [n] \textit{ such that } A_{xz} > A_{xy} \textit{ and } x <_\sigma y <_\sigma z, \\ \textit{there exists } u \in [n] \textit{ such that } u <_\sigma x \textit{ and } A_{uy} > A_{uz}.\end{array} \tag{6.2}$$

Proof. ($\Rightarrow$) Suppose σ is a SFS ordering of A. Assume $x <_\sigma y <_\sigma z$ and $A_{xz} > A_{xy}$, but $A_{uz} \geq A_{uy}$ for each $u <_\sigma x$. Assume first that $A_{uz} > A_{uy}$ for some

$u <_\sigma x$ and let u be the first such vertex in σ. Then $A_{wz} = A_{wy}$ for each $w <_\sigma u$, and thus y, z are in the same class of the queue of unvisited vertices when u is chosen as pivot. Therefore, z would be ordered before y in σ when computing the similarity partition of $N(u)$, i.e., we would have $z <_\sigma y$, a contradiction. Hence, one has $A_{uz} = A_{uy}$ for each $u <_\sigma x$. This implies that y, z are in the same class of the queue of unvisited vertices before x is chosen as pivot. Hence, when x is chosen as pivot, as $A_{xz} > A_{xy}$, when computing the similarity partition of $N(x)$ we would get $z <_\sigma y$, which is again a contradiction.

($\Leftarrow$) Assume that the condition (6.2) of the theorem holds, but σ is not a SFS ordering. Let a denote the first vertex of σ. Let τ be a SFS ordering of A starting at a with the largest possible initial overlap with σ. Say, σ and τ share the same initial order $(a, a_1, \dots, a_r)$ and they differ at the next position. Then we have that $\sigma = (a, a_1, \dots, a_r, y, \dots, z, \dots,)$ and $\tau = (a, a_1, \dots, a_r, z, \dots, y, \dots)$ with $y \neq z$.

In the SFS ordering τ, the two elements y, z do not lie in the slice of the pivot a_r. Indeed, if y, z would lie in the slice of a_r then one could select y as the next pivot instead of z, which would result in another SFS ordering τ' starting at a and with a larger overlap with σ than τ. Hence, there exists $i \leq r$ such that $A_{a_i z} > A_{a_i y}$. Since $a_i <_\sigma y <_\sigma z$ then applying the condition (6.2) to σ, we deduce that there exists $j < i$ such that $A_{a_j y} > A_{a_j z}$. Now, we have $a_j <_\tau z <_\tau y$ with $A_{a_j y} > A_{a_j z}$. As τ is a SFS ordering, as we have just shown it must satisfy the condition (6.2) and thus there must exist an index $k < j$ such that $A_{a_k z} > A_{a_k y}$. Hence, starting from an index $i \leq r$ for which $A_{a_i z} > A_{a_i y}$, we have shown the existence of another index $k < i \leq r$ for which $A_{a_k z} > A_{a_k y}$. Iterating this process, we reach a contradiction. $\square$

One can easily show that if σ is a SFS ordering of V and $S \subseteq V$ is a subset such that any element $x \notin S$ is homogeneous to S, then the restriction $\sigma[S]$ of σ to S is a SFS ordering of $A[S]$. Since, by construction, each vertex before a slice S in σ is homogeneous to S, a direct consequence of Theorem 6.3.3 is that the restriction of σ to any slice encountered throughout a SFS ordering σ, is a SFS ordering too.

6.3.3 The Path Avoiding Lemma

In this section we discuss a fundamental lemma which we call the 'Path Avoiding Lemma'. It will play a crucial role throughout the chapter and, in particular, for the characterization of anchors. Differently from the analysis in the previous section, where we did not make any assumption on the structure of the matrix A, the Path Avoiding Lemma states some important properties of SFS orderings when the input matrix is Robinsonian.

Before stating this lemma, we need to investigate in more detail the refinement step in the SFS algorithm. An important operation in the Refine task in Algo-

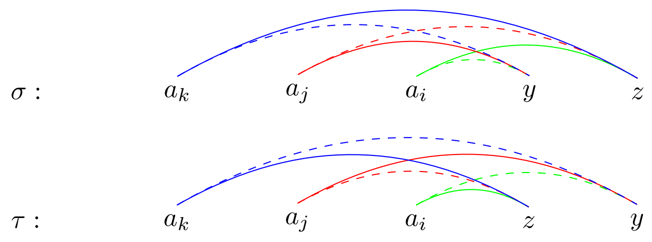


Figure 6.3: Illustration of the proof of Theorem 6.3.3 (the dotted lines indicate similarities that are strictly smaller than the continuous ones of the same color).

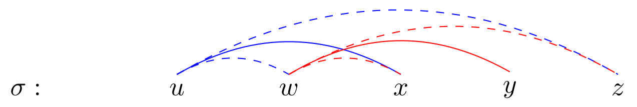


Figure 6.4: Illustrating the proof of Theorem 6.3.4: (w, x, y, z) and (u, w, x, z) are bad quadruples (the dotted lines indicate similarities that are strictly smaller than the continuous ones of the same color).

rithm 6.1 is the splitting procedure of each class of the queue ϕ. The following notion of 'vertex splitting a pair of vertices' is useful to understand it. Consider an order $\sigma = \text{SFS}(A)$ and vertices $x <_\sigma y <_\sigma z$, where $x = p_i$ is the pivot chosen at the ith iteration in Algorithm 6.1. We say that x *splits* y and z if x is the first pivot for which y and z do not belong to the same class in the queue ordered partition $\phi(p_i)$. Recall that $\phi(p_i)$ denotes the queue of unvisited nodes induced by pivot p_i, i.e., at the end of iteration i (after the refinement step). Hence, saying that y, z are split by x means that y, z belong to a common class B_j of $\phi(p_{i-1})$ and that they belong to distinct classes B_h, B_k of $\phi(p_i)$, where $y \in B_h$, $z \in B_k$ and B_h comes before B_k in $\phi(p_i)$. Equivalently, $x = p_i$ splits y and z if $A_{xy} > A_{xz}$ and $A_{uy} = A_{uz}$ for all $u <_\sigma p_i$.

Then, we say that two vertices $y <_\sigma z$ are *split* in σ if they are split by some vertex $x <_\sigma y$. When y and z are not split in σ, we say that they are *tied*. In this case, ties must be broken between y and z. In the SFS algorithm ties are broken arbitrarily. In Section 6.4 we will see the variation SFS_+ of SFS where ties are broken using a linear order τ given as input together with the matrix A. The following lemma will be used as base case for proving the Path Avoiding Lemma.

6.3.4 Lemma. *Assume that $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix and let $\sigma = \text{SFS}(A)$. Assume that $x <_\sigma y <_\sigma z$ and that there exists a Robinson ordering π of A such that $x <_\pi z <_\pi y$. Then y and z are not split in σ by any vertex $u \leq_\sigma x$. That is, $A_{uy} = A_{uz}$ for all $u \leq_\sigma x$.*

Proof. We first show that y, z are not split by any vertex w occurring before x in σ. Suppose, for contradiction, that y, z are split by a vertex $w <_\sigma x$. Hence,

$A_{wy} > A_{wz}$. This implies $z <_\pi w$ for, otherwise, $w <_\pi z <_\pi y$ would imply $A_{wy} \leq A_{wz}$, a contradiction. Hence we have $w <_\sigma x <_\sigma z$ and $x <_\pi z <_\pi w$. Because π is a Robinson ordering, we get $A_{wz} \geq A_{wx}$ and thus $A_{wy} > A_{wz} \geq A_{wx}$. Therefore, the quadruple (w, x, y, z) satisfies the following properties (a)-(d): (a) $w <_\sigma x <_\sigma y <_\sigma z$, (b) $x <_\pi z <_\pi w$ for some Robinson ordering π, (c) w is the pivot splitting y, z, and (d) $A_{wy} > A_{wx}, A_{wz}$ Call any quadruple satisfying (a)-(d) a *bad quadruple.*

We now show that if (w, x, y, z) is a bad quadruple, then there exists $u <_\sigma w$ for which (u, w, x, z) is also a bad quadruple. Hence, iterating we will get a contradiction. We now proceed to show the existence of $u <_\sigma w$ for which (u, w, x, z) is also a bad quadruple. Since $A_{wx} < A_{wy}$, the vertices x, y are already split before w becomes a pivot; otherwise, if they would belong to the same class when w is chosen as new pivot, then we would get $y <_\sigma x$. Let $u = p_i$ the pivot splitting x, y, i.e., $u <_\sigma w$ and $A_{ux} > A_{uy}$. Thus x, y belong to the same class (say) $B \in \phi(p_{i-1})$ when u is chosen as new pivot at iteration i, but in different classes of $\phi(p_i)$. Since w is the pivot splitting y, z and $u <_\sigma w$, it follows that y, z belong to the same class when u is chosen as pivot, and thus $x, y, z \in B$. Therefore u is also the pivot splitting x and z and thus $A_{ux} > A_{uy} = A_{uz}$. In turn this implies that $u <_\pi z$ for, otherwise, $x <_\pi z <_\pi u$ would imply $A_{ux} \leq A_{uz}$, a contradiction. Therefore, $u <_\pi z <_\pi w$ and by definition of Robinson ordering we have $A_{uw} \leq A_{uz}$ and, as $A_{ux} > A_{uz}$, this implies that $A_{uw} < A_{ux}$. Summarizing, we have shown that the quadruple (u, w, x, z) is bad since it satisfies the conditions (a)-(d): (a) $u <_\sigma w <_\sigma x <_\sigma z$, (b) $w <_{\overline{\pi}} z <_{\overline{\pi}} u$ for the Robinson ordering $\overline{\pi}$, (c) u splits x and z, and (d) $A_{ux} > A_{uw}, A_{uz}$. Thus we have shown that there cannot exist a bad quadruple and therefore that y, z are not split by any vertex w appearing before x in σ.

We now conclude the proof of the lemma by showing that y, z are also not split by x. For this, we need to show that $A_{xz} = A_{xy}$. Suppose for contradiction that $A_{xz} \neq A_{xy}$. As $x <_\pi z <_\pi y$, it can only be that $A_{xz} > A_{xy}$. Let $x = p_i$, i.e., x is the pivot chosen at iteration i of Algorithm 6.1. Since we have just shown that y, z are not split before x, then at the iteration i when x is chosen as pivot, we would order $z <_\sigma y$ as $A_{xz} > A_{xy}$, which is a contradiction because $y <_\sigma z$ by assumption. □

A first direct consequence of Lemma 6.3.4 is the following.

6.3.5 Corollary. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix, let $\sigma = SFS(A)$, and consider distinct elements $x, y, z \in V$ such that $x <_\sigma y <_\sigma z$. The following holds:*

(i) $A_{xy} \geq \min\{A_{xz}, A_{yz}\}$.

(ii) If $x <_\pi z <_\pi y$ for some Robinson ordering π, then the path $P = (x, z)$ does not avoid y.

Proof. (i) Assume, for contradiction, that $A_{xy} < \min\{A_{xz}, A_{yz}\}$. Pick a Robinson ordering π of A such that $x <_\pi y$. Then we must have $x <_\pi z <_\pi y$. Indeed, if $x <_\pi y <_\pi z$ then we would have $A_{xy} \geq A_{xz}$, and if $z <_\pi x <_\pi y$ we would have $A_{xy} \geq A_{yz}$, leading in both cases to a contradiction. Applying Lemma 6.3.4, we conclude that $A_{xy} = A_{xz}$, contradicting our assumption that $A_{xy} < A_{xz}$.

(ii) If (x, z) avoids y then $A_{xz} > \min\{A_{xy}, A_{yz})$, where $\min\{A_{xy}, A_{yz}) = A_{xy}$ since $x <_\pi z <_\pi y$. Hence this contradicts Lemma 6.3.4. □

Note that the above result is the analog of the 'P_3-rule' for chordal graphs in [38, Thm 3.12] (see Theorem 4.3.1). The next lemma strengthens the result of Corollary 6.3.5 (ii), by showing that there cannot exist *any* path from x to z avoiding y and appearing fully before z in σ. We will refer to Lemma 6.3.6 below as the 'Path Avoiding Lemma', also abbreviated as (PAL) for ease of reference in the rest of the chapter.

6.3.6 Lemma. ***(Path Avoiding Lemma (PAL))*** *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix and let $\sigma = \mathrm{SFS}(A)$. Consider distinct elements $x, y, z \in V$ such that $x <_\sigma y <_\sigma z$. If $x <_\pi z <_\pi y$ for some Robinson ordering π, then there does not exist a path $P = (x, u_1, \ldots, u_k, z)$ from x to z avoiding y and such that $u_1, \ldots, u_k <_\sigma z$.*

Proof. The proof is by induction on the length $|P| = k + 2$ of the path P. The base case is $|P| = 2$, i.e., $P = (x, z)$, which is settled by Corollary 6.3.5. Assume then, for contradiction, that there exists a path $P = (x, u_1, \ldots, u_k, z)$ from x to z avoiding y with $u_1, \ldots, u_k <_\sigma z$ and $|P| \geq 3$, i.e., $k \geq 1$. Let us call a path Q *short* if it is shorter than P, i.e., if $|Q| < |P|$. By the induction assumption, we know that the following holds:

> If $u <_\sigma v <_\sigma w$ and $u <_\tau w <_\tau v$ for some Robinson ordering τ, then no short path $Q = (u, v_1, \ldots, v_r, w)$ from u to w avoiding v and with $v_1, \ldots, v_r <_\sigma w$ exists. (6.3)

Set $u_0 = x$ and $u_{k+1} = z$. As P avoids y, the following relations hold:

$$A_{u_{i-1}u_i} > \min\{A_{yu_{i-1}}, A_{yu_i}\} \text{ for all } i \in [k+1]. \tag{6.4}$$

Since $x <_\sigma y <_\sigma z$ and $x <_\pi z <_\pi y$, then in view of Lemma 6.3.4 we have $A_{xy} = A_{xz}$. Furthermore, we know that $u_1, \ldots, u_k <_\sigma z$ by assumption. Recall that $\overline{\pi}$ denotes the reversal linear order of π. In order to conclude the proof, we use the following claim.

6.3.7 Claim. *$u_i <_\pi x$ and $y <_\sigma u_i$ for each $i \in [k]$.*

Proof. The proof is by induction on $i \geq 1$. For $i = 1$ we have to show that

$$u_1 <_\pi x \text{ and } y <_\sigma u_1. \tag{6.5}$$

We first show that $u_1 <_\pi x$. Suppose this is not the case and $x <_\pi u_1$. Recall that in view of (6.4) for $i = 1$ we have $A_{xu_1} > \min\{A_{yx}, A_{yu_1}\}$ and thus the path (x, u_1) avoids y. Hence, since $x <_\pi y$ and $A_{xu_1} > \min\{A_{yx}, A_{yu_1}\}$, in view of Lemma 6.2.3, y cannot appear between x and u_1 in any Robinson ordering and thus it must also be that $u_1 <_\pi y$. We then have two possibilities, depending on whether u_1 comes before or after z in π.

(i) Assume first that u_1 appears before z in π. Then we have $x <_\pi u_1 <_\pi z <_\pi y$. We discuss where can u_1 appear in σ. If $u_1 <_\sigma y$ then we have $u_1 <_\sigma y <_\sigma z$, $u_1 <_\pi z <_\pi y$, and $(u_1, \dots, u_k, z)$ is a short path from u_1 to z avoiding y with $u_2, \dots, u_k <_\sigma z$, which contradicts (6.3). Hence, $y <_\sigma u_1$ in which case we have $x <_\sigma y <_\sigma u_1$, $x <_\pi u_1 <_\pi y$, and (x, u_1) is a short path from x to u_1 avoiding y, which contradicts again (6.3).

(ii) Assume now that u_1 appears after z in π. Then we have $x <_\pi z <_\pi u_1 <_\pi y$. By (6.4) applied to $i = 1$ and using the Robinson ordering π, we then have that $A_{u_1x} > \min\{A_{yx}, A_{yu_1}\} = A_{yx}$. Recall that $A_{xy} = A_{xz}$. Then $A_{u_1x} > A_{xz}$. On the other hand, by the Robinson property of π, $A_{xu_1} \leq A_{xz}$, yielding a contradiction.

Therefore we have shown that $u_1 <_\pi x$. Next, we show that $y <_\sigma u_1$. Suppose not, i.e., $u_1 <_\sigma y$. Then we would have $u_1 <_\sigma y <_\sigma z$ and, as just shown, $u_1 <_\pi z <_\pi y$, while $(u_1, \dots, u_k, z)$ is a short path from u_1 to z avoiding y with $u_2, \dots, u_k <_\sigma z$. This contradicts (6.3) and thus shows $y <_\sigma u_1$, which concludes the proof for the base case $i = 1$.

Assume now that $i \geq 2$ and that $u_j <_\pi x$ and $y <_\sigma u_j$ for all $1 \leq j \leq i - 1$ by induction. We show that $u_i <_\pi x$ and $y <_\sigma u_i$. First we show $u_i <_\pi x$. Suppose, for the sake of contradiction, that $x <_\pi u_i$. By (6.4), the path $(u_i, \dots, u_k, z)$ is a path from u_i to z avoiding y with $u_{i+1}, \dots, u_k <_\sigma z$. Hence, since $z <_\pi y$ in view of Lemma 6.2.3 it must be also $u_i <_\pi y$, because y cannot appear between z and u_1 in any Robinson ordering. We then have two possibilities to discuss, depending whether u_i comes before or after z in π.

(i) Assume that u_i appears before z in π. Then $u_1, \dots, u_{i-1} <_\pi x <_\pi u_i <_\pi z <_\pi y$. First we claim that $y <_\sigma u_i$. Indeed, if by contradiction $u_i <_\sigma y$, then we would have: $u_i <_\sigma y <_\sigma z$ and $u_i <_\pi z <_\pi y$, while $(u_i, \dots, u_k, z)$ is a short path from u_i to z avoiding y with $u_{i+1}, \dots, u_k <_\sigma z$, contradicting (6.3).

Hence, $y <_\sigma u_i$ holds. Recall that $y <_\sigma u_j$ for $j \in [i-1]$ by induction. Hence, for $j = i-1$ we have $y <_\sigma u_{i-1}$. To recap, we are in the case $u_{i-1} <_\pi x <_\pi u_i <_\pi z <_\pi y$ and we have shown that $x <_\sigma y <_\sigma u_i, u_{i-1} <_\sigma z$.

We thus have $y <_\sigma u_{i-1} <_\sigma z$ and $y <_{\overline{\pi}} z <_{\overline{\pi}} u_{i-1}$. Then, in view of Lemma 6.3.4, one must have $A_{yu_{i-1}} = A_{yz}$. From the Robinson ordering we obtain that it holds $A_{yz} \geq A_{xy} \geq A_{yu_{i-1}} = A_{yz}$ and therefore we get the equality $A_{yz} = A_{xy}$. Analogously, because $x <_\sigma y <_\sigma u_i$ and $x <_\pi u_i <_\pi y$, by Lemma 6.3.4 we obtain $A_{xy} = A_{xu_i}$. Hence, we have

$$A_{yu_{i-1}} = A_{yz} = A_{xy} = A_{xu_i} \tag{6.6}$$

Finally, using relation (6.4) we get:

$$A_{u_{i-1}u_i} > \min\{A_{yu_{i-1}}, A_{yu_i}\} = A_{yu_{i-1}}. \tag{6.7}$$

In view of (6.6), the right hand side in (6.7) is $A_{yu_{i-1}} = A_{xu_i}$. On the other hand, as $u_{i-1} <_\pi x <_\pi u_i$ in the Robinson ordering π, then $A_{yu_{i-1}} = A_{xu_i} \geq A_{u_{i-1}u_i}$, which contradicts (6.7). Hence u_i cannot appear before z in π.

(ii) Assume u_i appears after z in π. Then $u_1, \ldots, u_{i-1} <_\pi x <_\pi z <_\pi u_i <_\pi y$. Observe that the path $(x, u_1, \ldots, u_{i-1}, z)$ is a short path from x to z with $u_1, \ldots, u_{i-1} <_\sigma z$ and thus it cannot avoid y, otherwise we would contradict (6.3). Since the path $(x, u_1, \ldots u_{i-1})$ avoids y (as it is a subpath of P), it follows that the path (u_{i-1}, z) does not avoid y. Hence $A_{u_{i-1}z} \leq \min\{A_{yu_{i-1}}, A_{yz}\}$ which, using the Robinson ordering π, in turn implies $A_{u_{i-1}z} = A_{yu_{i-1}}$. Then, using relation (6.4), we get: $A_{u_{i-1}u_i} > \min\{A_{yu_{i-1}}, A_{yu_i}\} = A_{yu_{i-1}}$. Now combining with $A_{yu_{i-1}} = A_{u_{i-1}z}$, we get $A_{u_{i-1}u_i} > A_{u_{i-1}z}$ which is a contradiction, since from the Robinson ordering π one must have $A_{u_{i-1}u_i} \leq A_{u_{i-1}z}$. Therefore we have shown also that u_i cannot appear after z in π.

In summary we have shown that $u_i <_\pi x$ as desired. Finally we show that $y <_\sigma u_i$. Indeed, if $u_i <_\sigma y$ then we would have: $u_i <_\sigma y <_\sigma z$ and $u_i <_\pi z <_\pi y$, while $(u_i, \ldots, u_k, z)$ is a short path from u_i to z avoiding y with $u_{i+1}, \ldots, u_k <_\sigma z$, which contradicts (6.3). This concludes the proof of the claim. □

We can now conclude the proof of Lemma 6.3.6. By Claim 6.3.7 we have the following relations for any $i \in [k]$: $x <_\sigma y <_\sigma u_i <_\sigma z$ and $y <_{\overline{\pi}} z <_{\overline{\pi}} x <_{\overline{\pi}} u_i$. By Lemma 6.3.4, this implies $A_{yu_i} = A_{yz}$ for all $i \in [k]$ which, using the Robinson ordering π, in turn implies $A_{yu_i} = A_{yz} = A_{yx}$. Now, use relation (6.4) for $i = k+1$ to get the inequality $A_{u_kz} > \min\{A_{yu_k}, A_{yz}\} = A_{yu_k}$. Recall that in view of Lemma 6.3.4, we have that $A_{xy} = A_{xz}$. Then as $A_{yu_i} = A_{yx}$ for all i, the right hand side is equal to $A_{yu_k} = A_{xz}$ while, using the Robinson ordering π, the left hand side satisfies $A_{u_kz} \leq A_{xz}$, which yields a contradiction. This concludes the proof of the lemma. □

6.3.4 End points of SFS orderings

In this section we show some fundamental properties of SFS orderings, using the results in Section 6.3.3. First we show that if A is Robinsonian then the last vertex of a SFS ordering of A is an anchor of A. We will see later in Corollary 6.4.3 that conversely any anchor can be obtained as the end point of a SFS ordering.

6.3.8 Theorem. *Let A be a Robinsonian similarity matrix and let $\sigma = \mathrm{SFS}(A)$. Then the last vertex of σ is an anchor of A.*

Proof. Let z be the last vertex of σ; we show that z is an anchor of A. In view of Theorem 6.2.6 it suffices to show that z is valid. Suppose for contradiction that, for some $x \neq y \in V \setminus \{z\}$, there exist a path P from x to z avoiding y and a path Q from y to z avoiding x. We may assume without loss of generality that $x <_\sigma y <_\sigma z$. Moreover, let π be a Robinson ordering of A such that $x <_\pi z$. Then, in view of Lemma 6.2.3, we must have $x <_\pi z <_\pi y$, since y must come either before or after both x and z (because of the path P) and x must come before or after both y and z (because of the path Q). As z is the last vertex, then $P <_\sigma z$ and thus we get a contradiction with Lemma 6.3.6 (PAL). □

The above result is the analog of [36, Cor 3.4] for Lex-BFS applied to interval graphs. We now introduce the concept of 'good SFS'.

6.3.9 Definition. (**Good SFS ordering**) We say that a SFS ordering σ of A is good if σ starts with a vertex which is the end-vertex of some SFS ordering.

Note that the analogous definition in [38] for Lex-BFS (see Subsection 4.2.2) is stronger, as it requires the first vertex of each slice to be an end-vertex of the slice itself. However, in our discussion we do not need such a strong definition and the above notion of good SFS will suffice to show the overall correctness of the multisweep algorithm. In the case when A is Robinsonian, in view of Theorem 6.3.8 (and Corollary 6.4.3 below), σ is a good SFS ordering precisely when it starts with an anchor of A. For good SFS orderings we have the following stronger result for their end points.

6.3.10 Theorem. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix and let σ be a good SFS ordering whose first vertex is a and whose last vertex is b. Then a, b are opposite anchors of A.*

Proof. By assumption, σ is a good SFS ordering and thus its first vertex a is an anchor of A. In view of Theorem 6.3.8, its last vertex b is also an anchor of A. Suppose, for the sake of contradiction, that a and b are not opposite anchors of A. Then, in view of Theorem 6.2.8, there exists a vertex $x \in V$ and a path P from a to b such that P avoids x. Let π be a Robinson ordering of A starting with a (which exists since a is an anchor of A). Using Lemma 6.2.3 applied to the path P, we can conclude that x cannot appear between a and b in any Robinson ordering, and thus we must have $a <_\pi b <_\pi x$. But then, using Lemma 6.3.6 (PAL), there cannot exist a path from a to b avoiding x and appearing before b in σ, which contradicts the existence of P. $\square$

6.4 The SFS$_+$ algorithm

In this section we introduce the SFS$_+$ algorithm. This is a variant of the standard SFS algorithm, and it is the analog of the variant Lex-BFS+ of Lex-BFS introduced by Simon [107] in the study of multisweep algorithms for interval graphs (see Subsection 4.2.2). The SFS$_+$ algorithm takes a given ordering as input, which is then used to break ties. SFS$_+$ will be the main ingredient in our multisweep algorithm for the recognition of Robinsonian matrices. In Subsection 6.4.1 we describe the SFS$_+$ algorithm in detail and we show a basic property, namely that it 'flips anchors' when applied to a Robinsonian matrix A and a good SFS order σ . Then in Subsection 6.4.2 we introduce the 'similarity layers' of a matrix, a strengthened version of BFS layers for unweighted graphs, which are useful for the correctness proof of the multisweep algorithm. We show in particular that the similarity layers enjoy some compatibility with Robinson and SFS$_+$ orderings.

6.4.1 Description

Consider again the SFS algorithm as described in Algorithm 6.1 in Subsection 6.3.1. The first main task is selecting the new pivot. In case of ties, as done at Line 4 of Algorithm 6.1, the ties are broken arbitrarily (choosing any vertex in the slice S). We now introduce a variant of SFS(A), which we denote by SFS$_+(A, \sigma)$ (see Algorithm 6.2). It takes as input a matrix $A \in \mathcal{S}^n$ and a linear order σ of V, and it returns a new linear order σ_+ of V. In the SFS$_+$ algorithm, the input linear order σ is used to break ties at Line 4 in Algorithm 6.1. Specifically, among the vertices in the slice S of the current iteration, we choose the vertex appearing last in σ. Notice that a SFS$_+$ ordering is still a SFS ordering and thus it satisfies all the properties discussed in Subsection 6.3.1.

If A is a Robinsonian matrix and the input linear order σ is a SFS ordering, then the SFS$_+$ ordering σ_+ has some important properties. In fact, since in the beginning of the SFS algorithm all the vertices are contained in the 'universal' slice (i.e., the full ground set V), the order σ_+ starts with the last vertex of σ, which in view of Lemma 6.3.8 is an anchor of A. Therefore, in this case, σ_+ is a good SFS ordering by construction. Furthermore, in view of Theorem 6.3.10, when A is Robinsonian then the first and last vertices of σ_+ are opposite anchors of A.

Algorithm 6.2: $\mathrm{SFS}_+(A,\sigma)$

```
input: a matrix $A \in \mathcal{S}^n$ and a linear order $\sigma$ of $[n]$
output: a linear order $\sigma_+$ of $[n]$

1  $\phi = (V)$
2  for $i = 1, \dots, n$ do
3      $S$ is the first class of $\phi$
4      choose $p$ as the vertex in $S$ appearing last in $\sigma$
5      $\sigma_+(p) = i$
6      remove $p$ from $\phi$
7      $N(p)$ is the set of vertices $y \in \phi$ with $A_{py} > 0$
8      $\psi_p$ is the similarity partition of $N(p)$ with respect to $p$
9      $\phi =$ Refine $(\phi, \psi_p)$
10 return: $\sigma_+$
```

If the input linear order σ is a good SFS ordering, then we have an even stronger property: the end points of σ_+ are the end points of σ but in reversed order. We call this the 'anchors flipping property', which is shown in the next theorem. This property will be crucial in Section 6.5 when studying the properties of the multisweep algorithm.

6.4.1 Theorem. ***(Anchors flipping property)*** *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity, let σ be a good SFS ordering of A and $\sigma_+ = \mathrm{SFS}_+(A,\sigma)$. Suppose that σ starts with a and ends with b. Then σ_+ starts with b and ends with a.*

Proof. By definition of the SFS_+ algorithm, the returned order σ_+ starts with the last vertex b of σ. Hence, we only have to show that a appears last in σ^+. Suppose, for the sake of contradiction, that a is not last in σ^+ and let instead y be the vertex appearing last in σ^+. Then we have $a <_\sigma y <_\sigma b$ and $b <_{\sigma_+} a <_{\sigma_+} y$. This implies that y and a must be split in σ^+. Indeed, if y and a would be tied in σ^+ then, as we use σ to break ties and as $a <_\sigma y$, the vertex y would be placed before a in σ^+, a contradiction. Let thus $x <_{\sigma_+} a$ be the pivot splitting a and y in σ^+, so that $A_{xa} > A_{xy}$. Then we have:

$$A_{xa} > \min\{A_{xy}, A_{ya}\}. \tag{6.8}$$

Hence the path $P = (x, a)$ avoids y. As b is the first vertex of σ^+, we have:

$$b <_{\sigma_+} x <_{\sigma_+} a <_{\sigma_+} y.$$

In view of Theorem 6.3.10 applied to σ, we know that a and b are opposite anchors of A. Therefore, there exists a Robinson ordering π starting with a and ending with b. In view of (6.8) and using Lemma 6.2.3, y cannot appear between a and x in any Robinson ordering and therefore we can conclude:

$$a <_\pi x <_\pi y <_\pi b. \tag{6.9}$$

Consider now σ. We have that $a <_\sigma y <_\sigma b$. Where can x appear in σ? Suppose $y <_\sigma x$. Then we would have $a <_\sigma y <_\sigma x$ and $a <_\pi x <_\pi y$, and in view of Lemma 6.3.6 (PAL) there cannot exist a path from a to x avoiding y and appearing before x in σ, which is a contradiction as the path $P = (x, a)$ avoids y in view of (6.8). Hence, we must have:

$$a <_\sigma x <_\sigma y <_\sigma b. \tag{6.10}$$

Therefore, starting from the pair (a, y) satisfying $a <_\sigma y$ and $a <_{\sigma_+} y$, we have constructed a new pair (x, y) satisfying $x <_\sigma y$ and $x <_{\sigma_+} y$, with $x <_{\sigma_+} a$. Iterating this construction we are going to get an infinite sequence of such pairs, yielding a contradiction. □

The flipping property of anchors is the analog of [38, Thm 4.6] for Lex-BFS (see Theorem 4.3.4). An important consequence of this property is that, if the linear order σ given as input is a Robinson ordering of A, then $\sigma_+ = \mathrm{SFS}_+(A, \sigma)$ is equal to $\overline{\sigma}$, the reversed order of σ.

6.4.2 Lemma. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix and let σ, τ be two SFS orderings of A. The following holds:*

(i) If $x <_\tau y <_\tau z$ and $z <_\sigma y <_\sigma x$ then the triple (x, y, z) is Robinson.

(ii) If τ is a Robinson ordering of A and $\sigma = \mathrm{SFS}_+(A, \tau)$, then $\sigma = \overline{\tau}$.

Proof. (i) Suppose for contradiction that the triple (x, y, z) is not Robinson. Then we have $A_{xz} > \min\{A_{xy}, A_{yz}\}$, and thus the path (x, z) avoids y. Let π be a Robinson ordering of A with (say) $x <_\pi y$. In view of Lemma 6.2.3, y cannot appear between x and z in any Robinson ordering and therefore we have $x <_\pi z <_\pi y$ or $z <_\pi x <_\pi y$. In both cases we get a contradiction with Lemma 6.3.6 (PAL) since $x <_\tau y <_\tau z$ and $z <_\sigma y <_\sigma x$.

(ii) Say τ starts at b and ends at a. Then σ starts at a. Assume that $\sigma \neq \overline{\pi}$. Let $(a = x_0, x_1, \dots, x_k)$ be the longest initial segment of σ whose reverse $(x_k, \dots, x_1, a)$ is the final segment of τ, with $k \geq 0$. Let y be the successor of x_k in σ. Then y is not the predecessor of x_k in τ (by maximality of k). Let z be the predecessor of x_k in τ. Then $a <_\sigma x_1 <_\sigma \dots <_\sigma x_k <_\sigma y <_\sigma z$ and $y <_\tau z <_\tau x_k <_\tau \dots <_\tau x_1 <_\tau a$. Hence, y, z cannot be tied in σ (otherwise we would choose z before y in σ as $y <_\tau z$). Therefore, there must exist a vertex $x <_\sigma y$ such that $A_{xy} > A_{xz}$. Hence, $x = x_i$ for some $0 \leq i \leq k$ and thus $y <_\tau z <_\tau x$. As τ is a Robinson ordering, then $A_{xy} \leq A_{xz}$, a contradiction. □

In other words, if the multisweep algorithm is applied to a Robinsonian matrix, every triple of vertices appearing in reversed order in two distinct sweeps is Robinson and, once a given sweep is a Robinson ordering, the next sweep will remain a Robinson ordering (precisely the reversed order). As an important application, if A is Robinsonian one can check if a given order σ is Robinson simply by computing the order $\sigma_+ = \text{SFS}_+(A, \sigma)$, and checking if it is the reversed of σ, hence without checking the Robinson property for all the entries of the matrix. This is analogous to a similar feature shown in [51] for their multisweep algorithm to recognize cocomparability graphs.

As a direct application of Lemma 6.4.2 combined with Theorem 6.3.8, we obtain the following characterization for anchors.

6.4.3 Corollary. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix. A vertex is an anchor of A if and only if it is the end point of a SFS ordering of A.*

6.4.2 Similarity layers

In this section we introduce the notion of 'similarity layer structure' for a matrix $A \in \mathcal{S}^n$ and an element $a \in V$ (then called the *root*), which we will use later to analyze properties of the multisweep algorithm.

Specifically, we define the following collection $\mathcal{L} = (L_0, L_1, \ldots, L_r)$ of subsets of V, whose members are called the *(similarity) layers of A rooted at a*, where $L_0 = \{a\}$ and the next layers L_i are the subsets of V defined recursively as follows:

$$L_i = \{y \notin L_0 \cup \cdots \cup L_{i-1} : A_{xy} \geq A_{xz} \ \forall x \in L_0 \cup \cdots \cup L_{i-1},\ \forall z \notin L_0 \cup \cdots \cup L_{i-1}\}. \tag{6.11}$$

Note that this notion of similarity layers can be seen as a refinement of the notion of BFS layers for graphs, which are obtained by layering the nodes according to their distance to the root (see Subsection 4.3.2). Hence, the two concepts are similar but different. For a concrete example on how the similarity layers look like, we refer the reader to the example in Subsection 6.5.5. We first show that this layer structure defines a partition of V when A is a Robinsonian matrix and the root a is an anchor of A.

6.4.4 Lemma. *Assume that $A \in \mathcal{S}^n$ is a Robinsonian matrix and that $a \in V$ is an anchor of A. Consider the similarity layer structure $\mathcal{L} = (L_0 = \{a\}, L_1, \ldots, L_r)$ of A rooted at a, as defined in (6.11), where r is the smallest index such that $L_{r+1} = \emptyset$. The following holds:*

(i) If $y \in L_i, z \notin L_0 \cup \ldots \cup L_i$ with $i \geq 1$, then there exists a path P from a to y avoiding z. Moreover, any path of the form $P = (a, a_1, \ldots, a_i = y)$, where $a_l \in L_l$ for $1 \leq l \leq i$, avoids z.

(ii) $V = L_0 \cup L_1 \cup \ldots \cup L_r$.

Proof. (i) Using the definition of the layers in (6.11) we obtain that $A_{aa_1} > A_{az}$ and $A_{a_1a_2} > A_{a_1z}$, $\dots, A_{a_{i-1}y} > A_{a_{i-1}z}$, which shows that the following path $(a, a_1, \dots, a_{i-1}, a_i = y)$ avoids z.

(ii) Suppose $L_0, L_1, \dots, L_r \neq \emptyset$, $L_{r+1} = \emptyset$, but $V \neq U := L_0 \cup \ldots \cup L_r$. Consider an element $z_1 \in V \setminus U$. As $z_1 \notin L_r$ (since this set is empty) there exist elements $x_1 \in U$ and $z_2 \notin U$ such that $A_{x_1z_1} < A_{x_1z_2}$. Analogously, as $z_2 \notin L_r$ there exist elements $x_2 \in U, z_3 \notin U$ such that $A_{x_2z_2} < A_{x_2z_3}$. Iterating we find elements $x_i \in U$, $z_i \notin U$ for all $i \geq 1$ such that $A_{x_iz_i} < A_{x_iz_{i+1}}$ for all i. At some step one must find one of the previously selected elements z_i, i.e., $z_j = z_i$ for some $i < j$.

As a is an anchor of A, there exists a Robinson ordering π of A starting at a. We first claim that $x_i <_\pi z_j$ for all i, j. This is clear if $x_i = a$. Otherwise, as $x_i \in U$ and $z_j \notin U$, it follows from (i) that there is a path from a to x_i avoiding z_j, which in view of Lemma 6.2.3 implies that $a \leq_\pi x_i <_\pi z_j$. Next we claim that $z_{i+1} <_\pi z_i$. Since $x_i <_\pi z_i$ and $A_{x_iz_i} < A_{x_iz_{i+1}}$, then (x_i, z_{i+i}) avoids z_i and in view of Lemma 6.2.3 it must be indeed $z_{i+1} <_\pi z_i$. Summarizing we have shown that $z_{i+1} <_\pi z_i <_\pi \ldots <_\pi z_1$ for all i, which contradicts the fact that two of the z_i's should coincide. □

Intuitively, each layer L_i will correspond to some slice of a SFS algorithm starting at a. As we see below, there is some compatibility between the layer structure $\mathcal{L}$ rooted at a with any Robinson ordering π and any good SFS ordering σ starting at a.

6.4.5 Lemma. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix and a is an anchor of A. Let σ be a good SFS ordering of A starting at a and let π be a Robinson ordering of A starting at a. Then the similarity layer structure $\mathcal{L} = (L_0 = \{a\}, \dots, L_r)$ of A rooted at a is compatible with both π and σ, i.e.,*

$$L_0 <_\pi L_1 <_\pi \ldots <_\pi L_r,$$
$$L_0 <_\sigma L_1 <_\sigma \ldots <_\sigma L_r.$$

Proof. Let $x \in L_i$ and $y \in L_j$ with $i < j$; we show that $x <_\pi y$ and $x <_\sigma y$. This is clear if $i = 0$, i.e., if $x = a$. Suppose now $i \geq 1$. Then, by Lemma 6.4.4, there exists a path from a to x avoiding y. This implies that $a <_\pi x <_\pi y$, as y cannot appear between a and x in any Robinson ordering in view of Lemma 6.2.3 and since π starts with a. Furthermore, if $a <_\sigma y <_\sigma x$ then we would get a contradiction with Lemma 6.3.6 (PAL). Hence $a <_\sigma x <_\sigma y$ holds, as desired. □

Furthermore, the following inequalities hold among the entries of A indexed by elements in different layers.

6.4.6 Lemma. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix and a is an anchor of A. Let $\mathcal{L} = (L_0 = \{a\}, L_1, \dots, L_r)$ be the similarity layer structure of A rooted at a. For each $u \in L_i, x, y \in L_j$ and $z \notin L_0 \cup L_1 \cup \ldots \cup L_j$ with $0 \leq i < j$ the following inequalities hold:*

$$A_{xy} \geq A_{ux} = A_{uy} \geq A_{uz}.$$

Furthermore, if $x \in L_j, z \notin L_0 \cup L_1 \cup \cdots \cup L_j$, then there exists $u \in L_0 \cup L_1 \cup \cdots \cup L_{j-1}$ such that $A_{ux} > A_{uz}$.

Proof. The inequalities $A_{ux} = A_{uy} > A_{uz}$ follow from the definition of the layers in (6.11). Suppose now that $A_{xy} < A_{ux} = A_{uy}$. Then u must appear between x and y in any Robinson ordering π, since $x <_\pi y <_\pi u$ implies $A_{xy} \geq A_{ux}$ and $y <_\pi x <_\pi u$ implies $A_{xy} \geq A_{uy}$. But in view of Lemma 6.4.5, if π is a Robinson ordering starting at a then $u <_\pi x$ and $u <_\pi y$, so we get a contradiction. $\square$

As an application of Lemma 6.4.6, it is easy to verify that if A is the adjacency matrix of a connected graph G, then each layer is a clique of G.

We now show a 'flipping property' of the similarity layers with respect to a good SFS ordering σ starting at the root and the next sweep $\sigma_+ = \text{SFS}_+(A, \sigma)$. Namely we show that the orders of the layers are reversed beween σ and σ_+, i.e., $L_i <_\sigma L_j$ and $L_j <_{\sigma_+} L_i$ for all $i < j$.

6.4.7 Theorem. ***(Layers flipping property)*** *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix and $a \in V$ be an anchor of A. Let $\mathcal{L} = (L_0 = \{a\}, \dots, L_r)$ be the similarity layer structure of A rooted at a, let σ be a good SFS ordering of A starting at a and let $\sigma^+ = \text{SFS}_+(A, \sigma)$. If $x \in L_i$, $y \in L_j$ with $0 \leq i < j \leq r$ then $y <_{\sigma_+} x$.*

Proof. Let $x \in L_i$, $y \in L_j$ with $i < j$. Assume for contradiction that $x <_{\sigma_+} y$. By Lemma 6.4.5, we know that $\mathcal{L}$ is compatible with σ and thus $x <_\sigma y$. As $x <_{\sigma_+} y$ and $x <_\sigma y$, we deduce that x, y are not tied in σ^+. Hence there exists $x_1 <_{\sigma_+} x$ such that $A_{x_1x} > A_{x_1y}$. Let L_ℓ denote the layer of $\mathcal{L}$ containing x_1. We claim that $\ell < j$. Indeed, if $\ell = j$ then x_1, y are in the same layer and, by Lemma 6.4.6, it must be $A_{x_1y} \geq A_{x_1x} = A_{xy}$ which is impossible, because $A_{x_1x} > A_{x_1y}$. Assume now that $\ell > j$. By Lemma 6.4.5, if π is a Robinson ordering starting at a, then we would get $x <_\pi y <_\pi x_1$, which implies $A_{x_1y} \geq A_{x_1x}$, again a contradiction. Therefore, we have $x_1 \in L_\ell$ with $\ell < j$. Recall that $x_1 <_{\sigma_+} y$. Hence, starting with the pair (x, y) which satisfies $x \in L_i$, $y \in L_j$ with $i < j$ and $x <_{\sigma_+} y$, we have constructed another pair (x_1, y) satisfying $x_1 \in L_l$, $y \in L_j$ with $l < j$ and $x_1 <_{\sigma_+} y$. As $x_1 <_+ x$, iterating this construction we reach a contradiction. $\square$

6.5 The multisweep algorithm

We now introduce our new SFS-based multisweep algorithm to recognize Robinsonian matrices and we prove its correctness. In Subsection 6.5.1 we describe the multisweep algorithm and show that it terminates in 3 sweeps when applied to a 0/1 matrix, thus giving a new proof of the result of Corneil [33] for unit interval graphs (see Subsection 4.3.2). In Subsection 6.5.2 we study properties of '3-good SFS orderings', which are orderings obtained after three SFS_+ sweeps. In particular we show that they contain classes of Robinson triples and that, after deleting their end points, they induce good SFS orderings, which will enable us to apply induction in the correctness proof. After that we have all the ingredients needed to conclude the correctness proof for the multisweep algorithm in Subsection 6.5.3. Then, in Subsection 6.5.4 we discuss the complexity of the SFS algorithm. Finally, in Subsection 6.5.5 we give an example to show how the algorithm works concretely.

6.5.1 Description

As the multisweep algorithm framework presented in Chapter 4 (Algorithm 4.4), our multisweep algorithm consists of computing successive SFS orderings of a given nonnegative matrix $A \in \mathcal{S}^n$. The first sweep is SFS(A), whose aim is to find an anchor of A. Each subsequent sweep is computed with the SFS_+ algorithm using the linear order returned by the preceding sweep to break ties (as in Algorithm 6.2). As it starts with the end point of the preceding sweep which is an anchor of A, each subsequent sweep is therefore a good SFS ordering of A (in the case when A is Robinsonian). The algorithm terminates either if a Robinson ordering has been found (in which case it certifies that A is Robinsonian), or if the $(n-1)$th sweep is not Robinson (in which case it certifies that A is not Robinsonian). The complete algorithm is given below.

Algorithm 6.3: *Robinson*(A)

```
input: a matrix $A \in \mathcal{S}^n$
output: a Robinson ordering $\pi$ of $A$, or stating that $A$ is not Robinsonian

1 $\sigma_0 = \mathrm{SFS}(A)$
2 for $i = 1, \dots n-2$ do
3     $\sigma_i = \mathrm{SFS}_+(A, \sigma_{i-1})$
4     if $\sigma_i$ is Robinson then
5         return: $\pi = \sigma_i$
6 return: 'A is NOT Robinsonian'
```

It is an open question whether there exists a family of instances for which exactly $n-1$ sweeps are needed to find a Robinson ordering. Nevertheless, we give below some examples for $n = 4, 5, 6$ where this happens. Note that for $n = 3$ it is clear that two sweeps may be needed and always suffice.

6.5.1 Example. Consider the following (Robinson) matrices:

$$A_4 = \begin{array}{c|cccc} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ \hline \mathbf{a} & * & 1 & 1 & 0 \\ \mathbf{b} & & * & 2 & 1 \\ \mathbf{c} & & & * & 2 \\ \mathbf{d} & & & & * \end{array}, A_5 = \begin{array}{c|ccccc} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ \hline \mathbf{a} & * & 2 & 2 & 0 & 0 \\ \mathbf{b} & & * & 2 & 1 & 1 \\ \mathbf{c} & & & * & 2 & 1 \\ \mathbf{d} & & & & * & 1 \\ \mathbf{e} & & & & & * \end{array}, A_6 = \begin{array}{c|cccccc} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} & \mathbf{f} \\ \hline \mathbf{a} & * & 1 & 1 & 1 & 1 & 0 \\ \mathbf{b} & & * & 2 & 2 & 1 & 1 \\ \mathbf{c} & & & * & 2 & 2 & 2 \\ \mathbf{d} & & & & * & 3 & 2 \\ \mathbf{e} & & & & & * & 2 \\ \mathbf{f} & & & & & & * \end{array}.$$

Consider first the matrix A_4. Then $\sigma_0 = (b, c, d, a)$ is a valid SFS ordering of A_4 and $\sigma_1 = (a, c, b, d)$, which is not a Robinson ordering. However, $\sigma_2 = (d, c, b, a)$ is a Robinson ordering of A_4.

Consider now the matrix A_5. Then $\sigma_0 = (c, d, b, a, e)$ is a valid SFS ordering of A_5, with $\sigma_1 = (e, b, c, d, a)$ and $\sigma_2 = (a, c, b, d, e)$, which are not Robinson orderings. However, $\sigma_3 = (e, d, c, b, a)$ is a Robinson ordering of A_5.

Consider finally the matrix A_6. Then $\sigma_0 = (b, d, c, e, f, a)$ is a valid SFS ordering of A_6, with $\sigma_1 = (a, e, d, c, b, f)$, $\sigma_2 = (f, c, d, e, b, a)$ and $\sigma_3 = (a, b, d, c, e, f)$ which are not Robinson orderings. However, $\sigma_4 = (f, e, d, c, b, a)$ is a Robinson ordering of A_6.

As already mentioned earlier, the SFS algorithm applied to 0/1 matrices reduces to Lex-BFS. As a warm-up we now show that our SFS multisweep algorithm terminates in three sweeps to recognize whether a 0/1 matrix A is Robinsonian. Since a 0/1 matrix A is Robinsonian if and only if the corresponding graph is a unit interval graph [97] (see Theorem 3.2.1), this is coherent with the fact that one can recognize unit interval graphs in three sweeps of Lex-BFS [33, Thm 9] (see Subsection 4.3.2). Hence we have a new proof for this result, which has similarities but yet differs from the original proof in [33].

6.5.2 Theorem. *Let G be a connected graph and let A be its (extended) adjacency matrix. Consider the orders $\sigma_0 = \mathrm{SFS}(A)$, $\sigma_1 = \mathrm{SFS}_+(A, \sigma_0)$ and $\sigma_2 = \mathrm{SFS}_+(A, \sigma_1)$. Then G is a unit interval graph (i.e., A is a Robinsonian similarity) if and only if σ_2 is a Robinson ordering of A.*

Proof. Clearly, if σ_2 is Robinson then A is Robinsonian. Assume now that A is Robinsonian; we show that σ_2 is Robinson. Suppose, for contradiction, that there exists a triple $x <_{\sigma_2} y <_{\sigma_2} z$ which is not Robinson, i.e., $A_{xz} > \min\{A_{xy}, A_{yz}\}$. Then the path (x, z) avoids y and thus, in view of Lemma 6.3.6 (PAL), in any Robinson ordering π one cannot have $x <_\pi z <_\pi y$. We may assume without loss of generality that $z <_\pi x <_\pi y$ in some Robinson ordering π. Because A is a 0/1 matrix, then $A_{xz} = 1$, $A_{yz} = 0$ and thus $\{x, z\} \in E, \{y, z\} \notin E$. By construction, σ_1 is a good SFS ordering of A starting (say) at the anchor a. Let $\mathcal{L} = \{L_0, L_1 \dots, L_r\}$ be the similarity layer structure of A rooted at a. By Lemma 6.4.5, we know that $\mathcal{L}$ is compatible with σ_1, i.e., $a <_{\sigma_1} L_1 <_{\sigma_1} \dots <_{\sigma_1} L_r$. Using Theorem 6.4.7 we obtain that $L_r <_{\sigma_2} L_{r-1} <_{\sigma_2} \dots <_{\sigma_2} L_1 <_{\sigma_2} a$. Moreover, using Lemma 6.4.6 and the fact that G is connected, it is easy to see that each layer L_i is a clique of G. Hence, y, z cannot be in the same layer of $\mathcal{L}$, as $\{y, z\} \notin E$. Since $y <_{\sigma_2} z$, it follows that $z \in L_i, y \in L_j$ with $i < j$ and thus $z <_{\sigma_1} y$. Say $x \in L_h$. One cannot have $h < j$ since this would contradict $x <_{\sigma_2} y$. If $h = j$ then $x, y \in L_j$ and thus $A_{zx} = A_{zy}$ by definition of the layers, contradicting the fact that $A_{xz} = 1$, $A_{yz} = 0$. Hence one must have $j < h$. Then $z \in L_i$, $y \in L_j$, $x \in L_h$ with $i < j < h$ and thus $z <_{\sigma_1} y <_{\sigma_1} x$. Now we get a contradiction with Lemma 6.3.6 (PAL), as $z <_\pi x <_\pi y$ and the path (x, z) avoids y. □

The proof of Theorem 6.5.2 outlines a fundamental difference between unit interval graphs and Robinsonian matrices. In fact, in view of Lemma 6.4.6, it is easy to see that, for 0/1 Robinsonian matrices, each layer L_i of the similarity layer structure $\mathcal{L}$ rooted at an anchor a is a clique of G. This property in fact permits to bound by a constant factor the number of sweeps to recognize 0/1 Robinsonian matrices. However, if A is a Robinsonian matrix with at least three distinct values, then we loose such a strong property of the layers, which explains why we might need $n - 1$ sweeps in the worst case.

We now formulate our main result, namely that the SFS multisweep algorithm recognizes in at most $n - 1$ steps whether an $n \times n$ matrix is Robinsonian.

6.5.3 Theorem. *Let $A \in \mathcal{S}^n$ and let $\sigma_0 = \mathrm{SFS}(A)$, $\sigma_i = \mathrm{SFS}_+(A, \sigma_{i-1})$ for $i \geq 1$ be the successive sweeps returned by Algorithm 6.3. Then A is a Robinsonian similarity matrix if and only if σ_{n-2} is a Robinson ordering of A.*

We will give the full proof of Theorem 6.5.3 in Subsection 6.5.3 below. What we need to show is that if A is Robinsonian then the order σ_{n-2} in Algorithm 6.3 is a Robinson ordering of A. We now give a rough sketch of the strategy which we will use to prove this result. The proof will use induction on the size n of the matrix A.

As was shown earlier, the sweep σ_1 is a good SFS ordering of A with end points (say) a and b, and all subsequent sweeps have the same end points (flipping their order at each sweep) in view of Theorem 6.4.1. A first key ingredient will be to show that if we delete both end points a and b and set $S = V \setminus \{a, b\}$, then the induced order $\sigma_3[S]$ is a good SFS ordering of the principal submatrix $A[S]$. A second crucial ingredient will be to show that the induced order $\sigma_{n-2}[S]$ can be obtained with the multisweep algorithm applied to $A[S]$ starting from $\sigma_3[S]$. This will enable us to apply the induction assumption and to conclude that $\sigma_{n-2}[S]$ is a Robinson ordering of $A[S]$. Hence all triples (x, y, z) in σ_{n-2} that are contained in $S = V \setminus \{a, b\}$ are Robinson. The last step is to show that all triples (x, y, z) in σ_{n-2} that contain a or b are also Robinson.

As we see in the above sketch, the sweep σ_3 plays a special role. It is obtained by applying three sweeps of SFS_+ starting from the good SFS ordering σ_1. For this reason we call it a *3-good SFS ordering.* We introduce and investigate in detail this notion of '3-good SFS ordering' in Subsection 6.5.2 below.

6.5.2 3-good SFS orderings

Consider a Robinsonian matrix $A \in \mathcal{S}^n$. Recall that a SFS ordering τ of A is said to be *good* if its first vertex is an anchor of A (see Section 6.3.4). We now introduce the notion of 3-good SFS ordering. A linear order τ is called a *3-good SFS ordering* of A if there exists a good SFS ordering τ' of A such that, if we set $\tau'' = \text{SFS}_+(A, \tau')$, then $\tau = \text{SFS}_+(A, \tau'')$ holds. In other words, a 3-good SFS ordering is obtained by performing three consecutive good sweeps. Of course any 3-good SFS ordering is also a good SFS ordering. Furthermore, if we consider Algorithm 6.3, then any sweep σ_i with $i \geq 3$ is a 3-good SFS ordering by construction. First we present the following flipping property of layers which follows as a direct application of Theorem 6.4.7.

6.5.4 Corollary. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix. Let τ' be a good SFS ordering of A, $\tau'' = \text{SFS}_+(A, \tau')$ and $\tau = \text{SFS}_+(A, \tau'')$. Let $\mathcal{L} = \{L_0, \dots, L_r\}$ be the similarity layer structure of A rooted at the first vertex of τ. If $x \in L_i$, $y \in L_j$ with $i < j$ then $y <_{\tau''} x$.*

We now show some important properties of 3-good SFS orderings, that we will use in the proof of correctness of the multisweep algorithm. First we show that some triples in a 3-good SFS ordering can be shown to be Robinson.

6.5.5 Lemma. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix. Let τ be a 3-good SFS ordering starting at a and ending at b. Let $\mathcal{L} = \{L_0 = \{a\}, L_1, \dots, L_r\}$ be the similarity layer structure of A rooted at a. Then the following holds:*

(i) If $x <_\tau y <_\tau z$ and (x, y, z) is not Robinson, then $x, y, z \in L_i$ with $1 \leq i \leq r$.

(ii) Every triple (a, x, y) with $x <_\tau y$ is Robinson.

(iii) Every triple (x, y, b) with $x <_\tau y$ is Robinson.

Proof. Let τ' be a good SFS order such that $\tau'' = \text{SFS}_+(A, \tau')$, $\tau = \text{SFS}_+(A, \tau'')$. Let $\mathcal{L}'' = \{L''_0 = \{b\}, L''_1, \dots\}$ denote the similarity layer structure of A rooted at b, which is compatible with τ''.

(i) Let $x <_\tau y <_\tau z$ such that x, y, z do not all belong to the same layer of $\mathcal{L}$ and assume that (x, y, z) is not Robinson. Then $A_{xz} > \min\{A_{xy}, A_{yz}\}$ and the path (x, z) avoids y. Let π be a Robinson ordering and let $\overline{\pi}$ its reversal. Assume, without loss of generality, that $x <_\pi z$. Then, since (x, z) avoids y, in view of Lemma 6.2.3 y cannot appear between x and z in any Robinson ordering. If y appears after z in π then we have $x <_\pi z <_\pi y$ and $x <_\tau y <_\tau z$, and we get a contradiction with Lemma 6.3.6 (PAL) as there cannot exists a path from x to z avoiding y. Therefore $y <_\pi x <_\pi z$ and thus $A_{xz} > \min\{A_{xy}, A_{yz}\} = A_{yz}$. In view of Lemma 6.4.5, x, y, z do not belong to three distinct layers of $\mathcal{L}$ (since otherwise (x, y, z) would be Robinson). Moreover, one cannot have $x \in L_i$ and $y, z \in L_j$ with $i < j$ (since this would imply $A_{xy} = A_{xz} \leq A_{yz}$, a contradiction). Hence we must have $x, y \in L_i$ and $z \in L_j$ with $i < j$.

Consider now τ''; applying Corollary 6.5.4, we derive that $z <_{\tau''} x, y$. Moreover, we cannot have that $z <_{\tau''} y <_{\tau''} x$, since we would get a contradiction with Lemma 6.3.6 (PAL) as $z <_{\overline{\pi}} x <_{\overline{\pi}} y$ and the path (x, z) avoids y. Hence we have $z <_{\tau''} x <_{\tau''} y$. Summarizing, the triple (x, y, z) satisfies the properties:

$$x, y \in L_i, \quad z \in L_j, \quad x <_\tau y <_\tau z, \quad x <_{\tau''} y, \quad y <_\pi x <_\pi z. \tag{6.12}$$

We will now show that the properties in (6.12) (together with the inequality $A_{xz} > A_{yz}$) permit to find an element $x_1 <_\tau x$ for which the triple (x_1, y, z) again satisfies the properties of (6.12), replacing x by x_1. Then, iterating this construction leads to a contradiction.

We now proceed to show the existence of such an element x_1. As $x <_{\tau''} y$ and $x <_\tau y$, x, y are not tied in τ and thus there exists $x_1 <_\tau x$ such that

$$A_{x_1x} > A_{x_1y}.$$

This implies $x_1 \in L_i$ (recall Lemma 6.4.6). Moreover, the path (x_1, x, z) avoids y, since $A_{x_1x} > A_{x_1y}$ and $A_{xz} > A_{yz}$. By construction we have: $x_1 <_\tau x <_\tau y <_\tau z$. We claim that

$$y <_\pi x_1 <_\pi z.$$

Indeed, if $x_1 <_\pi y$, then $x_1 <_\pi y <_\pi x$ and thus $A_{x_1x} \leq A_{x_1y}$, a contradiction. Moreover, if $z <_\pi x_1$ then $y <_\pi x <_\pi z <_\pi x_1$, which implies $A_{x_1z} \geq A_{x_1x} > A_{x_1y}$ and thus the triple (x_1, y, z) is not Robinson. Then $A_{x_1z} > \min\{A_{x_1y}, A_{yz}\}$ and the path (x_1, z) avoids y. Now, as $x_1 <_\tau y <_\tau z$ and $x_1 <_{\overline{\pi}} z <_{\overline{\pi}} y$, we get a contradiction with Lemma 6.3.6 (PAL). So we have shown that $y <_\pi x_1 <_\pi z$.

Next we claim that $x_1 <_{\tau''} y$. Indeed, if $y <_{\tau''} x_1$ then $z <_{\tau''} y <_{\tau''} x_1$, which together with $z <_{\overline{\pi}} x_1 <_{\overline{\pi}} y$ and the fact that the path (x_1, x, z) avoids y, contradicts Lemma 6.3.6 (PAL). Hence we have shown that the triple (x_1, y, z) satisfies (6.12), which concludes the proof of *(i)*.

(ii) follows directly from *(i)*, since any triple containing a is not contained in a unique layer, and thus it must be Robinson.

(iii) Assume for contradiction that (x, y, b) is not Robinson for some $x <_\tau y$, i.e., $A_{bx} > \min\{A_{by}, A_{xy}\}$. Then the path (b, x) avoids y. If π is a Robinson ordering ending at b (which exists since b is an anchor) then we must have $y <_\pi x <_\pi b$ because, in view of Lemma 6.2.3, y cannot appear between x and b in any Robinson ordering. Hence, $A_{bx} > A_{by}$. Since τ'' is compatible with $\mathcal{L}''$ which is rooted at b, we must have $b <_{\tau''} x <_{\tau''} y$ and moreover x, y belong to distinct layers of $\mathcal{L}''$. Thus $x \in L''_i, y \in L''_j$ with $i < j$ which, in view of Theorem 6.4.7, implies $y <_\tau x$, a contradiction. □

As a first direct application of Lemma 6.5.5(i), we can conclude that the multisweep algorithm terminates in at most four steps when applied to a matrix A whose similarity layers rooted at the end-vertex of the first sweep σ_0 all have cardinality at most 2.

Consider a 3-good SFS ordering τ of a Robinsonian matrix A with end points a and b and consider the induced order $\tau[S]$ of the submatrix $A[S]$ indexed by the subset $S = V \setminus \{a, b\}$. In the next lemmas we show some properties of $\tau[S]$. First, we show that $\tau[S]$ is a good SFS ordering of $A[S]$ (Lemma 6.5.7). Second, we show that applying the SFS$_+$ algorithm to τ and then deleting a and b yields the same order as applying the SFS$_+$ algorithm to the induced order $\tau[S]$ (Lemma 6.5.8). These properties will be used in the induction step for the proof of correctness of the multisweep algorithm in the next section. We start with showing a 'flipping property' of the second smallest element of τ.

6.5.6 Lemma. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix. Let τ' be a good SFS ordering of A, $\tau'' = \mathrm{SFS}_+(A, \tau')$ and $\tau = \mathrm{SFS}_+(A, \tau'')$. Let a be the first vertex of τ. Then the successor a_1 of a in τ is the predecessor of a in τ''.*

Proof. As before, $\mathcal{L} = \{L_0 = \{a\}, L_1, \ldots, L_r\}$ is the layer structure of A rooted at a, which is compatible with τ. The slice of a in τ is precisely the first layer L_1 in $\mathcal{L}$. By definition, a_1 is the element of L_1 coming last in τ''. By the flipping property in Corollary 6.5.4, we know that the layer L_1 comes last but one in τ'', just before the layer $L_0 = \{a\}$. Then a_1 is the element of L_1 appearing last in τ'', and thus it coincides with the predecessor of a in τ''. □

6.5.7 Lemma. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix. Let τ be a 3-good SFS ordering of A with end points a and b and set $S = V \setminus \{a, b\}$. Then $\tau[S]$ is a good SFS ordering of $A[S]$.*

Proof. Say that a is the first element of τ and that b is its last element. Consider the similarity layer structure rooted at a, i.e., $\mathcal{L} = (L_0, L_1, \ldots, L_r)$, which is compatible with τ. First we show that $\tau[S]$ is a SFS ordering of $A[S]$. For this consider elements $x, y, z \in S$ such that $A_{xz} > A_{xy}$. Then (x, y, z) is not Robinson and thus $x, y, z \in L_i$ with $i \geq 1$ in view of Lemma 6.5.5. As τ is a SFS ordering, then in view of Theorem 6.3.3 there exists $u <_\tau x$ such that $A_{uy} > A_{uz}$. We have $u \neq a$ (since $u = a$ would imply $A_{uy} = A_{uz}$) and thus $u \in S$. This shows that $\tau[S]$ is a SFS ordering of $A[S]$. Finally $\tau[S]$ is good since, in view of Lemma 6.5.6, it starts at a_1, the successor of a in τ, which is an anchor of $A[V \setminus \{a\}]$ (and thus also of $A[S]$) in view of Theorem 6.3.8. $\square$

6.5.8 Lemma. *Assume $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix. Let τ be a 3-good SFS ordering with end points a and b and $\tau^+ = \mathrm{SFS}_+(A, \tau)$. Then $\tau^+[S] = \mathrm{SFS}_+(A[S], \tau[S])$ with $S = V \setminus \{a, b\}$.*

Proof. Say b is the first element of τ and a be its last element. Then a is the first element of τ^+ and b is its last element (Theorem 6.4.1). Let consider the similarity layer structure $\mathcal{L} = (L_0 = \{a\}, L_1, \ldots, L_r)$ of A rooted at a, which is compatible with τ^+ (and thus we denote here by $\mathcal{L}^+$).

Set $\sigma = \mathrm{SFS}_+(A[S], \tau[S])$. Let a_1 the predecessor of a in τ. As τ^+ is clearly also a 3-good SFS ordering then, in view of Lemma 6.5.6, a_1 is the successor of a in τ^+ and thus both $\tau^+[S]$ and σ start at a_1. Assume that σ and $\tau^+[S]$ agree on their first p elements $a_1, \ldots, a_p$, but not at the next $(p + 1)$th element. That is, $\tau^+[S] = (a_1, \ldots, a_p, x, \ldots, y, \ldots)$, while $\sigma = (a_1, \ldots, a_p, y, \ldots, x, \ldots)$, where x, y are distinct elements. We distinguish three cases.

Assume first that x, y are tied in τ^+ (and thus in σ too). Then one must have $y <_\tau x$ (to place x before y in $\tau^+[S]$) and $x <_\tau y$ (to place y before x in σ), a contradiction.

Assume now that x, y are not tied in τ^+, but they are tied in σ. Then we have $A_{ax} > A_{ay}$. Hence, since the similarity layer structure $\mathcal{L}$ of A is rooted at a, then we have $x \in L_j$, $y \in L_k$ for some $j < k$. This implies $y <_\tau x$ (by Corollary 6.5.4) and thus, since x, y are tied in σ, one would place x before y in σ, a contradiction.

Assume finally that x, y are not tied in τ^+ and also not in σ. Let a_j be the pivot splitting x and y in σ so that $A_{a_j y} > A_{a_j x}$, with $1 \leq j \leq p$. We claim that a is the pivot splitting x and y in $\tau^+[S]$. For this, suppose that a_i is the pivot splitting x and y in $\tau^+[S]$ for some $1 \leq i \leq p$, so that $A_{a_i x} > A_{a_i y}$ and $i \neq j$. It is now easy to see that $i > j$ would imply $y <_{\tau^+} x$, while $i < j$ would imply $x <_\sigma y$, a contradiction in both cases. Hence, a is the pivot splitting x, y in $\tau_+[S]$ and thus $A_{ax} > A_{ay}$. Then, as $\mathcal{L}^+$ is the similarity layer structure of A rooted at a, x and y belong to distinct layers of $\mathcal{L}^+$. Moreover, $a_j <_{\tau_+} x <_{\tau_+} y$ and the triple (a_j, x, y) is not Robinson. As τ^+ is a 3-good SFS ordering, we can apply Lemma 6.5.5 and conclude that a_j, x, y must belong to a common layer of $\mathcal{L}$, which contradicts the fact that x, y belong to distinct layers of $\mathcal{L}^+$. $\square$

6.5.3 Proof of correctness

We can finally put all ingredients together and show the correctness of our multisweep algorithm. We show the following result, which implies directly Theorem 6.5.3.

6.5.9 Theorem. *Let $A \in \mathcal{S}^n$ be a Robinsonian similarity matrix, let τ_1 be a good SFS ordering of A and let $\tau_i = \mathrm{SFS}_+(A, \tau_{i-1})$ for $i \geq 2$. Then τ_{n-2} is a Robinson ordering of A.*

Proof. The proof is by induction on the size n of A. For $n < 3$ there is nothing to prove and for $n = 3$ the result holds trivially. Hence, suppose $n \geq 4$. Then, by the induction assumption, we know that the following holds:

> If σ_1 is a good SFS ordering of a Robinsonian matrix $A' \in \mathcal{S}^k$ with $k \leq n-1$ and $\sigma_i = \mathrm{SFS}_+(A', \sigma_{i-1})$ for $i \geq 2$, then σ_{k-2} is a Robinson ordering of A'.

Suppose τ_1 starts with a and ends with b. By Lemma 6.4.1, the end points of any τ_i with $i \geq 2$ are a and b (flipped at every consecutive sweep). For any $i \geq 3$,then τ_i is a 3-good SFS ordering of A. Hence, setting $S = V \setminus \{a, b\}$, in view of Lemma 6.5.8, we obtain that $\tau_{i+1}[S] = \mathrm{SFS}_+(A[S], \tau_i[S])$ for each $i \geq 3$.

Consider the order $\sigma_1 := \tau_3[S]$ and the successive sweeps $\sigma_i = \mathrm{SFS}_+(A[S], \sigma_{i-1})$ $(i \geq 2)$ returned by the multisweep algorithm applied to $A[S]$ starting from σ_1. As τ_3 is a 3-good SFS ordering of A, in view of Lemma 6.5.7 we know that σ_1 is a good SFS ordering of $A[S]$. Hence, using the induction assumption applied to $A[S]$ and σ_1, we can conclude that the sweep $\sigma_{|S|-2} = \sigma_{n-4}$ (returned by the multisweep algorithm applied to $A[S]$ with σ_1 as first sweep) is a Robinson ordering of $A[S]$.

We now observe that equality $\tau_{i+2}[S] = \sigma_i$ holds for all $i \geq 1$, using induction on $i \geq 1$. This is true for $i = 1$ by the definition of σ_1. Inductively, if $\tau_{i+2}[S] = \sigma_i$ then $\tau_{i+3}[S] = \mathrm{SFS}_+(A[S], \tau_{i+2}[S]) = \mathrm{SFS}_+(A[S], \sigma_i) = \sigma_{i+1}$. Hence, we can conclude that $\tau_{n-2}[S] = \sigma_{n-4}$ is a Robinson ordering of $A[S]$. Finally, using with Lemma 6.5.5, we can conclude that all triples (x, y, z) in τ_{n-2} that contain a or b are Robinson. Therefore we have shown that τ_{n-2} is a Robinson ordering of A, which concludes the proof. □

In other words, starting with a good SFS ordering of a Robinsonian matrix $A \in \mathcal{S}^n$, after at most $n-2$ sweeps we find a Robinson ordering of A. Finally, we can now prove Theorem 6.5.3, since the last vertex of the first sweep σ_0 in Algorithm 6.3 is an anchor of A (Theorem 6.3.8) and thus the second sweep σ_1 is a good SFS ordering.

Hence, if $A \in \mathcal{S}^n$ is a Robinsonian similarity matrix, in view of Theorem 6.5.9, the multisweep algorithm returns a Robinson ordering in at most $n-2$ sweeps starting from σ_1, and thus in at most $n-1$ sweeps counting also the initialization sweep σ_0.

6.5.4 Complexity

In this section we discuss the complexity of the SFS algorithm. As for the Lex-BFS based algorithm presented in Chapter 5, throughout we assume that

$A \in \mathcal{S}^n$ is a nonnegative symmetric matrix, given as adjacency list of an undirected weighted graph $G = (V = [n], E)$. So G is the support graph of A, whose edges are the pairs $\{x, y\}$ such that $A_{xy} > 0$ with edge weight A_{xy}, and $N(x) = \{y \in V : A_{xy} > 0\}$ is the neighborhood of $x \in V$. We assume that each vertex $x \in V = [n]$ is linked to the list of vertex/weight pairs (y, A_{xy}) for its neighbors $y \in N(x)$ and we let m denote the number of nonzero entries of A.

6.5.10 Theorem. *The SFS algorithm (Algorithm 6.1) applied to an $n \times n$ symmetric nonnegative matrix with m nonzero entries runs in $O(n + m \log n)$ time.*

Proof. As in [33] for Lex-BFS, we may assume that we are given an initial order τ of V and that the vertices and their neighborhoods are ordered according to τ (in increasing order). This assumption is useful also for the discussion of the implementation of SFS$_+$.

In order to run Algorithm 6.1, we need to update the queue ϕ consisting of the unvisited vertices at each iteration. The update consists in computing the similarity partition ψ_p with respect to the current pivot p and then refining ϕ by ψ_p.

As for Lex-BFS (Algorithm 4.3), our implementation is based on the partition refinement paradigm presented in Subsection 2.2.3. Specifically, to maintain the priority among the unvisited vertices, the queue $\phi = (B_1, \dots, B_p)$ is stored in a doubly linked list, whose elements are the classes $B_1, \dots, B_p$. Moreover each vertex has a pointer to the class B_i containing it and a pointer to its position in the class, which are updated throughout the algorithm. This data structure permits constant time insertion and deletion of a vertex in ϕ.

Initially, the queue ϕ has only one class, namely the full set V. At an iteration of Algorithm 6.1, there are three main tasks to be performed: choose the next pivot, compute the similarity partition ψ_p and refine the queue ϕ by ψ_p.

(1) Choose the new pivot $p = p_i$. Since in Algorithm 6.1 the choice of the new pivot is arbitrary in case of ties, we will choose the first vertex of the first block in ϕ. This operation can be done in constant time. We then remove p from the queue ϕ of unvisited vertices and we update the queue ϕ by deleting p from the class B_1.

(2) Compute the similarity partition $\psi_p = (C_1, \dots, C_s)$ of the set $N_\phi(p)$ with respect to $p = p_i$ (as defined in Definition 6.3.2). Here $N_\phi(p) = N(p) \cap \phi$ denotes the set of unvisited vertices in the neighborhood $N(p)$ of p. First we order the vertices y in $N_\phi(p)$ for nonincreasing values of their similarities A_{py} with respect to p, which can be done in in $O(|N_\phi(p)| \log |N_\phi(p)|)$ time using a sorting algorithm. Then we create the similarity partition $\psi_p = (C_1, \dots, C_s)$ simply by passing through the elements in $N_\phi(p)$ in the order of nonincreasing similarities to p which has just been found. This task can be done

in $O(|N_\phi(p)|)$ time. Finally we order the elements in each class C_j (increasingly) according to τ, which can be done in $O(|N_\phi(p)| \log |N_\phi(p)|)$. So we have constructed the ordered partition $\psi_p = (C_1, \dots, C_s)$ of $N_\phi(p)$ as a doubly linked list, where all classes of ψ_p are ordered according to τ. To conclude, the overall complexity of this task is bounded by $O(|N_\phi(p)| \log |N_\phi(p)|)$.

(3) The last task is to refine $\phi = (B_1, \dots, B_p)$ by $\psi_p = (C_1, \dots, C_s)$ (as defined in Definition 6.3.1). In order to obtain the new queue of unvisited vertices, we proceed as follows: starting from $j = 1$, for each class C_j of ψ, we simply remove each vertex of C_j from its corresponding class (say) B_i in ϕ and we place it in a new class B_i' which we position immediately before B_i in ϕ. Since both C_j and B_i are ordered according to τ, the initial order τ in the new block B_i' is preserved. Using the above described data structure, such tasks can be performed in $O(|C_j|)$. Once a vertex is relocated in ϕ, its pointers to the corresponding block and position in ϕ are updated accordingly. Hence this last task can be performed in time $O(\sum_{j=1}^{s} |C_j|) = O(|N_\phi(p)|)$.

Recall that at iteration i we set $p = p_i$. Then the complexity at the ith iteration is $O(1 + |N_\phi(p_i)| \log |N_\phi(p_i)|)$. Since we repeat the above three tasks for each vertex, the overall complexity is $O(\sum_{i=1}^{n} (1 + N_\phi(p_i)| \log |N_\phi(p_i)|)) = O(n + m \log n)$. □

Using the same data structure as above, we can show that the SFS$_+$ algorithm can be implemented in the same running time as the SFS algorithm.

6.5.11 Theorem. *The* SFS$_+$ *algorithm (Algorithm 6.2) applied to an $n \times n$ symmetric nonnegative matrix with m nonzero entries runs in $O(n + m \log n)$ time.*

Proof. The only difference between the SFS algorithm and the SFS$_+$ algorithm lies in the tie-breaking rule. In the SFS$_+$ algorithm, in case of ties we choose as next pivot the vertex in the slice appearing last in the given order σ. We now show that, using the same data structure and assumption as in the proof of Theorem 6.5.10, this choice can be done in constant time, and thus does not affect the complexity of the SFS$_+$ algorithm.

Recall that we assumed V to be initially ordered according to a linear order τ. Let $\overline{\sigma}$ be the reversal of σ. We now select $\tau = \overline{\sigma}$. Then, since we showed that the initial order τ is always preserved in the classes of ϕ throughout the algorithm, we ensure that the first vertex in each slice S is exactly the vertex of S appearing first in τ, i.e., last in σ.

Hence, the only thing we need to discuss is the complexity of reordering A according to τ. This can be done in $O(m + n)$ time as follows. We build a new adjacency list A' where the vertices are ordered according to τ: starting from the vertex appearing first in τ, for each vertex x in τ and for each $y \in N(x)$, we push $\tau(x)$ back in the list of A' corresponding to the neighbors of y. At the end of the process, each neighborhood in A' is then sorted according to τ.

Therefore, the overall complexity of the SFS$_+$ algorithm is $O(n + m \log n)$. $\square$

It follows directly from Theorem 6.5.11 that any SFS multisweep algorithm with k sweeps can be implemented in $O(k(n + m \log n))$. Indeed the only additional tasks we need to do are two following. First, when we start a new SFS$_+$ sweep we need to reorder the vertices and their neighborhoods according to the reversal of the previous sweep, which can be done in $O(m + n)$ time (see proof of Theorem 6.5.11). Second, we need to check if the current sweep σ_i is a Robinson ordering or not, which can be done in $O(m + n)$ time as follows: we first check if, for each $x \in [n]$, the closed neighborhood $N[x]$ is an interval in τ. If this is the case, we then check if conditions $A_{x,y} \leq A_{x,y-1}$ and $A_{x,y} \leq A_{x+1,y}$ hold for each $x <_\tau y$ such that $y \in N(x)$, otherwise the matrix is not Robinsonian. Since A is stored as an adjacency list and the (closed) neighborhood $N[x]$ of each vertex is ordered for increasing τ, it is easy to see that the above operations can be performed in $O(m + n)$ time.

Therefore, as the multisweep algorithm (Algorithm 6.3) needs $k \leq n - 1$ sweeps, it runs in time $O(n^2 + nm \log n)$.

As already mentioned in Section 6.4.1, if the matrix has only 0/1 entries, then there is no need to order the neighborhood $N(p)$ of a given pivot p, because the similarity partition ψ_p has only one class, equal to $N(p)$. For this reason, in this case the SFS algorithm can be implemented in linear time $O(m + n)$. Furthermore, as shown in Theorem 6.5.2, three sweeps suffice in the multisweep algorithm to find a Robinson ordering. Therefore, if A is a 0/1 matrix, the multisweep algorithm in Algorithm 6.3 has an overall running time of $O(m + n)$. This is coherent with the fact that in the 0/1 case SFS reduces to Lex-BFS.

When the graph G associated to the matrix A is connected the complexity of SFS and SFS$_+$ is $O(m \log n)$. Of course we may assume without loss of generality that we are in the connected case since we may deal with the connected components independently. Indeed a matrix A is Robinsonian if and only if the submatrices $A[C]$ are Robinsonian for all connected components C of G, and Robinson orderings of the connected components $A[C]$ can be concatenated to give a Robinson ordering of the full matrix A.

Finally we observe that we may also exploit the potential sparsity induced by the *largest* entries of A. While G is the graph whose edges are the pairs $\{x, y\}$ with entry $A_{xy} > 0$ (where 0 is the smallest possible entry as A is assumed to be nonnegative), we can also consider the graph G' whose edges are the pairs $\{x, y\}$ with entry $A_{xy} < \alpha_L$, where α_L is the largest possible entry of A. Let $N'(p)$ denote the neighborhood of a vertex p in G' and let m' denote the number of entries with $A_{xy} < \alpha_L$. We claim that the SFS (SFS$_+$) algorithm can also be implemented in time $O(n + m' \log n)$.

For this we modify the definition of the similarity partition of a vertex p, which is now a partition of $N'(p)$ (so that the vertices $y \notin N'(p)$ have entry $A_{py} = \alpha_L$) and the refinement of the queue ϕ by it: while we previously build the queue ϕ of unvisited vertices using a 'push-first' strategy (put the vertices with highest similarity first) we now build the queue with a 'push-last' strategy (put the vertices with lowest similarity last).

We will see in Chapter 8 another modification of the similarity partition, namely ϵ-similarity partition, which leads to the ϵ-SFS algorithm.

6.5.5 Example

We show a simple example to illustrate how Algorithm 6.3 works concretely. Consider the following matrix:

$$A = \begin{array}{c|cccccccccccc}
 & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{5} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{11} & \mathbf{13} & \mathbf{14} & \mathbf{17} & \mathbf{19} \\
\mathbf{1} & * & 0 & 7 & 3 & 3 & 3 & 0 & 3 & 3 & 4 & 0 & 3 \\
\mathbf{2} & & * & 0 & 7 & 6 & 3 & 8 & 3 & 3 & 0 & 8 & 6 \\
\mathbf{3} & & & * & 3 & 3 & 3 & 0 & 3 & 3 & 8 & 0 & 3 \\
\mathbf{5} & & & & * & 6 & 5 & 7 & 5 & 5 & 3 & 7 & 8 \\
\mathbf{7} & & & & & * & 5 & 6 & 5 & 5 & 3 & 6 & 7 \\
\mathbf{8} & & & & & & * & 4 & 8 & 6 & 5 & 4 & 5 \\
\mathbf{9} & & & & & & & * & 4 & 3 & 0 & 8 & 6 \\
\mathbf{11} & & & & & & & & * & 7 & 5 & 4 & 5 \\
\mathbf{13} & & & & & & & & & * & 5 & 3 & 5 \\
\mathbf{14} & & & & & & & & & & * & 0 & 3 \\
\mathbf{17} & & & & & & & & & & & * & 6 \\
\mathbf{19} & & & & & & & & & & & & *
\end{array} \tag{6.13}$$

which is a (connected) submatrix of the matrix considered in the example in Section 5.6. Below are illustrated all the iterations of the SFS algorithm using as initial order of the vertices the reversal of the original labeling of the matrix. At each iteration, the vertices in the blocks are the univisited vertices in the queue. In bold the pivot which is chosen at the current iteration. The numbers above the blocks represent the similarity between the new pivot and the vertices in the blocks of the queue. The first line on the top-left shows the initialization step of the algorithm.

Note that in this case we have $\sigma_2 = \overline{\sigma}_1$. This is coherent with Lemma 6.4.2(ii), since σ_2 is a Robinson ordering of A. Hence, in this example our SFS multisweep algorithm finds a Robinson ordering in 2 sweeps only.

We give in Figure 6.8 the similarity layers structures $\mathcal{L}_+$ and $\mathcal{L}_{++}$ corresponding to the similarity layer rooted at the first vertex of $\sigma_+ = \sigma_1$ and at the first vertex of $\sigma_{++} = \sigma_2$, respectively. Both similarity layers are well defined as they start with an anchor of A (since A is indeed Robinsonian). For the sake of visualization, weights on the edges are omitted.

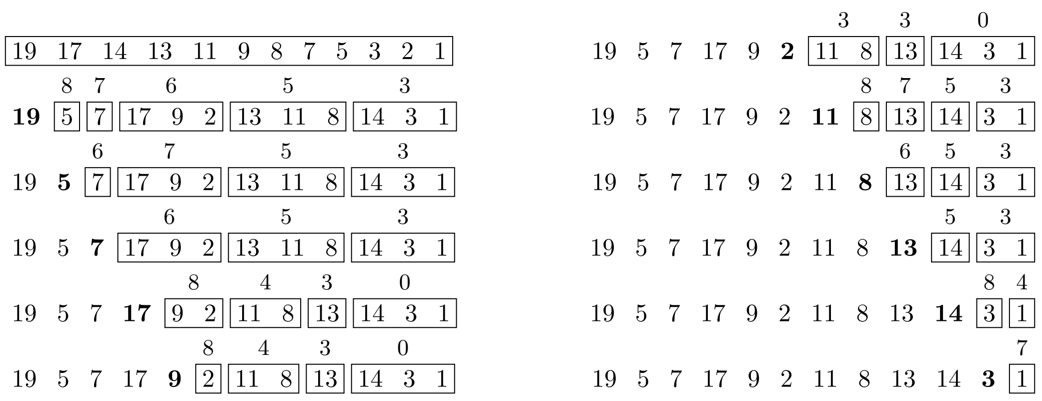


Figure 6.5: First iteration σ_0 of Algorithm 6.3 applied to the matrix in (6.13).

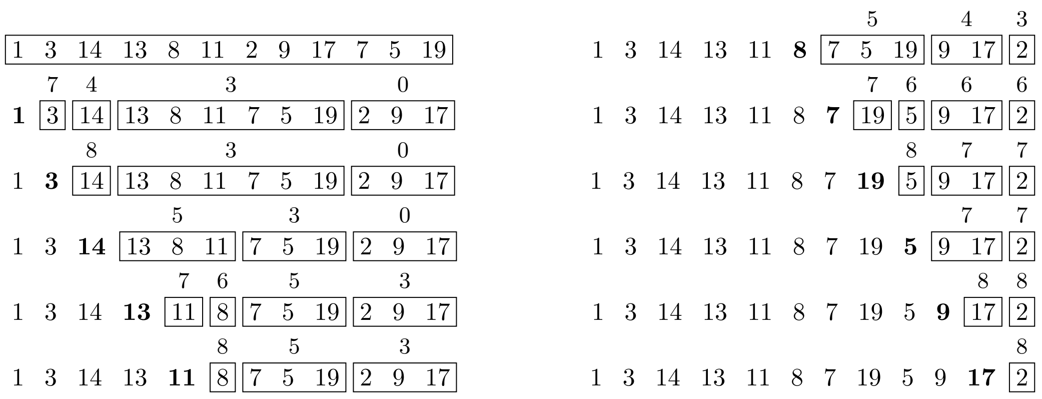


Figure 6.6: Second iteration σ_1 of Algorithm 6.3 applied to the matrix in (6.13).

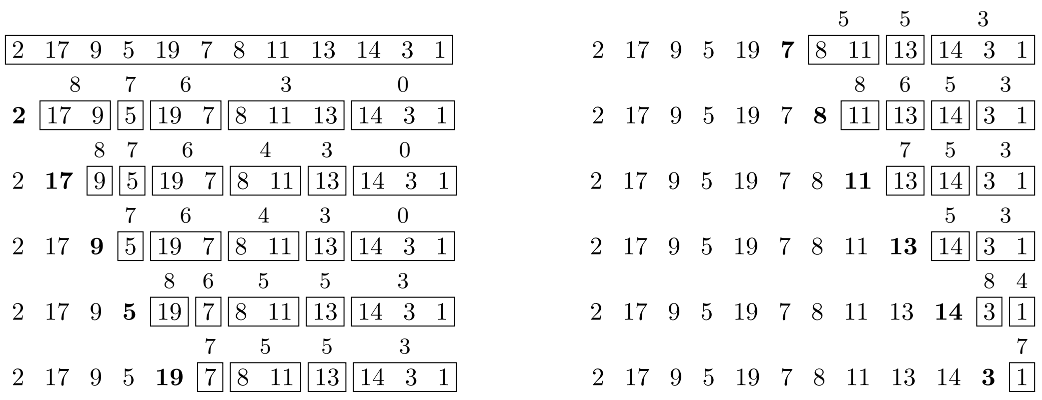
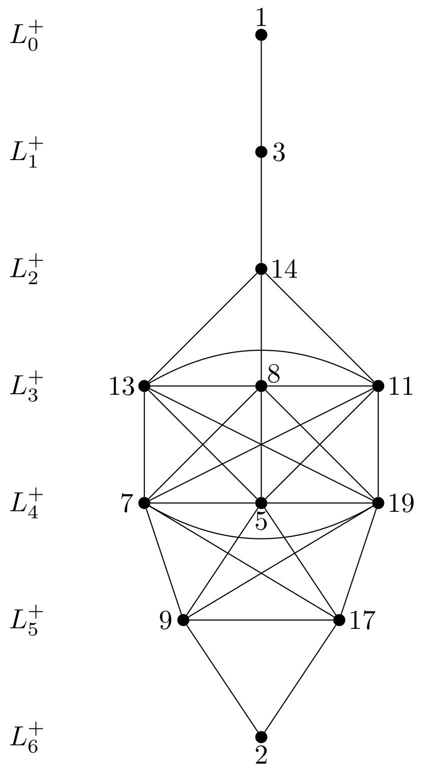


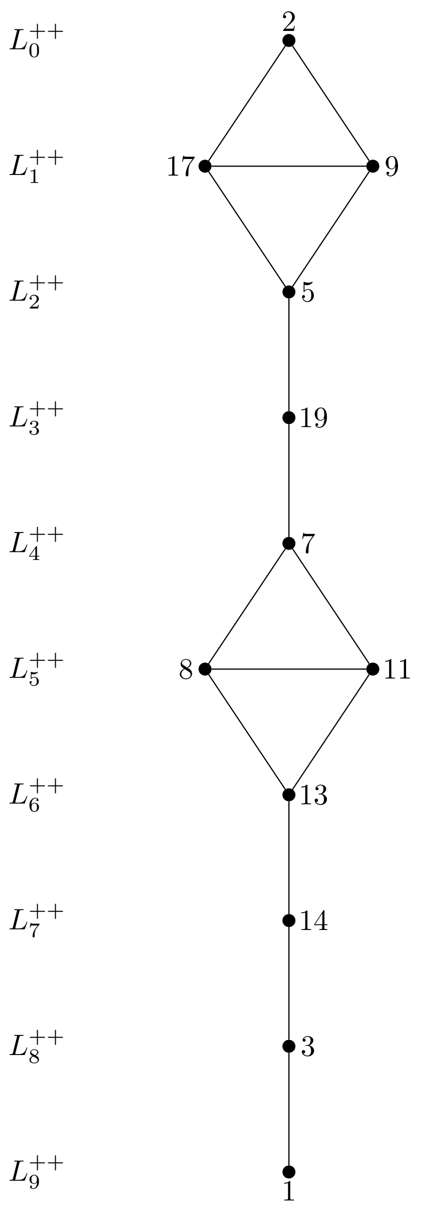
Figure 6.7: Third iteration σ_2 of Algorithm 6.3 applied to the matrix in (6.13).

Furthermore, we draw only the edges among consecutive layers and intralayers. Note that, differently from the layers in Figure 4.4 at 57, in this case the number of layers is not the same for consecutive sweeps.

Each layer corresponds to a slice at some iteration of the SFS algorithm (see Figures 6.6 and 6.7).



(a) Similarity layers of σ_+.



(b) Similarity layers of σ_{++}.

Figure 6.8: Similarity layers of the second and third sweeps $\sigma_+ = \sigma_1$ and $\sigma_{++} = \sigma_2$ of the SFS multisweep algorithm (Algorithm 6.3) applied to the matrix in (6.13) (weights are omitted for a better visualization).

6.6 Conclusions and future work

In this chapter we have introduced the new search algorithm *Similarity-First Search* (SFS) and its variant SFS$_+$, which are generalizations to weighted graphs of the classical Lex-BFS algorithm and its variant Lex-BFS$_+$.

The algorithm is entirely based on the main task of partition refinement, it is conceptually simple and easy to implement. We have shown that a multisweep algorithm can be designed using SFS and SFS$_+$, which permits to recognize if a symmetric $n \times n$ matrix is Robinsonian and if so to return a Robinson ordering after at most $n-1$ sweeps. We believe that this recognition algorithm is simpler than the other existing algorithms. Moreover, to the best of our knowledge, this is the first work extending multisweep graph search algorithms to the setting of weighted graphs (i.e., matrices).

Algorithm 6.3 is substantially different from the algorithm presented in Chapter 5 (Algorithm 5.5). Both algorithms are inspired by the Lex-BFS algorithm with partition refinement implementation (Algorithm 4.3). Nevertheless, Algorithm 5.5 aims to decompose the original matrix in many 0/1 submatrices, which solves using Lex-BFS. On the other hand, Algorithm 6.3, intuitively, uses Lex-BFS (in fact, SFS) without the need to decompose the original matrix, which can lead to really efficient performance especially when there are many distinct values (compared with Algorithm 5.5), since the less distinct values the more ties we have in the SFS algorithm, which in the worst case could lead to compute all the $n-1$ sweeps theoretically needed to recognize a Robinsonian matrix. Furthermore, another difference between the two algorithms is that Algorithm 5.5 can be extended to return all the Robinson orderings (Algorithm 5.7). Whether also Algorithm 6.3 can be extended to return a PQ-tree of all Robinson orderings is an open question. As we will see in Chapter 9, when A is Robinsonian then the SFS multisweep outperforms the Lex-BFS based algorithm.

Our algorithm can also be used to recognize Robinsonian dissimilarities. Recall that $D \in \mathcal{S}^n$ is a *Robinson dissimilarity matrix* if $D_{xz} \geq \max\{D_{xy}, D_{yz}\}$ for all $1 \leq x < y < z \leq n$, and a *Robinsonian dissimilarity* if its rows and columns can be simultaneously reordered to get a Robinson dissimilarity matrix. Clearly D is a Robinsonian dissimilarity matrix if and only if the matrix $A = -D$ is a Robinsonian similarity matrix. Therefore, one can check whether D is a Robinsonian dissimilarity by applying the SFS-based multisweep algorithm to the matrix A.

Alternatively one may also modify the SFS algorithm so that it can deal directly with dissimilarity matrices. Say D is a nonnegative dissimilarity matrix and G is the corresponding weighted graph with edges the pairs $\{x, y\}$ with $D_{xy} > 0$. Then we can modify the SFS algorithm as follows. First, we now order the vertices in the neighborhood $N(p)$ of a vertex p for *nondecreasing values* of the dissimilarities D_{py} (instead of nonincreasing values of the similarities A_{py} as was the case in SFS). Then we construct the (dis)similarity partition ψ_p of $N(p)$

by grouping the vertices with the same dissimilarity to p, in increasing values of the dissimilarities. Finally, when refining the queue ϕ by ψ_p, we apply a 'push-first' strategy and place the vertices with lowest dissimilarity first. The resulting algorithm, which we name DiSFS, standing for *Dissimilarity-Search First*, has the same running time in $O(n+m\log n)$. Moreover, as explained above in Subsection 6.5.4, it can also be implemented in time $O(n+m'\log n)$, where m' denotes the number of entries of D satisfying $D_{xy} < D_{\max}$ and $D_{\max}$ denotes the largest entry of D. Using DiSFS we can define the analogous multisweep algorithm for recognizing Robinsonian dissimilarities in time $O(n^2 + nm\log n)$ (or $O(n^2 + nm'\log n)$).

It is an open question whether the number of sweeps needed to recognize Robinsonian (dis)similarities can be bounded by a constant number. If this would be the case, then the multisweep algorithm would have running time $O(n + m\log n)$ and thus become (theoretically) competitive with the optimal one in [94]. However, as we have seen in Example 6.5.1 in Subsection 6.5.1, there are examples for $n = 4, 5, 6$ where $n-1$ sweeps are needed. Hence, in order to show a constant number of sweeps, one possibly needs to define another variant of SFS, where ties are broken using the SFS orderings returned by two previous sweeps (and not only one as in the SFS$_+$ variant). This approach has been successfully applied to Lex-BFS for the recognition of interval graphs in five Lex-BFS sweeps [38], where the last sweep used is the variant Lex-BFS$_*$, which breaks ties using the linear order returned by two previous sweeps. Dusart and Habib [51] conjecture that a similar approach applies to recognize cocomparability graphs with a constant number of sweeps. Investigating whether such an approach applies to Robinsonian matrices will be the subject of future work. We will see in Subsection 9.2 that, in our computational experiments, the maximum number of sweeps for verifying Robinsonian matrices was at most 4 (even for 1000×1000 matrices). On the other hand, some matrices required $n-1$ sweeps in order to be recognized as non-Robinsonian.

Finally, it will be interesting to investigate whether the new SFS algorithm can be used to study other classes of structured matrices. For example, as Robinsonian matrices represent a generalization of unit interval graphs to weighted graphs, one could also define a generalization of other graph classes (e.g., chordal, interval) to weighted graphs and study the properties of the SFS algorithm on such classes. More generally, the SFS algorithm could be used as tool in the area of similarity search and clustering analysis.

Part III

QAP and Robinsonian approximation

7

Seriation and the quadratic assignment problem

In this chapter we focus on a specific model of seriation based on the Quadratic Assignment Problem (QAP). In Section 7.1 we introduce QAP and we discuss some polynomial solvable cases. In Section 7.2 we present our main result, by showing that a Robinson ordering gives an optimal solution for a special class of QAP involving Robinsonian and Toeplitz matrices. Finally, in Section 7.3 we conclude with some possible direction for future work. The content of the chapter is based on our work [78].

7.1 The Quadratic Assignment Problem

The *Quadratic Assignment Problem* (QAP) is a well studied hard combinatorial optimization problem, which was introduced by Koopmans and Beckman [76] in 1957 as a mathematical model for the location of indivisible economic activities. In QAP(A, B), we are given n facilities, n locations, a *flow matrix* A whose entry A_{ij} represents the flow of activity between two facilities i and j, and a *distance matrix* B whose entry B_{ij} represents the distance between the locations i and j. Then the objective is to find an assignment of the facilities to the locations minimizing the total cost of the assignment.

As in 2.2.1, we denote by π a permutation of $[n]$ and by $\Pi \in \{0,1\}^{n\times n}$ the permutation matrix with entry $\Pi_{ij} = 1$ if $\pi(i) = j$. Then, $\Pi A \Pi^{\mathsf{T}} = (A_{\pi(i),\pi(j)})_{i,j=1}^{n}$ is the matrix obtained by symmetrically permuting rows and columns of A.

QAP corresponds to solving the following optimization problem over the set $\mathcal{P}_n$ of all possible permutations π of $[n]$.

QAP(A,B) Given a flow matrix $A \in \mathcal{S}^n$ and a distance matrix $B \in \mathcal{S}^n$, find a permutation π of $[n]$ minimizing:

$$\min_{\pi \in \mathcal{P}_n} \sum_{i,j=1}^{n} A_{\pi(i)\pi(j)} B_{ij}, \tag{7.1}$$

where $A_{\pi(i)\pi(j)} B_{ij}$ is the cost inferred by assigning facility i to location $\pi(i)$ and facility j to location $\pi(j)$.

In other words, QAP attempts to find a permutation of the rows and columns of A minimizing (7.1), which corresponds to finding a linear order of the objects represented by the rows/columns of A.

QAP has been extensively studied in the past decades, in particular due to its many real world applications. We refer to [21, 23] and references therein for an exhaustive survey. QAP is an NP-hard problem and it cannot even be approximated within a constant factor [103]. However, there exist many special cases which are solvable in polynomial time by exploiting the structure of the matrices A, B. In this section we discuss some of these special cases known in the literature. Furthermore, we present the 2-SUM problem, which is a special QAP instance that can be optimally solved by a Robinson ordering when the flow matrix is Robinsonian.

7.1.1 Polynomial solvable cases of QAP

We are interested in the 'easy cases' of QAP where an optimal solution is known in explicit form and is represented by a *fixed* permutation. These cases have a practical importance in designing heuristics, approximation and enumeration algorithms and they occur when the matrices A and B have a specific ordered structure, like being Monge, Toeplitz or monotone matrices (see [20, 22, 42, 41], [23, §8.4] for a survey and the recent works [26, 56]).

For instance, if A is monotonically nondecreasing, i.e., if its rows (columns) are nondecreasing from left to right (from top to bottom) and B is monotonically nonincreasing (symmetrically defined), then it is known that the identity permutation is an optimal solution to QAP(A, B) [23, Proposition 8.23].

Another instance of QAP(A, B) for which the identity permutation is optimal arises when A is a Kalmanson matrix, i.e., A is symmetric and satisfies:

$$\max\{A_{ij} + A_{kl}, A_{il} + A_{jk}\} \leq A_{ik} + A_{jl} \quad \text{for all } 1 \leq i < j < k < l \leq n,$$

and B is a symmetric circulant matrix (i.e., B_{ij} depends only on $|i-j|$ modulo n) satisfying $B_{12} \leq B_{13} \leq \ldots \leq B_{1,\lfloor n/2 \rfloor+1}$ [41, Theorem 2.3]. This extends an earlier result of Kalmanson [74] when B is the adjacency matrix of the cycle $(1, 2, \ldots, n)$, in which case QAP(A, B) models the shortest Hamiltonian cycle problem on the flow matrix A. Finally, Çela *et al.* considered very recently a new tractable case of QAP(A,B) with A being a special Robinson dissimilarity [25].

7.1.2 2-SUM and seriation

We describe the 2-SUM problem, which is a special instance of QAP (7.1) where the distance matrix is $B = ((i-j)^2)_{i,j=1}^n$.

2-SUM problem Given a matrix $A \in \mathcal{S}^n$, find a permutation π of $[n]$ which minimizes:

$$\min_{\pi \in \mathcal{P}_n} \sum_{i,j=1}^{n} A_{\pi(i)\pi(j)}(i-j)^2. \tag{7.2}$$

As already discussed in Section 1.1, the 2-SUM problem is used to model the general seriation problem (see, e.g., [5, 58]). In this subsection we will show an interesting result presented in [56] when the matrix A in (7.2) is a Robinsonian similarity. The authors of [58] show that the 2-SUM problem (7.2) is an NP-complete problem and they use the spectral method of reordering the Fiedler vector of A described in Subsection 3.2.3 to produce a heuristic solution. This in turn permits to bound important matrix structural parameters, like envelope-size and bandwidth [7]. However, no assumption is made on the structure of the matrix A. As observed in [58, 56], the following fact can be used to motivate the spectral approach for 2-SUM (7.2). If we define the vectors $x = (1, \ldots, n)^\mathsf{T}$ and $x_\pi = (\pi(1), \ldots, \pi(n))^\mathsf{T} \in \mathbb{R}^n$ for $\pi \in \mathcal{P}_n$, then $\sum_{i,j=1}^n A_{ij}(\pi(i) - \pi(j))^2 = (x_\pi)^\mathsf{T} L_A x_\pi$, where L_A denotes the Laplacian of A (see Subsection 3.2.3). Therefore, computing the Fiedler value arises as a natural continuous relaxation for the 2-SUM problem (7.2).

Fogel *et al.* [56] pointed out an interesting connection between Robinsonian matrices and the 2-SUM problem (7.2), which was in fact our main motivation for our work in [78]. Specifically, they consider a special class of Robinson similarity matrices for which they can show that the identity permutation is optimal for 2-SUM (7.2), namely interval-cut matrices, defined as follows. Given two integers $1 \leq u \leq v \leq n$, the *interval-cut matrix* $I(u, v)$ is the symmetric $n \times n$ matrix with (i, j)-th entry 1 if $u \leq i, j \leq v$ and 0 otherwise. Clearly each interval-cut matrix is a Robinson similarity and thus conic combinations of interval-cut matrices are Robinson matrices as well. The following result is shown in [56], for which we give a short proof.

7.1.1 Theorem. *[56] If $A \in \mathcal{S}^n$ can be written as a conic combination of interval-cut matrices, then the identity permutation is optimal for 2-SUM (7.2). More generally if, for some $\pi \in \mathcal{P}_n$, A_π can be written as a conic combination of interval-cut matrices, then π is optimal for 2-SUM (7.2).*

Proof. First we show the result when A is an interval-cut matrix. Say, $A = I(u,v)$ and set $t = v - u + 1$. Then, we need to show that:

$$\sum_{i,j=u}^{v} (\pi(i) - \pi(j))^2 \geq \sum_{i,j=u}^{v} (i-j)^2$$

for any permutation $\pi \in \mathcal{P}_n$. Suppose that π maps the elements of the interval $[u,v]$ to the elements of the set $\{i_1, \ldots, i_t\}$, ordered as $1 \leq i_1 < \ldots < i_t \leq n$. Because the left-hand side of the above inequality involves all pairs of indices in the interval $[u,v]$, equivalently, we need to show that

$$\sum_{1 \leq r < s \leq t} (i_r - i_s)^2 \geq \sum_{1 \leq r < s \leq t} (r-s)^2.$$

Now the latter easily follows from the fact that $|i_r - i_s| \geq |r - s|$ for all r, s. Hence we have shown that the identity permutation is an optimal solution of 2-SUM (7.2) when A is an interval-cut matrix and this easily implies that this also holds when A is a conic combination of interval-cut matrices. The second statement follows as a direct consequence. $\square$

In other words, the above result shows that for Robinsonian similarity matrices as in Theorem 7.1.1, any permutation reordering A as a Robinson similarity matrix also solves (7.2). As not every Robinson similarity is a conic combination of interval-cut matrices, this raises the question whether the above result extends to the case when A is an arbitrary Robinson similarity matrix.

There is a second possible way in which one may want to generalize the result of Theorem 7.1.1. We introduce the concept of *Toeplitz* matrix, which is a matrix $B \in \mathbb{R}^{n \times n}$ with constant entries along its diagonals, i.e., $B_{ij} = B_{i+1,j+1}$ for all $1 \leq i, j \leq n-1$. The 2-SUM problem (7.2) is the instance of QAP(A,B) (7.1), where the distance matrix is $B = ((i-j)^2)_{i,j=1}^n$, which turns out to be a Toeplitz dissimilarity Robinson matrix. In fact there are many other interesting classes of QAP whose distance matrix B is a Toeplitz Robinson dissimilarity matrix. For instance, QAP(A,B) models the minimum linear arrangement (aka 1-SUM) problem when $B = (|i-j|)_{i,j=1}^n$ and, more generally, the p-SUM problem when we have $B = (|i-j|^p)_{i,j=1}^n$, for $p \geq 1$. Moreover, QAP(A,B) models the minimum bandwidth problem when A is the adjacency matrix of a graph and B is of the form B_n^{Δ}, as defined in relation (7.3) in Section 7.2 below. For more details on these and other graph (matrix) layout problems with practical impact we refer to the survey [44] and references therein.

This thus raises the further question whether the result of Theorem 7.1.1 extends to instances of QAP(A,B), where B is an arbitrary Toeplitz Robinson dissimilarity matrix. This is precisely what we do in this chapter. We remove both assumptions on A and B and show that the identity permutation is optimal for QAP(A,B) when A is any Robinson similarity and B is any Robinson dissimilarity, assuming that B (or A) has a Toeplitz structure.

7.2 QAP is easy for Robinsonian matrices

The main contribution of this section is to provide a new class of QAP instances that admit a closed form optimal solution. Specifically, the result can be formulated as follows. Assume that A is a Robinson similarity, B is a Robinson dissimilarity and that one of the two matrices A or B is a Toeplitz matrix. Then the identity permutation is an optimal solution for QAP(A, B) (Theorem 7.2.5). From this we derive a more general result when both matrices are Robinsonian (Corollary 7.2.6). Hence our result uncovers an interesting connection between QAP and Robinsonian matrices and introduces a new class of QAP instances which is solvable in polynomial time.

In order to introduce the new class of polynomial solvable QAP, we first present some key (easy) observations. Namely, if B is a Toeplitz Robinson dissimilarity matrix, then we can decompose B as a conic combination of 0/1 Toeplitz Robinson dissimilarities. For this, given an integer $\Delta \in [n]$, we define the symmetric matrix $B_n^\Delta \in \mathcal{S}^n$ with entries

$$\left(B_n^\Delta\right)_{ij} = \begin{cases} 1 & \text{if } |i-j| \geq n-\Delta \\ 0 & \text{else} \end{cases} \qquad \text{for } \; i,j = 1,\ldots,n. \tag{7.3}$$

Note that for $\Delta = n$, we have that $B_n^\Delta = J$, i.e., the all-ones matrix. Clearly, each matrix B_n^Δ is a Toeplitz matrix and a Robinson dissimilarity. In fact all Toeplitz Robinson dissimilarities can be decomposed in terms of these matrices B_n^Δ.

7.2.1 Lemma. *Let $B \in \mathcal{S}^n$ be a Toeplitz matrix and let $\beta_0, \ldots, \beta_{n-1} \in \mathbb{R}$ such that $B(i,j) = \beta_k$ for all $i,j \in [n]$ with $|i-j| = k$ for $0 \leq k \leq n$. Then,*

$$B = \beta_0 J + \sum_{k=1}^{n-1} (\beta_k - \beta_{k+1}) B_n^{n-k}. \tag{7.4}$$

If moreover B is a Robinson dissimilarity, i.e., if $\beta_0 = 0 \leq \beta_1 \leq \ldots \leq \beta_{n-1}$, then B is a conic combination of the matrices B_n^Δ (for $\Delta = 1, \ldots, n-1$).

Proof. Direct verification. □

Our main result, which we show in this section, is that the identity permutation is optimal for QAP(A, B_n^Δ) for any integer $1 \leq \Delta \leq n-1$.

7.2.2 Theorem. *Let $A \in \mathcal{S}^n$ be a Robinson similarity matrix and let $\Delta \in [n-1]$. Then, for any permutation π of $[n]$, we have:*

$$\langle A_\pi, B_n^\Delta \rangle \geq \langle A, B_n^\Delta \rangle. \tag{7.5}$$

As we will see at the end of the section, using Theorem 7.2.2 we can easily derive that the identity permutation is an optimal solution for QAP(A, B) when A is a Robinson similarity, B is a Robinson dissimilarity and one of the two matrices A or B is a Toeplitz matrix (Theorem 7.2.5). Hence, in the rest of the section we show how to prove the correctness of Theorem 7.2.2.

7.2.1 Sketch of proof

In order to show the correctness of Theorem 7.2.2 we need to prove relation (7.5). Let E_n^Δ denote the support of the matrix B_n^Δ, i.e., the set of upper triangular positions where B_n^Δ has a nonzero entry. That is:

$$E_n^\Delta = \{\{i, n-\Delta+j\} : 1 \leq i \leq j \leq \Delta\}.$$

Then we can reformulate the inner products in (7.5) as

$$\langle A_\pi, B_n^\Delta \rangle = \sum_{i,j=1}^n A_{\pi(i),\pi(j)} \left(B_n^\Delta\right)_{i,j} = 2 \sum_{\{i,j\}\in E_n^\Delta} A_{\pi(i),\pi(j)},$$

$$\langle A, B_n^\Delta \rangle = \sum_{i,j=1}^n A_{i,j} \left(B_n^\Delta\right)_{i,j} = 2 \sum_{\{i,j\}\in E_n^\Delta} A_{i,j},$$

and (7.5) is equivalent to the following inequality:

$$\sum_{\{i,j\}\in E_n^\Delta} A_{\pi(i),\pi(j)} \geq \sum_{\{i,j\}\in E_n^\Delta} A_{ij}. \tag{7.6}$$

We show the inequality (7.6) using induction on $n \geq 2$. The base case $n = 2$ is trivial, since then $\Delta = 1$ and both summations in (7.6) are identical. We now assume that the result holds for $n-1$ and we show that it also holds for n. For the remaining of the proof, we fix a Robinson similarity matrix $A \in \mathcal{S}^n$, an integer $\Delta \in [n-1]$ and a permutation π of $[n]$. Moreover we let $k \in [n]$ denote the index such that $n = \pi(k)$.

The key idea in the proof is to show that there exist a subset $F \subseteq E_n^\Delta$ and a permutation τ of $[n]$ satisfying the following properties:

(C1) $|F| = \Delta$,

(C2) the indices $\min\{\pi(i), \pi(j)\}$ for the pairs $\{i, j\} \in F$ are pairwise distinct,

(C3) $\tau(n) = k$ and the set $R := E_n^\Delta \setminus F$ satisfies

$$R = \tau(E_{n-1}^{\Delta-1}) := \{\{\tau(i), \tau(j)\} : \{i, j\} \in E_{n-1}^{\Delta-1}\}.$$

Here the set $E_{n-1}^{\Delta-1}$ is the support of the matrix $B_{n-1}^{\Delta-1}$, defined by

$$E_{n-1}^{\Delta-1} = \{\{i, n-\Delta+j\} : 1 \le i \le j \le \Delta - 1\},$$

so that E_n^{Δ} is partitioned into the two sets $\{\{i,n\} : 1 \le i \le \Delta\}$ and $E_{n-1}^{\Delta-1}$.

In a first step, we show (in Lemma 7.2.3 below) that if we can find a set F and a permutation τ satisfying (C1)-(C3), then we can conclude the proof of the inequality (7.6) using the induction assumption. The proof relies on the following idea: we split the summation

$$\Sigma_\pi(A) := \sum_{\{i,j\}\in E_n^{\Delta}} A_{\pi(i),\pi(j)} \tag{7.7}$$

into two terms, obtained by summing over the set F and over its complement R, and we show that the first term is at least $\sum_{i=1}^{\Delta} A_{in}$ (using the conditions (C1)-(C3)) and that the second term is at least $\sum_{\{i,j\}\in E_{n-1}^{\Delta-1}} A_{ij}$ (using the induction assumption applied to the smaller Robinson similarity $(A_{ij})_{i,j=1}^{n-1}$).

In a second step, we formulate (in Lemma 7.2.4 below) two new conditions (C4) and (C5) which together with (C1),(C2) imply (C3). These two conditions will be simpler to check than (C3).

7.2.2 Intermediate results

In this subsection we show the correctness of Lemma 7.2.3 and Lemma 7.2.4. These two lemmas are intermediate results which will be used to conclude the proof of Theorem 7.2.2 in Section 7.3.

7.2.3 Lemma. *Assume that there exist a set $F \subseteq E_n^{\Delta}$ and a permutation τ of $[n]$ satisfying (C1)-(C3), then the inequality (7.6) holds.*

Proof. Let us decompose the summation $\Sigma_\pi(A)$ from (7.7) as the sum of the two terms:

$$\Sigma_\pi(A) = \Sigma_{\pi,F}(A) + \Sigma_{\pi,R}(A), \tag{7.8}$$

where we set:

$$\Sigma_{\pi,F}(A) := \sum_{\{i,j\}\in F} A_{\pi(i),\pi(j)}, \quad \Sigma_{\pi,R}(A) := \sum_{\{i,j\}\in R} A_{\pi(i),\pi(j)}.$$

We now bound each term separately. First we consider the term $\Sigma_{\pi,F}(A)$. As A is a Robinson similarity matrix, it follows that for all indices $i,j \in [n]$:

$$A_{ij} \ge A_{n,\min\{i,j\}}.$$

Hence, we can deduce:

$$\Sigma_{\pi,F}(A) = \sum_{\{i,j\}\in F} A_{\pi(i),\pi(j)} \geq \sum_{\{i,j\}\in F} A_{n,\min\{\pi(i),\pi(j)\}} \geq \sum_{i=1}^{\Delta} A_{n,i}, \tag{7.9}$$

where for the right most inequality we have used the conditions (C1),(C2) combined with the fact that A is a Robinson similarity.

We now consider the second term $\Sigma_{\pi,R}(A)$. Define the permutation $\sigma = \pi\tau$. Then, by (C3), we have that $\sigma(n) = \pi(k) = n$ and thus $\sigma(E_{n-1}^{\Delta-1}) = \pi(\tau(E_{n-1}^{\Delta-1})) = \pi(R)$. We can then write:

$$\Sigma_{\pi,R}(A) = \sum_{\{i,j\}\in E_{n-1}^{\Delta-1}} A_{\sigma(i),\sigma(j)}. \tag{7.10}$$

As $\sigma(n) = n$, the permutation σ of $[n]$ induces a permutation σ' of $[n-1]$. We let $A', B' \in \mathcal{S}^{n-1}$ denote the principal submatrices obtained by deleting the row and column indexed by n in A and in B_n^{Δ}, respectively. Then A' is again a Robinson similarity matrix (now of size $n-1$) and $B' = B_{n-1}^{\Delta-1}$ is supported by the set $E_{n-1}^{\Delta-1}$. Then, using the induction assumption applied to A', $\Delta - 1$ and σ', we obtain:

$$\Sigma_{\sigma'}(A') := \sum_{\{i,j\}\in E_{n-1}^{\Delta-1}} A'_{\sigma(i),\sigma(j)} \geq \Sigma_{\text{id}}(A') := \sum_{\{i,j\}\in E_{n-1}^{\Delta-1}} A'_{i,j}. \tag{7.11}$$

Using (7.10), we get:

$$\Sigma_{\pi,R}(A) = \Sigma_{\sigma'}(A') \geq \Sigma_{\text{id}}(A') = \sum_{\{i,j\}\in E_{n-1}^{\Delta-1}} A_{i,j}. \tag{7.12}$$

Finally, combining (7.8),(7.9) and (7.12), we get the desired inequality:

$$\Sigma_{\pi}(A) \geq \sum_{i=1}^{\Delta} A_{i,n} + \sum_{\{i,j\}\in E_{n-1}^{\Delta-1}} A_{ij} = \sum_{\{i,j\}\in E_n^{\Delta}} A_{ij},$$

which concludes the proof. □

7.2.4 Lemma. *Assume that the sets $F \subseteq E_n^{\Delta}$ and $R := E_n^{\Delta} \setminus F$ satisfy the conditions (C1),(C2) and the following two conditions:*

(C4) no pair in the set R contains the element k,

(C5) no pair $\{i, n-\Delta+i\}$ with $1 \leq i \leq \Delta$ and $k+1-n+\Delta \leq i \leq k-1$ belongs to the set R.

Define the permutation $\tau = (k, k+1, \ldots, n)$. Then, we have $R = \tau(E_{n-1}^{\Delta-1})$.

Proof. By (C1), $|F| = \Delta$, thus R has the same cardinality as the set $\tau(E_{n-1}^{\Delta-1})$ and therefore it suffices to show the inclusion $R \subseteq \tau(E_{n-1}^{\Delta-1})$. For this consider a pair $\{i, n-\Delta+j\}$ in R, where $1 \le i \le j \le \Delta$. We show that $i = \tau(r)$ and $j = \tau(s)$ for some $1 \le r \le s \le \Delta - 1$. In view of (C4), we shall define r, s depending on whether i lies before or after k, getting the following cases:

1) $i \le k-1$, then $r = i$ and $i = \tau(i)$,

2) $i \ge k+1$, then $r = i-1$ and $i = \tau(i-1)$.

We do the same for index j, getting the following cases:

a) $n - \Delta + j \le k - 1$, then $s = j$ and $n - \Delta + j = \tau(n - \Delta + j)$,

b) $n - \Delta + j \ge k + 1$, then $s = j - 1$ and $n - \Delta + j = \tau(n - \Delta + j - 1)$.

It remains only to check that $1 \le r \le s \le \Delta - 1$ holds. For this, we now discuss all possible combinations for indices i and j according to the above cases:

1a) Then, $r = i$ and $s = j$. Since $1 \le i \le j \le \Delta$, we only have to check that $j \le \Delta - 1$. Indeed, if $j = \Delta$, then from a) we get $n \le k-1$, which is impossible.

1b) Then, $r = i$ and $s = j-1$. It suffices to check that $r \le s$, i.e., $i \neq j$. Indeed, assume that $i = j$. Then, the pair $\{i, n-\Delta+i\}$ belongs to R with $i \le k-1$ (as we are in case 1) for index i) and $i \ge k+1-n+\Delta$ (as we are in case b) for index j), which contradicts the condition (C5).

2a) Then, $r = i-1$ and $s = j$. It suffices to check that $r \ge 1$ and $s \le \Delta - 1$. The first one holds since $i \ge 2$ as we are in case 2) for index i. The second one is also true, since we are in case a) for index j and $k \le n$.

2b) Then, $r = i-1$ and $s = j-1$. It suffices to check that $r \ge 1$, which holds since we are in case 2) for index i and thus $i \ge 2$.

Thus we have shown that $R \subseteq \tau(E_{n-1}^{\Delta-1})$, which concludes the proof. □

7.2.3 Conclusion of the proof

In view of Lemmas 7.2.3 and 7.2.4, in order to conclude the proof of Theorem 7.2.2, i.e., show that the inequality (7.6) holds, it suffices to find a set $F \subseteq E_n^{\Delta}$ and a permutation τ of $[n]$ satisfying the conditions (C1), (C2), (C4) and (C5).

For the permutation τ, we choose $\tau = (k, k+1, \ldots, n)$ as in Lemma 7.2.4, thus $\tau(n) = k$. It remains to construct the set F. This is the last step in the proof which is a bit technical.

The following terminology will be useful, regarding the pairs $\{i, n-\Delta+j\}$ (for $1 \le i \le j \le \Delta$) in the set E_n^Δ. We refer to the pairs $\{i, n-\Delta+i\}$ (with $1 \le i \le \Delta$) as the *diagonal pairs*. Furthermore, for each $1 \le i_0 \le \Delta$, we refer to the pairs $\{i_0, n-\Delta+j\}$ (with $i_0+1 \le j \le \Delta$) as the *horizontal pairs* on row i_0, meaning the pairs of E_n^Δ in the row indexed by i_0. Finally, for each $k_0 = n-\Delta+j_0$ such that $1 \le j_0 \le \Delta$, we refer to the pairs $\{i, n-\Delta+j_0\}$ (with $1 \le i \le j_0-1$) as the *vertical pairs* on column k_0, meaning the pairs of E_n^Δ in the column indexed by k_0. Note that, in both horizontal and vertical pairs, the diagonal pair $\{i, n-\Delta+i\}$ is not included. As an illustration see Figure 7.1.

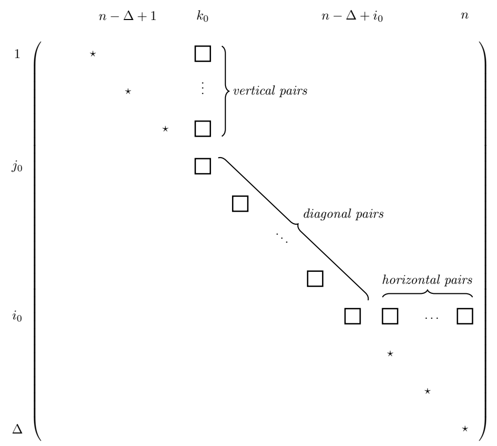


Figure 7.1: Vertical, diagonal and horizontal pairs in the set E_n^Δ.

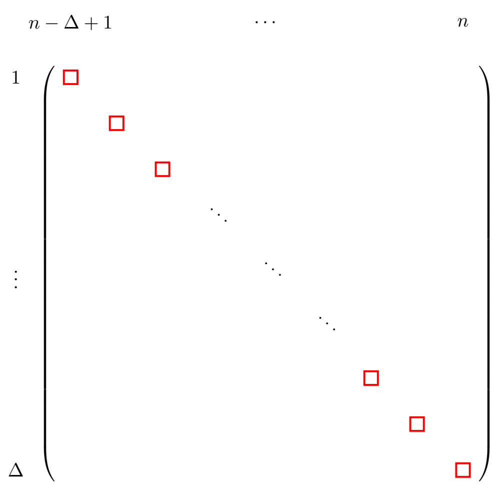
Moreover, we denote by:

$$U = U_R \cup U_C, \text{ where } U_R = \{1, \ldots, \Delta\}, \quad U_C = \{n-\Delta+1, \ldots, n\},$$

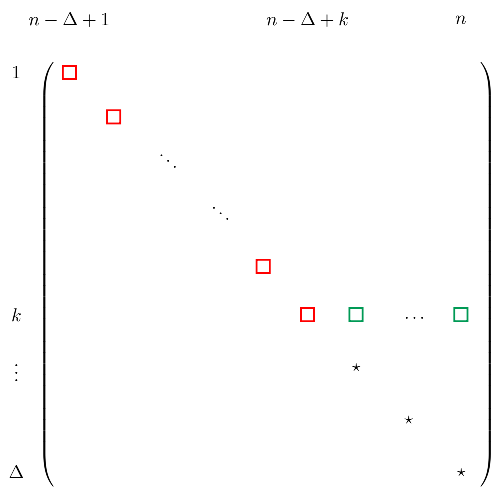
the set consisting of the row and column indices for the nonzero entries of the matrix B_n^Δ.

In the rest of the proof we indicate how to construct the set F. In view of (C4), the set F must contain all the pairs in E_n^Δ that contain the index k. Moreover, in view of (C5), F must contain all the diagonal pairs, except those coming before the position $(k+1-n+\Delta, k)$ on column k (if it exists) and those coming after the diagonal position $(k, n-\Delta+k)$ on row k (if it exists). Hence we must discuss depending whether the index k belongs to the sets U_R and/or U_C. Namely we consider the following four cases: (1) $k \notin U_R \cup U_C$, (2) $k \in U_R \setminus U_C$, (3) $k \in U_C \setminus U_R$, and (4) $k \in U_R \cap U_C$. In each of these cases, we define the set F which, by construction, will satisfy the conditions (C1), (C4) and (C5). Hence it will remain only to verify that condition (C2) holds in each of the four cases and this is what we do below.

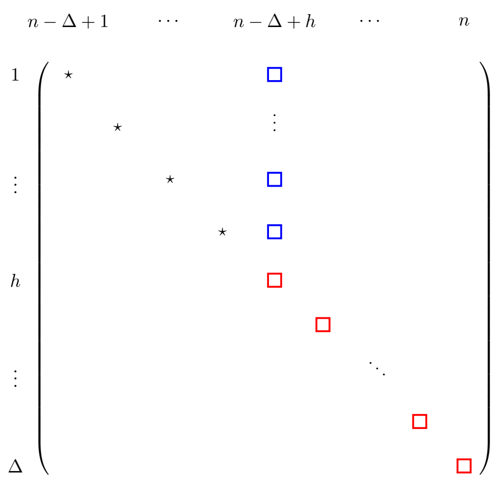
Case (1): $k \notin U_R \cup U_C$.
Then, $\Delta+1 \le k \le n-\Delta$, which implies $\Delta \le (n-1)/2$. In this case we define F



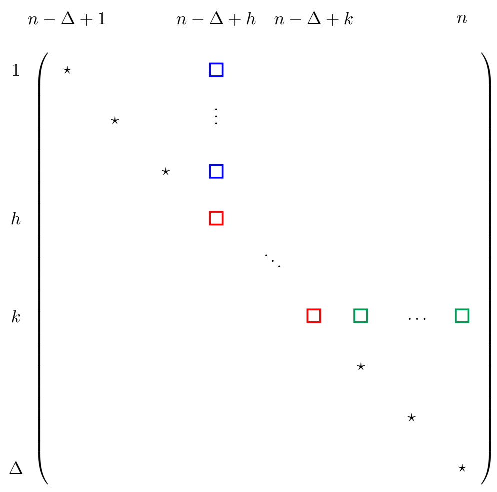
(a) Definition of set F for case (1).



(b) Definition of set F for case (2).



(c) Definition of set F for case (3).



(d) Definition of set F for case (4).

Figure 7.2: Different definition of set F for all 4 cases

as the set of all diagonal pairs (see Figure 7.2a at page 141), namely:

$$F = \{\{i, n-\Delta+i\} : 1 \le i \le \Delta\}.$$

To see that (C2) holds, let $r \neq s \in [\Delta]$; then $\min\{\pi(r), \pi(n-\Delta+r)\} \neq \min\{\pi(s), \pi(n-\Delta+s)\}$ holds. This is clear since the four indices $\pi(r)$, $\pi(n-\Delta+r)$, $\pi(s)$, and $\pi(n-\Delta+s)$ are pairwise distinct. Indeed, if equality $\pi(r) = \pi(n-\Delta+s)$ would hold, this would imply the inequalities: $n-\Delta+1 \le r = n-\Delta+s \le \Delta$ and thus $\Delta \ge (n+1)/2$, a contradiction.

Case (2): $k \in U_R \setminus U_C$.
Then, $1 \le k \le \Delta$ and $k \le n-\Delta$. In this case we let F consist of the diagonal

pairs till position $(k, n-\Delta+k)$ and then of the horizontal pairs on the k-th row (see Figure 7.2b at page 141), namely:

$$F = \{\{i, n-\Delta+i\} : 1 \le i \le k\} \cup \{\{k, n-\Delta+i\} : k+1 \le i \le \Delta\}.$$

In order to check that (C2) holds, we consider the following three cases:

- For $r \ne s \in [k]$, we get $\min\{\pi(r), \pi(n-\Delta+r)\} \ne \min\{\pi(s), \pi(n-\Delta+s)\}$, since the four indices $\pi(r), \pi(n-\Delta+r), \pi(s), \pi(n-\Delta+s)$ are pairwise distinct (using the fact that $k \le n-\Delta$).
- For $r \ne s \in \{k+1, \dots, \Delta\}$, using $\pi(k) = n$, $\min\{\pi(k), \pi(n-\Delta+r)\} = \pi(n-\Delta+r) \ne \min\{\pi(k), \pi(n-\Delta+s)\} = \pi(n-\Delta+s)$.
- Finally, for $r \in [k]$ and $s \in \{k+1, \dots, \Delta\}$, we have $\min\{\pi(r), \pi(n-\Delta+r)\} \ne \min\{\pi(k), \pi(n-\Delta+s)\} = \pi(n-\Delta+s)$. Indeed, $\pi(r) \ne \pi(n-\Delta+s)$ (since otherwise this would imply that $n-\Delta+k+1 \le r = n-\Delta+s \le k$ and thus $n+1 \le \Delta$, a contradiction) and it is clear that $\pi(n-\Delta+r) \ne \pi(n-\Delta+s)$. The case for $s \in [k]$ and $r \in \{k+1, \dots, \Delta\}$ is symmetric.

Case (3): $k \in U_C \setminus U_R$.
This case corresponds to $k = n-\Delta+h$, where $1 \le h \le \Delta$. Then, since k belongs to the column indices, we have that $n-\Delta+h = k \ge \Delta+1$, which implies $\Delta \le (n+h-1)/2$.

In this case we let F consists of the vertical pairs on the k-th column and of the diagonal pairs from position $(k, n-\Delta+k)$ (see Figure 7.2c at page 141), i.e,

$$F = \{\{i, k\} : 1 \le i \le h-1\} \cup \{\{i, n-\Delta+i\} : h \le i \le \Delta\}.$$

To see that (C2) holds, we consider the following three cases.

- For $r \ne s \in [h-1]$, $\pi(r) = \min\{\pi(r), \pi(k)\} \ne \min\{\pi(s), \pi(k)\} = \pi(s)$.
- For $r \ne s \in \{h, \dots, \Delta\}$, $\min\{\pi(r), \pi(n-\Delta+r)\} \ne \min\{\pi(s), \pi(n-\Delta+s)\}$, since the four indices $\pi(r), \pi(n-\Delta+r), \pi(s), \pi(n-\Delta+s)$ are pairwise distinct. Indeed, $\pi(r) = \pi(n-\Delta+s)$ would imply: $n-\Delta+h \le r = n-\Delta+s \le \Delta$ and thus $\Delta \ge (n+h)/2$, a contradiction.
- For $r \in [h-1]$ and $s \in \{h, \dots, \Delta\}$, we have that $\min\{\pi(r), \pi(k)\} = \pi(r) \ne \min\{\pi(s), \pi(n-\Delta+s)\}$, since the indices $\pi(r), \pi(s), \pi(n-\Delta+s)$ are pairwise distinct. Indeed, if $\pi(r) = \pi(n-\Delta+s)$, then we have that $n-\Delta+h \le r = n-\Delta+s \le h-1$ and thus $\Delta \ge n+1$, a contradiction.

Case (4): $k \in U_R \cap U_C = \{n-\Delta+1, \dots, \Delta\}$.
This case corresponds to $k = n-\Delta+h$, where $1 \le h \le 2\Delta-n$. Moreover, we have $\Delta \ge (n+1)/2$.

Then we let F consist of the vertical pairs on the k-th column, of the diagonal pairs from position $(h, n-\Delta+h)$ to position $(k, n-\Delta+k)$, and of the horizontal pairs on the k-th row (see Figure 7.2d at page 141), namely:

$$F = \{\{i,k\} : 1 \leq i \leq h\} \ \cup\ \{\{i, n-\Delta+i\} : h+1 \leq i \leq k\} \\ \cup \{\{k, n-\Delta+i\} : k+1 \leq i \leq \Delta\}.$$

In order to check condition (C2), as F consists of the union of three subsets we need to consider the following six cases.

- For $r \neq s \in [h]$, $\min\{\pi(r), \pi(k)\} = \pi(r) \neq \min\{\pi(s), \pi(k)\} = \pi(s)$.
- For $r \neq s \in \{h+1, \ldots, k\}$, we have that $\min\{\pi(r), \pi(n-\Delta+r)\} \neq \min\{\pi(s), \pi(n-\Delta+s)\}$, since $\pi(r), \pi(n-\Delta+r), \pi(s)$ and $\pi(n-\Delta+s)$ are pairwise distinct. Indeed, equality $\pi(r) = \pi(n-\Delta+s)$ would imply $r = n-\Delta+s$ and thus $k+1 = n-\Delta+h+1 \leq n-\Delta+s = r \leq k$, yielding a contradiction.
- For $r \neq s \in \{k+1, \ldots, \Delta\}$, it holds $\min\{\pi(k), \pi(n-\Delta+r)\} = \pi(n-\Delta+r) \neq \min\{\pi(k), \pi(n-\Delta+s)\} = \pi(n-\Delta+s)$.
- For $r \in [h]$ and $s \in \{h+1, \ldots, k\}$, we have that $\min\{\pi(r), \pi(k)\} = \pi(r) \neq \min\{\pi(s), \pi(n-\Delta+s)\}$, since the indices $\pi(r), \pi(s), \pi(n-\Delta+s)$ are pairwise distinct. Indeed, $\pi(r) = \pi(n-\Delta+s)$ would imply that $n-\Delta+h+1 \leq n-\Delta+s = r \leq h$ and thus $\Delta \geq n+1$, a contradiction.
- For $r \in [h]$ and $s \in \{k+1, \ldots, \Delta\}$, then we have that $\min\{\pi(r), \pi(k)\} = \pi(r) \neq \min\{\pi(k), \pi(n-\Delta+s)\} = \pi(n-\Delta+s)$, since the two indices $\pi(r), \pi(n-\Delta+s)$ are distinct. Indeed, equality $\pi(r) = \pi(n-\Delta+s)$ would imply that $r = n-\Delta+s$ and thus $2(n-\Delta)+h+1 = n-\Delta+k+1 \leq n-\Delta+s = r \leq h$, which implies $\Delta \geq n+1$, a contradiction.
- For $r \in \{h+1, \ldots, k\}$ and $s \in \{k+1, \ldots, \Delta\}$, then $\min\{\pi(r), \pi(n-\Delta+r)\} \neq \min\{\pi(k), \pi(n-\Delta+s)\} = \pi(n-\Delta+s)$. Indeed, $r = n-\Delta+s$ would imply that $n-\Delta+k+1 \leq n-\Delta+s = r \leq k$, which implies $\Delta \geq n+1$, a contradiction.

Thus (C2) holds, which concludes the proof in case (4) and thus the proof of the theorem.

7.2.4 Applications of the main result

We now formulate several applications of our main result in Theorem 7.2.2. As a first direct consequence, we can show that the identity matrix is an optimal solution for QAP(A, B) whenever A is a Robinson similarity matrix, B is a Robinson dissimilarity matrix, and at least one of A or B is a Toeplitz matrix.

7.2.5 Theorem. *Let $A, B \in \mathcal{S}^n$ and assume that A is a Robinson similarity matrix, B is a Robinson dissimilarity matrix and moreover A or B is a Toeplitz matrix. Then the identity permutation is an optimal solution for* QAP(A, B).

Proof. Assume first that B is a Toeplitz matrix. Then, by Lemma 7.2.1, B is a conic combination of the matrices B_n^Δ, i.e., $B = \sum_{\Delta=1}^{n-1} \lambda_\Delta B_n^\Delta$ for some scalars $\lambda_\Delta \geq 0$. Applying the inequality (7.5) in Theorem 7.2.2, we obtain that

$$\langle A_\pi, B\rangle = \sum_{\Delta=1}^{n-1} \lambda_\Delta \langle A_\pi, B_n^\Delta\rangle \geq \sum_{\Delta=1}^{n-1} \lambda_\Delta \langle A, B_n^\Delta\rangle = \langle A, B\rangle,$$

which shows that the identity permutation is optimal for QAP(A, B).

Assume now that A is a Toeplitz Robinson similarity (and B is a Robinson dissimilarity). Then we simply exchange the roles of A and B after a simple modification. Namely, let α and β denote the maximum value of the entries of A and B, respectively. Let us define the two matrices $B' = \alpha J - A$ and $A' = \beta J - B$. Then A' is a Robinson similarity matrix and B' is a Toeplitz Robinson dissimilarity matrix. For any permutation π, by the previous result applied to QAP(A', B'), $\langle (A')_\pi, (B')\rangle \geq \langle A', B'\rangle$. If we compute the inner product of both sides, we obtain $\langle (A')_\pi, B'\rangle = \alpha\beta\langle J_\pi, J\rangle - \alpha\langle J_\pi, A\rangle - \beta\langle B_\pi, J\rangle + \langle B_\pi, A\rangle$ and $\langle A', B'\rangle = \alpha\beta\langle J, J\rangle - \alpha\langle J, A\rangle - \beta\langle B, J\rangle + \langle B, A\rangle$, from which we can easily conclude that $\langle A, B_\pi\rangle \geq \langle A, B\rangle$. Hence, this shows that the identity permutation is optimal for QAP(A, B) and it concludes the proof. $\square$

As a direct application, Theorem 7.2.5 extends to the case when the matrices A and B are Robinsonian.

7.2.6 Corollary. *Let $A, B \in \mathcal{S}^n$. Assume that A is a Robinsonian similarity matrix, B is a Robinsonian dissimilarity matrix, and let π, τ be permutations that reorder A and B as Robinson similarity and dissimilarity matrices, respectively. Assume furthermore that one of the matrices A_π or B_τ is a Toeplitz matrix. Then the permutation $\tau^{-1}\pi$ is optimal for* QAP(A, B).

Proof. Directly from Theorem 7.2.5, using relation (2.1). $\square$

Finally we observe that the assumption that either A or B has a Toeplitz structure cannot be omitted in Theorem 7.2.2. Indeed, consider the following matrices:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

so that A (resp., B) is a Robinson similarity (resp., dissimilarity). In this case, the identity permutation gives a solution of value $\langle A, B\rangle = 8$. Consider now the permutation $\pi = (4, 5, 1, 2, 3)$ which reorders A as follows:

$$A_\pi = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

This gives a solution of value $\langle A_\pi, B\rangle = 4$. Hence, in this case the identity permutation is not optimal, and thus the Toeplitz assumption cannot be removed in Theorem 7.2.2.

Our result can be seen as the analog for symmetric matrices of the above mentioned result about QAP for monotone matrices (see Subsection 7.2.1), where we replace the monotonicity property by the Robinson property (which implies unimodal rows and columns).

Moreover, our result extends two previously known cases. The first case is when the Robinson similarity A is a conic combination of interval-cut matrices and the Robinson dissimilarity B is the Toeplitz matrix $B = ((i-j)^2)_{i,j=1}^n$, discussed above in Theorem 7.1.1 and considered in [56].

The second case is when the similarity matrix A is the adjacency matrix of the path $(1, 2 \dots, n-1, n)$, which is Toeplitz, and B is a special Robinsonian dissimilarity matrix which is metric and strongly monotone (i.e., $B_{jk} = B_{jl}$ implies $B_{ik} = B_{il}$ and $B_{jk} = B_{ik}$ implies $B_{jl} = B_{il}$, for all $1 \leq i < j < k < l \leq n$.) Then, QAP(A,B) corresponds to solving the minimum weight Hamiltonian path [1] problem on the undirected complete weighted graph represented by B, and it is optimally solved by a Robinson ordering of B [31, Lemma 10]. Interestingly the above class of strongly monotone Robinson dissimilarity metrics plays a central role in the recognition algorithm of [31] for matrices that can be permuted to Kalmanson matrices.

7.3 Conclusions and future work

In this chapter we discussed how to model the seriation problem as a special instance of QAP(A, B), where both A and B are Robinson(ian) matrices and at least one of the two has a Toeplitz structure. We have then shown that Robinson orderings of A and B lead to an optimal solution for QAP(A, B). This result highlights the importance of the Robinsonian recognition algorithms discussed in Chapters 5 and 6.

[1] A Hamiltonian path of a graph is a path visiting each vertex of the graph exactly once.

Furthermore, it motivates our interest in approximating the Robinsonian structure when dealing with seriation, which will be discussed in Chapter 8.

An important open question is what is the complexity of QAP(A, B) when keeping the Robinsonian assumption on both A and B and removing the Toeplitz assumption in Theorem 7.2.5. As we have seen in the example above, in fact, the identity permutation is in general not optimal anymore.

8

Robinsonian matrix approximation

In this chapter we discuss how to solve the seriation problem by finding a ‘close’ Robinsonian approximation of the original (non-Robinsonian) similarity matrix. In Section 8.1 we give a short overview of the two main approaches to find such an approximation. In Section 8.2 we define the l_∞-FITTING-BY-ROBINSONIAN problem and the ϵ-ROBINSONIAN-RECOGNITION problem. Then, in Section 8.3 we introduce the ϵ-Similarity-First Search algorithm (ϵ-SFS), an extension of the SFS algorithm presented in Section 6, and we discuss a multisweep algorithm to recognize ϵ-Robinsonian matrices based on ϵ-SFS. Finally, in Section 8.4 we conclude with some possible directions for future work.

8.1 Combinatorial aspects

In Chapter 7 we have discussed how to model the seriation problem as an instance of the Quadratic Assignment Problem (QAP). In this chapter, we discuss a different approach. The main difference with respect to the approach discussed in Chapter 7 is that we try here to approximate the Robinsonian property as follows: given a similarity matrix $A \in \mathcal{S}^n$, the idea is to find another matrix $A^R \in \mathcal{S}^k$ with $k \leq n$ which is a ‘Robinsonian approximation’ of A. There exist two main complementary variants in the literature to define such an approximation matrix. We briefly discuss them providing some references. In both cases, we are given as input a similarity matrix $A \in \mathcal{S}^n$.

The Robinsonian submatrix problem The problem consists in finding the minimum cardinality subset $S \subseteq [n]$ whose deletion makes A a Robinsonian matrix or, equivalently, in finding the maximum cardinality subset $S \subseteq [n]$ such that $A[S]$ is a Robinsonian matrix. This approach is used when the Robinsonian property is obstructed by few outlier elements (e.g., because of error measurement), and it is a well known practice in computational biology to detect inconsistencies in experimental data [95]. The approximation of the new matrix $A^R = A[S]$ with respect to the original similarity information A is evaluated in terms of the number of removals. This problem has been studied for unit interval graphs and the consecutive ones problem (see e.g. [48, 12] and references therein). Specifically, the problem of finding the largest consecutive ones submatrix is NP-complete [64] as well as the UNIT INTERVAL VERTEX DELETION problem [80]. Given the equivalence between unit interval graphs and 0/1 Robinsonian matrices (see Subsection 3.2.1), also the Robinsonian submatrix problem is a hard problem.

The 'Fitting by a Robinsonian matrix' problem This problem asks to find a Robinsonian matrix $A^R \in \mathcal{S}^n$ which is 'close' to a given proximity matrix A. There exist different ways to define such a degree of 'closeness'. A classic approach is, for example, to use the Euclidean distance between the two matrices. The topic of fitting a given proximity matrix by some structured matrix is a wide topic in the field of multivariate data analysis. We thus refer the interested reader to the book [70] and references therein for an exhaustive review on the topic. In this paragraph we simply remark that the problem of fitting proximity matrices by Robinsonian matrices has been investigated in the past years. One of the reasons why the 'fitting by a Robinsonian matrix' problem has been more studied than the 'Robinsonian submatrix problem' is due to the relation between Robinsonian (dissimilarities) matrices and ultrametrics, which are used in taxonomy and phylogenetic tree construction (see Subsection 3.1.2). Specifically, Hubert and Arabie [68] investigated how to approximate proximity matrices by sums of Robinsonian matrices using least-squares minimization, and Hubert *et al.* [70] focused on the approximation by sums of strongly Robinsonian matrices. Other relevant related results can be found in [54, 8, 29, 2, 17, 63].

In this chapter we discuss in more detail the problem of fitting a similarity matrix by a Robinsonian matrix. The motivation for this choice is twofold. First of all, we have seen in Chapter 7 that if A is Robinsonian then any Robinson ordering is an optimal solution of the seriation problem when modeled as a special subclass of QAP. Therefore, intuitively, if A is not Robinsonian, the permutation reordering a 'close' Robinsonian approximation of the original similarity matrix could lead to a 'good' solution for the original seriation problem. Second, differently from the Robinsonian submatrix problem, the size of the input matrix is not modified. Hence, comparing the solution returned by the fitting problem and the quadratic program in Section 7 can be easily done by evaluating, e.g., the objective function of the QAP at the permutation of the elements of A returned by the two approaches.

The results in this section are inspired by the papers [29, 30], as our work in Chapter 6 can be naturally extended to their fitting model. Again, we treat Robinsonian similarity matrices. Hence, when talking about Robinson(ian) matrices, we mean Robinson(ian) similarities unless explicitly specified.

8.2 Fitting Robinsonian matrices

The problem of fitting a given proximity matrix $A \in \mathcal{S}^n$ consists in finding another equally sized matrix $A' \in \mathcal{S}^n$ which is 'close' to A. Standard approaches to define the distance between two matrices are, for example, to use the l_p-norm as error measure for some $0 < p < \infty$, i.e.,:

$$\|A - A'\|_p = \left(\sum_{x,y \in [n]} |A_{xy} - A'_{xy}|^p \right)^{\frac{1}{p}}. \tag{8.1}$$

In this case, the distance measure takes into account the sum of the differences among the entries of the two matrices A and A'. We are interested in the special case when the distance error is defined using the l_∞-norm, i.e.,:

$$\|A - A'\|_\infty = \max_{x,y,\in[n]} \{|A_{xy} - A'_{xy}|\}. \tag{8.2}$$

In other words, the distance between the two matrices corresponds to the largest difference between their entries. For the sake of ease, we will use the following notation: when we write $A \geq B$ for some $A, B \in \mathcal{S}^n$, we mean that the inequality holds for each pair of indices, i.e., $A_{xy} \geq B_{xy}$ for each $x, y \in [n]$. Hence, when we write $A \in [A - \epsilon, A + \epsilon]$ or $A - \epsilon \leq A \leq A + \epsilon$ for some $\epsilon \geq 0$, we mean, respectively, $A_{xy} \in [A_{xy} - \epsilon, A_{xy} + \epsilon]$ and $A_{xy} - \epsilon \leq A_{xy} \leq A_{xy} + \epsilon$ for each $x, y \in [n]$. We will use the indices only when the relation does not hold for all pairs of vertices.

8.2.1 l_∞-fitting Robinsonian matrices

The problem of finding the closest Robinsonian similarity to a given matrix A can be modeled as an optimization problem as follows.

l_∞-FITTING-BY-ROBINSONIAN problem Given a matrix $A \in \mathcal{S}^n$ find a *Robinsonian* similarity matrix $A^R \in \mathcal{S}^n$ closest to A for the ℓ_∞-norm, i.e.,

$$\epsilon_{opt} = \min_{\substack{A^R \in \mathcal{S}^n \\ \text{Robinsonian}}} \|A - A^R\|_\infty. \tag{8.3}$$

Given a matrix A, we will refer to A_{xy} as the nominal value of A for $x, y \in [n]$. Let us denote by ϵ_{opt} the optimal of the program (8.3). Then we have that $A^R \in [A - \epsilon_{opt}, A + \epsilon_{opt}]$, so A^R is a matrix obtained by perturbing each entry of A by at most ϵ_{opt}.

Chepoi *et al.* [29] showed that l_∞-FITTING-BY-ROBINSONIAN (8.3) is an NP-hard problem. This answers an open question of Barthélemy and Brucker [8], who earlier established the NP-hardness of the problem of finding an optimal approximation for the l_p-norm (8.1) of a similarity matrix by a strongly Robinsonian matrix, for $0 < p < \infty$. The complete result in [29] is given below.

8.2.1 Theorem. *[29] The l_∞-FITTING-BY-ROBINSONIAN problem (8.3) is NP-hard to approximate within $\frac{3}{2} - \delta$ for any $\delta > 0$.*

Chepoi and Seston [30] recently introduced a factor 16-approximation algorithm [1] for l_∞-FITTING-BY-ROBINSONIAN (8.3).

On the other hand, the variant of problem (8.3) where we ask to find a Robinson similarity matrix closest to A for the ℓ_∞-norm, is easy to solve and admits in fact an explicit solution. Consider the following problem.

l_∞-FITTING-BY-ROBINSON problem Given a similarity matrix $A \in \mathcal{S}^n$, find a *Robinson* similarity matrix $A^* \in \mathcal{S}^n$ closest to A for the ℓ_∞-norm, i.e.,

$$\epsilon^* = \min_{\substack{A^* \in \mathcal{S}^n \\ \text{Robinson}}} \|A - A^*\|_\infty. \tag{8.4}$$

Hence, one can see l_∞-FITTING-BY-ROBINSON (8.4) as a special case of the l_∞-FITTING-BY-ROBINSONIAN problem (8.3) restricted to a fixed permutation π of $[n]$.

We now show that the solution of l_∞-FITTING-BY-ROBINSON (8.2) can be easily computed in a closed form. Let us fix a linear order π of the index set of A (e.g., the natural ordering). We then introduce two special matrices A^+ and A^-, earlier introduced in [29] for dissimilarities. Both matrices are ordered according to the same linear order π as A, and their entries are defined as follows:

$$A^-_{xy} = \min\{A_{uv} : u, v \in [n], x \le u < v \le y\} \qquad \forall x, y \in [n], \tag{8.5}$$

$$A^+_{xy} = \max\{A_{uv} : u, v \in [n], u \le x < y \le v\} \qquad \forall x, y \in [n]. \tag{8.6}$$

In addition to the original definition in [29] given above, it will be useful to also consider the following equivalent recursive definition of A^+ and A^-.

[1]Recall that an algorithm for a minimization problem Π is called an *α-approximation algorithm* if for any instance I of Π it returns a solution whose value is at most α times the optimal value $\mathrm{OPT}_\Pi(I)$ of Π on I.

8.2.2 Lemma. *Let $A \in \mathcal{S}^n$. Then, the following holds for each $x < y \in [n]$:*

$$A^-_{xy} = \min \begin{cases} A_{xy}, \\ A^-_{x,(y-1)}, \\ A^-_{(x+1),y}, \end{cases} \tag{8.7}$$

$$A^+_{xy} = \max \begin{cases} A_{xy}, \\ A^+_{x,(y+1)}, \\ A^+_{(x-1),y}, \end{cases} \tag{8.8}$$

where undefined and diagonal entries are ignored.

Proof. Use induction on the distance between indices x and y. $\square$

The reason to introduce the above equivalent formulation is that A^+ and A^- are easier to compute using (8.7) and (8.8) than using the formulas in (8.5) and (8.6). Namely, one can compute the entries of A^+ by 'propagation': starting at the upper right corner and passing through the first row, then passing through the second row from right to left, and so on. Analogously for A^-, starting at the upper diagonal left corner and passing through the first upper diagonal, then passing through the second upper diagonal from top to bottom and so on. Furthermore, using (8.7) and (8.8), it is immediate to derive some other important properties of A^+ and A^-, which are already discussed in [29] and whose alternative proof is stated below.

8.2.3 Lemma. *Let $A \in \mathcal{S}^n$ and consider the matrices A^- as in (8.5) and A^+ as in (8.6). Then the following holds:*

(i) A^+ and A^- are Robinson similarities,

(ii) $A^- \leq A \leq A^+$.

Furthermore, let B be a Robinson similarity. Then:

(iii) if $B \leq A$ then $B \leq A^-$,

(iv) if $A \leq B$ then $A^+ \leq B$.

Proof. (i) and (ii) are immediately proved using the recursive definition of A^- and A^+ in (8.7) and (8.8). (iii) By definition of A^- in (8.5) we have that $A^-_{xy} = A_{uv}$ for some $x \leq_\pi u <_\pi v \leq_\pi y$. Using the fact that B is a Robinson similarity and $B \leq A$, we get the relations $B_{xy} \leq B_{uv} \leq A_{uv} = A^-_{xy}$, which shows indeed that $B \leq A^-$. (iv) By definition of A^+ in (8.6) we have that $A^+_{xy} = A_{uv}$ for some $u \leq_\pi x <_\pi y \leq_\pi v$. Using the fact that $A \leq B$ and B is Robinson, we obtain the relations $A^+_{xy} = A_{uv} \leq B_{uv} \leq B_{xy}$, for some $u \leq_\pi x <_\pi y \leq_\pi v$, which shows indeed that $A^+ \leq B$. $\square$

As a concrete example, consider the matrix A given below.

$$A = \begin{array}{c|cccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \hline \mathbf{1} & * & 8 & 6 & 7 & 5 & 0 \\ \mathbf{2} & & * & 22 & 15 & 14 & 11 \\ \mathbf{3} & & & * & 20 & 16 & 9 \\ \mathbf{4} & & & & * & 21 & 12 \\ \mathbf{5} & & & & & * & 13 \\ \mathbf{6} & & & & & & * \end{array} \tag{8.9}$$

which is not Robinson. Then the corresponding matrices A^- and A^+ are the following:

$$A^- = \begin{array}{c|cccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \hline \mathbf{1} & * & 8 & 6 & 6 & 5 & 0 \\ \mathbf{2} & & * & 22 & 15 & 14 & 9 \\ \mathbf{3} & & & * & 20 & 16 & 9 \\ \mathbf{4} & & & & * & 21 & 12 \\ \mathbf{5} & & & & & * & 13 \\ \mathbf{6} & & & & & & * \end{array}, \quad A^+ = \begin{array}{c|cccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \hline \mathbf{1} & * & 8 & 7 & 7 & 5 & 0 \\ \mathbf{2} & & * & 22 & 15 & 14 & 11 \\ \mathbf{3} & & & * & 20 & 16 & 11 \\ \mathbf{4} & & & & * & 21 & 12 \\ \mathbf{5} & & & & & * & 13 \\ \mathbf{6} & & & & & & * \end{array}.$$

Hence, A^- and A^+ represent intuitively the lower and upper Robinson approximations of A, respectively (this is why they are denoted by '$-$' and '$+$'). We now present the closed form formula for the solution of l_∞-FITTING-BY-ROBINSON (8.4). Consider the following matrix:

$$A^* := \frac{A^- + A^+}{2}, \tag{8.10}$$

which is the 'average' of A^- and A^+. Since in view of Lemma 8.2.3 both A^- and A^+ are Robinson similarities, also A^* is Robinson. As we show below, A^* is not only Robinson, but it is in fact the closest Robinson approximation of A. This result is the analog of [30, Lemma 2.4] for dissimilarities, although our proof here is a bit simpler.

8.2.4 Theorem. *Let $A \in \mathcal{S}^n$, let A^* be as in (8.10), and let ϵ^* be the optimal value of the l_∞-FITTING-BY-ROBINSON problem (8.4). Then A^* is optimal for l_∞-FITTING-BY-ROBINSON (8.4). Furthermore, $\|A - A^-\|_\infty = \|A - A^+\|_\infty = \|A^+ - A^-\|_\infty = 2\epsilon^*$.*

Proof. Let $\hat{A}$ be an optimal solution of l_∞-FITTING-BY-ROBINSON (8.4). Then $\epsilon^* = \|A - \hat{A}\|_\infty$ and thus the following inequalities hold.

$$-\epsilon^* \leq A - \hat{A} \leq \epsilon^*,$$

which we can rewrite equivalently as:

$$\hat{A} - \epsilon^* \leq A \leq \hat{A} + \epsilon^*. \tag{8.11}$$

Since $\hat{A}$ is Robinson, then $\hat{A} \pm \epsilon^*$ is also Robinson. Hence, consider $B = \hat{A} - \epsilon^*$, which is Robinson. Then, using (8.11) we get $B \leq A$, and in view of Lemma 8.2.3 we have $B = \hat{A} - \epsilon^* \leq A^-$. Analogously, consider now $B = \hat{A} + \epsilon^*$, which is also Robinson. Then, using (8.11) we get $A \leq B$, and in view of Lemma 8.2.3 we have $A^+ \leq \hat{A} + \epsilon^* = B$. We then get the following chain of inequalities:

$$\hat{A} - \epsilon^* \leq A^- \leq A \leq A^+ \leq \hat{A} + \epsilon^*. \tag{8.12}$$

On the other hand, using the definition of A^* in (8.10), we have:

$$A - A^* = A - \frac{A^+ + A^-}{2} = \frac{1}{2}\left(A - A^+\right) + \frac{1}{2}\left(A - A^-\right).$$

Then, in view of Lemma 8.2.3 we have that on the right-hand side $(A - A^+) \leq 0$ while $(A - A^-) \geq 0$. Therefore the following inequalities hold:

$$\frac{1}{2}\left(A - A^+\right) \leq A - A^* \leq \frac{1}{2}\left(A - A^-\right).$$

Finally, using the above inequalities and (8.12) we get:

$$\frac{1}{2}\left(A - \hat{A} - \epsilon^*\right) \leq \frac{1}{2}\left(A - A^+\right) \leq A - A^* \leq \frac{1}{2}\left(A - A^-\right) \leq \frac{1}{2}\left(A - \hat{A} + \epsilon^*\right).$$

Since $\epsilon^* \leq A - \hat{A} \leq \epsilon^*$, we get that:

$$-\epsilon^* \leq \frac{1}{2}\left(A - A^+\right) \leq A - A^* \leq \frac{1}{2}\left(A - A^-\right) \leq \epsilon^*. \tag{8.13}$$

This implies: $\|A-A^*\|_\infty \leq \epsilon^*$, $\|A-A^-\|_\infty \leq 2\epsilon^*$, $\|A-A^+\|_\infty \leq 2\epsilon^*$. Using (8.12), we have $A^+ - A^- \leq 2\epsilon^*$, which implies $\|A^+ - A^-\|_\infty \leq 2\epsilon^*$. Since ϵ^* is the optimal value of l_∞-FITTING-BY-ROBINSON (8.4) and A^* is Robinson, then equality $\|A - A^*\|_\infty = \epsilon^*$ must hold. Hence, there exist distinct $x, y \in [n]$ such that $|A_{xy} - A^*_{xy}| = \epsilon^*$. Say $A_{xy} - A^*_{xy} = \epsilon^*$ to fix ideas (the case $A_{xy} - A^*_{xy} = -\epsilon^*$ is analogous). Then equality holds in the two rightmost inequalities in (8.13), i.e.,:

$$A_{xy} - A^*_{xy} = \frac{1}{2}(A_{xy} - A^-_{xy}) = \epsilon^*,$$

which in turn implies $A^+_{xy} - A^-_{xy} = A_{xy} - A^-_{xy} = 2\epsilon^*$. This already gives the equalities $\|A - A^-\|_\infty = 2\epsilon^*$ and $\|A^+ - A^-\|_\infty = 2\epsilon^*$. We finally show that also $\|A^+ - A\|_\infty = 2\epsilon^*$ holds. Using definition (8.5) for A^-, there exist u, v such that $x \leq u < v \leq y$ and $A^-_{xy} = A_{uv} = A_{xy} - 2\epsilon^*$. Using now the definition (8.6) for A^+_{uv}, we deduce that $A^+_{uv} \geq A_{xy} = A^-_{xy} + 2\epsilon^* = A_{uv} + 2\epsilon^*$ and thus $A^+_{uv} - A_{uv} \geq 2\epsilon^*$, which implies equality $\|A^+ - A\|_\infty = 2\epsilon^*$. This concludes the proof. □

There are two substantial differences with respect to the result in [30] above mentioned. First, our optimal solution A^* for l_∞-FITTING-BY-ROBINSON (8.4) is easier to compute using the recursive formulas (8.7) and (8.8). Second, A^* has fewer modified entries than if we would use the corresponding dissimilarity in [30].

Just to give an idea, we give below our optimal solution for l_∞-FITTING-BY-ROBINSON (8.4), denoted by A^*, and the optimal solution for l_∞-FITTING-BY-ROBINSON (8.4) as described in [30], denoted by $\tilde{A}$.

$$A^* = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{array} \begin{array}{c} \begin{array}{cccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{array} \\ \begin{pmatrix} * & 8 & 6.5 & 6.5 & 5 & 0 \\ & * & 22 & 15 & 14 & 10 \\ & & * & 20 & 16 & 10 \\ & & & * & 21 & 12 \\ & & & & * & 13 \\ & & & & & * \end{pmatrix} \end{array}, \quad \tilde{A} = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \end{array} \begin{array}{c} \begin{array}{cccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{array} \\ \begin{pmatrix} * & 7 & 6 & 6 & 4 & 0 \\ & * & 21 & 14 & 13 & 10 \\ & & * & 19 & 15 & 10 \\ & & & * & 20 & 11 \\ & & & & * & 12 \\ & & & & & * \end{pmatrix} \end{array}.$$

Note that in both cases the largest difference with respect to the matrix in (8.9) is equal to $\epsilon^* = 1$. However, looking for the smallest number of modifications to build a Robinson matrix can indeed be relevant in understanding how far is a matrix from being Robinson.

To recap, the l_∞-FITTING-BY-ROBINSON problem (8.4) is easily solvable by the matrix defined in (8.10). In other words, for any *fixed* permutation π of $[n]$, the l_∞-FITTING-BY-ROBINSONIAN problem (8.3) becomes easy to solve. The hardness arises from the fact that we need to choose also the permutation π. Indeed, l_∞-FITTING-BY-ROBINSONIAN (8.3) is equivalent to solve $n!$ times the l_∞-FITTING-BY-ROBINSON problem (8.4) and then choose the permutation leading to the best error.

8.2.2 ϵ-Robinsonian recognition

As mentioned in [30], one can rephrase the l_∞-FITTING-BY-ROBINSONIAN problem (8.3) as a combinatorial problem by relaxing the notions of Robinsonian matrix and of Robinson ordering. Specifically, given a symmetric matrix $A \in \mathcal{S}^n$ and a real number $\epsilon \geq 0$, we introduce the notions of *ϵ-Robinsonian* matrix and of *ϵ-Robinson ordering.*

8.2.5 Definition. (ϵ-Robinsonian) Given $A \in \mathcal{S}^n$ and a real number $\epsilon \geq 0$, A is *ϵ-Robinsonian* if there exists a permutation π of $[n]$ such that:

$$A_{xz} \leq \min\{A_{xy}, A_{yz}\} + \epsilon \qquad \forall\, x <_\pi y <_\pi z. \tag{8.14}$$

We then call such a permutation π an *ϵ-Robinson ordering* of A.

Clearly, if A is Robinsonian then the inequality (8.14) is always satisfied. Furthermore, if condition (8.14) holds, then the following four-points condition (8.15) also holds.

8.2.6 Lemma. *Let $A \in \mathcal{S}^n$ be a similarity matrix, $\epsilon \geq 0$ be a real number. If a linear order π of $[n]$ is an ϵ-Robinson ordering of A then:*

$$A_{uv} \leq A_{xy} + 2\epsilon \qquad \forall\ u \leq_\pi x <_\pi y \leq_\pi v. \tag{8.15}$$

Proof. Applying condition (8.14) to the triples (u, y, v) and (u, x, y) we have that $A_{uv} \leq A_{uy} + \epsilon \leq (A_{xy} + \epsilon) + \epsilon = A_{xy} + 2\epsilon$. □

We now establish a relationship between ϵ-Robinsonian matrices and the l_∞-FITTING-BY-ROBINSONIAN problem (8.3).

8.2.7 Lemma. *Let $A \in \mathcal{S}^n$, $\epsilon \geq 0$ and let π be an ordering of $[n]$. Consider the matrix $(A_\pi)^*$ defined by the condition (8.10) applied to A_π and the following statements:*

(1) π is a ϵ-Robinson ordering of A.

(2) $\|A_\pi - (A_\pi)^*\|_\infty \leq \epsilon$,

(3) The matrix $((A_\pi)^*)_{\pi^{-1}}$ lies in $[A - \epsilon, A + \epsilon]$.

(4) there exists a Robinsonian matrix $A^R \in [A - \epsilon, A + \epsilon]$ admitting π as a Robinson ordering.

(5) π is a 2ϵ-Robinson ordering of A.

Then (1) ⇒ (2) ⇒ (3) ⇒ (4) ⇒ (5).

Proof. (1) ⇒ (2): To simplify notation, assume that π is the natural order and set $\epsilon^* = \|A - A^*\|_\infty$. Assume for contradiction that $\epsilon^* > \epsilon$. Consider $x, y \in [n]$ such that $\epsilon^* = |A_{uv} - A^*_{uv}|$ and, to fix ideas, suppose $\epsilon^* = A_{uv} - A^*_{uv}$ (the case $\epsilon^* = A^*_{uv} - A_{uv}$ is analogous). Using the fact that $A \leq A^+$ in view of Lemma 8.2.3 and the definition of A^* in (8.10), we obtain:

$$\epsilon^* = A_{uv} - A^*_{uv} = \frac{A_{uv} - A^+_{uv}}{2} + \frac{A_{uv} - A^-_{uv}}{2} \leq \frac{A_{uv} - A^-_{uv}}{2},$$

and thus $A_{uv} - A^-_{uv} \geq 2\epsilon^* > 2\epsilon$. Using the definition (8.5) for A^-, there exists $x, y \in [n]$such that $u \leq x < y \leq v$ and $A^-_{uv} = A_{xy}$. This gives $A_{uv} > A_{xy} + 2\epsilon$, which contradicts (8.15) and thus, in view of Lemma 8.2.6, the assumption that π is a ϵ-Robinson ordering of A.

(2) ⇒ (3): By assumption, $(A_\pi)^*$ lies in $[A_\pi - \epsilon, A_\pi + \epsilon]$ and thus $((A_\pi)^*)_{\pi^{-1}}$ lies in $[A - \epsilon, A + \epsilon]$.

(3) ⇒ (4): We may choose $A^R = ((A_\pi)^*)_{\pi^{-1}}$, since π is a Robinson ordering of A^R, as $(A_\pi)^*$ is a Robinson matrix by definition.

(4) $\Rightarrow$ (5): Assume π is a Robinson ordering of $A^R \in [A-\epsilon, A+\epsilon]$. Then $A^R_{xz} \leq \min\{A^R_{xy}, A^R_{yz}\}$ for all $x <_\pi y <_\pi z$. Since $A^R \in [A-\epsilon, A+\epsilon]$ we get that:

$$A_{xz} - \epsilon \leq A^R_{xz} \leq \min\{A^R_{xy}, A^R_{yz}\} \leq \min\{A_{xy}, A_{yz}\} + \epsilon.$$

This implies $A_{xz} \leq \min\{A_{xy}, A_{yz}\} + 2\epsilon$, thus stating that π is a 2ϵ-Robinson ordering of A. $\square$

The l_∞-FITTING-BY-ROBINSONIAN problem (8.3) is then closely related to the problem of finding the smallest ϵ such that A is ϵ-Robinsonian, as stated by the next corollaries.

8.2.8 Corollary. *Let $A \in \mathcal{S}^n$ and $\epsilon \geq 0$. Consider the following statements:*

(1) A is ϵ-Robinsonian,

(2) there exists a Robinsonian matrix $A^R \in [A-\epsilon, A+\epsilon]$,

(3) A is 2ϵ-Robinsonian.

Then (1) $\Rightarrow$ (2) $\Rightarrow$ (3).

Proof. Direct application of Lemma 8.2.7. $\square$

8.2.9 Corollary. *Let $A \in \mathcal{S}^n$ and let ϵ_{opt} be the optimal value of l_∞-FITTING-BY-ROBINSONIAN (8.3). Let $\epsilon \geq 0$ the smallest real number such that A is ϵ-Robinsonian. Then $\epsilon_{opt} \leq \epsilon \leq 2\epsilon_{opt}$.*

Proof. Direct application of Corollary 8.2.8. $\square$

Hence, roughly speaking, finding the smallest ϵ such that A is ϵ-Robinsonian produces bounds on l_∞-FITTING-BY-ROBINSONIAN (8.3). To find such ϵ, it can be useful to recognize ϵ-Robinsonian matrices, i.e., answering the following decision problem.

ϵ-ROBINSONIAN-RECOGNITION problem Given a similarity matrix $A \in \mathcal{S}^n$ and a real number $\epsilon > 0$, is A ϵ-Robinsonian?

Chepoi *et al.* [29] show that the ϵ-ROBINSONIAN-RECOGNITION problem is NP-complete. We now show that the optimal value ϵ_{opt} of FITTING-BY-ROBINSONIAN (8.3) is contained in a finite set which depends on the values of the entries of the similarity matrix A. The result given below is an alternative proof of [29, Lemma 4.5].

8.2.10 Lemma. *Let $A \in \mathcal{S}^n$ and let ϵ_{opt} be the optimal value of l_∞-FITTING-BY-ROBINSONIAN (8.3). Then $\epsilon_{opt} \in \Delta := \{|A_{xy} - A_{uz}|/2 : u, x, y, z \in [n]\}$.*

Proof. Let A^R be an optimal solution of l_∞-FITTING-BY-ROBINSONIAN (8.3) and let π be a Robinson ordering of A^R. We claim that $(A^R)_\pi$ is a Robinson matrix which is closest to A_π. Indeed:

$$\epsilon_{opt} = \|A - A^R\|_\infty = \|A_\pi - (A^R)_\pi\|_\infty \geq \|A_\pi - (A_\pi)^*\|_\infty = \|A - ((A_\pi)^*)_{\pi^{-1}}\|_\infty \geq \epsilon_{opt}$$

where for the first inequality we used the fact that $(A_\pi)^*$ is a Robinson matrix closest to A_π (Theorem 8.2.4) and for the last inequality we used the fact that the matrix $((A_\pi)^*)_{\pi^{-1}}$ is Robinsonian. Hence, equality holds throughout and thus $(A^R)_\pi$ is also an optimal solution of l_∞-FITTING-BY-ROBINSON (8.4) for matrix A_π. Applying Theorem 8.2.4 we can deduce that:

$$2\epsilon_{opt} = \|A_\pi - (A_\pi)^+\|_\infty = \max_{x,y,\in[n]} \{|A_{\pi(x),\pi(y)} - A^+_{\pi(x),\pi(y)}|\}$$

Using relation (8.6) we have that $A^+_{\pi(x),\pi(y)} = A_{\pi(u),\pi(v)}$ for some u, x, y, v, and therefore we get that:

$$\epsilon^* = \frac{1}{2}(|A_{uv} - A_{xy}|)$$

which concludes the proof. $\square$

In other words, the optimal error of l_∞-FITTING-BY-ROBINSONIAN (8.3) belongs to the set Δ obtained by computing all the differences among the values of the entries of A, which is a discrete set and can be enumerated in polynomial time by comparing all the possible entries of the matrix A. If the matrix is given as adjacency matrix, the size of Δ is thus $O(n^4)$, whereas if it is given as an adjacency list, the size of Δ is $O(m^2)$.

Hence, instead of solving FITTING-BY-ROBINSONIAN (8.3) by finding the right permutation π among $n!$ possible permutations, one can check whether A is ϵ-Robinsonian (i.e., it admits an ϵ-Robinson ordering) for each $\epsilon \in \Delta$. Then, in view of Corollary 8.2.7, the smallest ϵ for which A is ϵ-Robinsonian leads to an upper bound for FITTING-BY-ROBINSONIAN (8.3). In this setting, Chepoi and Seston [30] defined a sophisticated approximation algorithm which returns a $16\epsilon_{opt}$-Robinson ordering of A and which runs in $O(n^6 \log n)$ time.

8.3 The ϵ-multisweep algorithm

We present a new heuristic for l_∞-FITTING-BY-ROBINSONIAN (8.3). Roughly speaking, we extend the SFS algorithm (Algorithm 6.1) to the case when an additional parameter ϵ is given in input. We denote by ϵ-SFS this new variant of SFS. As for SFS, we use ϵ-SFS in a multisweep algorithm. Since in view of Lemma 8.2.10 the optimal value of l_∞-FITTING-BY-ROBINSONIAN (8.3) is contained in the set $\Delta = \{|A_{xy} - A_{uz}|/2 : u, x, y, z \in [n]\}$, which has polynomial size, we can then repeat the multisweep algorithm for each $\epsilon \in \Delta$ as in [30], and return the smallest ϵ for which an ϵ-Robinson ordering is found.

As in Chapters 5 and 6, we may assume without loss of generality that the matrix A is nonnegative and we denote by $0 = \alpha_0 < \cdots < \alpha_L$ the distinct values assumed by its entries.

8.3.1 ϵ-Similarity-First Search

In order to extend the SFS algorithm to solve seriation when the similarity matrix is not Robinsonian, we relax the concept of similarity partition of a given element p (Definition 6.3.2), in a similar fashion as in Definition 8.2.5.

8.3.1 Definition. (ϵ-similarity partition) Consider a nonnegative matrix $A \in \mathcal{S}^n$ and an element $p \in [n]$. Let $a_1 > \ldots > a_s > 0$ be the distinct values taken by the entries A_{px} of A for $x \in N(p) = \{y \in [n] : A_{py} > 0\}$ and, for $i \in [s]$, set $C_i^\epsilon = \{x \in N(p) : x \notin C_1^\epsilon \cup \cdots \cup C_{i-1}^\epsilon, |A_{px} - a_i| \leq 2\epsilon\}$. Then we define $\psi_p^\epsilon = (C_1^\epsilon, \ldots, C_s^\epsilon)$ (keeping only nonempty classes), which we call the *ϵ-similarity partition* of $N(p)$ with respect to p.

In other words, we group in the same blocks of ψ_p^ϵ the elements of $N(p)$ which are similar up to a 2ϵ-error. Hence, ϵ represents the sensitivity of partitioning the neighborhood $N(p)$ of the pivot p at each iteration of the ϵ-SFS algorithm. It is clear that ψ_p^ϵ reduces to the definition of ψ_p in Definition 6.3.2 if $\epsilon = 0$. On the other hand, say $\epsilon_{max} = \frac{\alpha_L - \alpha_0}{2}$ represents the largest entry in Δ; then $\psi_p^\epsilon = (N(p))$ for $\epsilon = \epsilon_{max}$.

Hence, we can directly extend the SFS algorithm to the case when a parameter ϵ is given as additional input (Algorithm 8.1). Basically, the only difference with respect to the classic SFS algorithm is that instead of using the original similarity partition ψ_p (see Definition 6.3.2) we now use the relaxed ϵ-similarity partition ψ_p^ϵ (see Definition 8.3.1), while the other operations are left unchanged.

For $\epsilon = 0$ Algorithm 8.1 reduces to Algorithm 6.1. As for the multisweep algorithm in Chapter 6, we define the variant ϵ-SFS$_+$ where the ties at line 4 in Algorithm 8.1 are broken using a linear order given as additional input to the algorithm. This variant will be used to define the multisweep algorithm based on ϵ-SFS.

8.3.2 ϵ-Robinson orderings

The output of the ϵ-SFS algorithm (Algorithm 8.1) is a linear order of $[n]$. As for the SFS multisweep algorithm (Algorithm 6.3), the idea is now to compute a finite number of ϵ-SFS$_+$ sweeps, each one of which takes as input the linear order returned by the previous sweep. The only difference is that this time, instead of checking if the current sweep σ_i is a Robinson ordering, we would need to

Algorithm 8.1: ϵ-*SFS*(A, ϵ)

```
input: a nonnegative matrix $A \in \mathcal{S}^n$ and a real number $\epsilon \geq 0$
output: a linear order $\sigma$ of $[n]$

1  $\phi = (V)$
2  for $i = 1, \ldots, n$ do
3      $S$ is the first class of $\phi$
4      choose $p$ arbitrarily in $S$
5      $\sigma(p) = i$
6      remove $p$ from $\phi$
7      $N(p)$ is the set of vertices $y \in \phi$ with $A_{py} > 0$
8      $\psi_p^\epsilon$ is the $\epsilon$-similarity partition of $N(p)$ with respect to $p$
9      $\phi$ =Refine $(\phi, \psi_p^\epsilon)$
10 return: $\sigma$
```

check if it is a ϵ-Robinson ordering for A, i.e., if relation (8.14) holds. Since this task requires $O(n^3)$ time, we adopt the following strategy, which is based on Lemma 8.2.7 and which requires only $O(n^2)$ time. Specifically, we compute the Robinson matrix $(A_{\sigma_i})^*$, obtained by applying definition (8.10) to A_{σ_i}. Let $\epsilon_i = \|A_{\sigma_i} - (A_{\sigma_i})^*\|_\infty$. Then we have to distinguish two cases:

1) If $\epsilon_i \leq \epsilon$ then $((A_{\sigma_i})^*)_{\sigma_i^{-1}}$ is a Robinsonian matrix lying in $[A - \epsilon_i, A + \epsilon_i] \subseteq [A - \epsilon, A + \epsilon]$ and thus, in view of Lemma 8.2.7, σ_i is a 2ϵ-Robinson ordering of A (in fact a $2\epsilon_i$-Robinson ordering).

2) If $\epsilon_i > \epsilon$ then, again in view of Lemma 8.2.7, σ_i is not an ϵ-Robinson ordering of A. Furthermore, since ϵ_i is the optimal value of the l_∞-FITTING-BY-ROBINSON problem (8.4) applied to (A_{σ_i}), then there cannot exist a Robinsonian matrix $A^R \in [A - \epsilon, A + \epsilon]$ whose Robinson ordering is σ_i.

In other words, if $\epsilon_i \leq \epsilon$ then, by Corollary 8.2.9, ϵ_i represents an upper bound on the optimal value of the l_∞-FITTING-BY-ROBINSONIAN problem (8.3).

We thus introduce the following multisweep procedure (Algorithm 8.2), which is the natural extension of Algorithm 6.3 applied to solve seriation when the similarity matrix A is not Robinsonian. Given a matrix A and a real number $\epsilon \geq 0$, the algorithm returns a positive answer if a 2ϵ-Robinson ordering is found, and a negative answer otherwise.

It is important to note that, while Algorithm 6.3 is a Robinsonian recognition algorithm (i.e., it returns a linear order π if and only if A is Robinsonian), Algorithm 8.2 is only a heuristic. In fact, we return a positive answer only if one of the computed linear orders σ_i is 2ϵ-Robinson. Therefore, it could be that the given matrix is 2ϵ-Robinsonian but none of the linear orders σ_i is a 2ϵ-Robinson ordering.

Algorithm 8.2: *ϵ-Robinson(A, ϵ)*

```
input: a matrix $A \in \mathcal{S}^n$ and a real number $\epsilon \geq 0$
output: TRUE if a $2\epsilon$-Robinson ordering of $A$ is found, FALSE if not

1 $\sigma_0 = \epsilon$-SFS$(A, \epsilon)$
2 for $i = 1, \dots n-2$ do
3     $\sigma_i = \epsilon$-SFS$_+(A, \epsilon, \sigma_{i-1})$
4     $\epsilon_i = \|A_{\sigma_i} - (A_{\sigma_i})^*\|_\infty$, using (8.10) applied to $A_{\sigma_i}$ to compute $(A_{\sigma_i})^*$
5     if $\epsilon_i \leq \epsilon$ then
6         return: TRUE: $\sigma_i$ is a $2\epsilon$-Robinson ordering
7 return: FALSE: No $2\epsilon$-Robinson ordering has been found.
```

8.3.3 ϵ-SFS-based heuristic

We can finally introduce the (heuristic) algorithm to solve seriation when the similarity matrix A is not Robinsonian, using Algorithm 8.2,

We may assume that the elements in Δ are all distinct and sorted by increasing values, after an appropriate preprocessing step. Then, starting from $\epsilon = 0$ (recall that we assume the matrix to be nonnegative) we scan the set Δ until we find the first ϵ such that Algorithm 8.3 returns a positive answer, i.e., A is 2ϵ-Robinsonian. Note that, for $\epsilon = 0$, Algorithm 8.2 reduces to Algorithm 6.3. For the sake of ease, here we use Algorithm 8.2 also for $\epsilon = 0$ but in practice one might want to first run Algorithm 6.3 for $\epsilon = 0$ and then compute Δ (which is time consuming) and repeat Algorithm 8.2 for $\epsilon > 0$. It is easy to see that Algorithm 8.3 will always return a positive answer, as any ordering π is ϵ-Robinson for A when $\epsilon = \epsilon_{max}$, where ϵ_{max} denotes the largest element in Δ.

We could not immediately prove that if Algorithm 8.2 returns a positive answer for some $\epsilon \in \Delta$, then it must return a positive answer also for every $\epsilon < \epsilon' \in \Delta$. For this reason, we could not implement binary search algorithms over the set Δ, which is more efficient than scanning it from the lowest to the highest value.

As already mentioned before, our algorithm is a heuristic. It is not able to state that A is not ϵ-Robinsonian, but it only returns that A is 2ϵ-Robinsonian if at some point in Algorithm 8.2 a 2ϵ-Robinson ordering σ_i is found. Hence, it might return a negative answer even though A is 2ϵ-Robinsonian, just because it cannot 'see' the correct 2ϵ-Robinson ordering. This is is coherent with the fact that the ϵ-ROBINSONIAN-RECOGNITION problem is NP-complete, and consequently it is not possible to recognize ϵ-Robinsonian matrices in polynomial time. Analyzing the quality of the approximation ϵ returned by Algorithm 8.3 will be carried out in Chapter 9.

Algorithm 8.3: *ϵ-Robinsonian*(A)

```
input: a nonnegative matrix $A \in \mathcal{S}^n$
output: a real number $\epsilon \geq 0$ such that $A$ is $2\epsilon$-Robinsonian
1 Construct the list $\Delta = \{|A_{xy} - A_{uz}|/2 : u, x, y, z \in [n]\}$ and sort it
2 $\epsilon_{max}$ is the largest element in $\Delta$
3 $\epsilon = 0$
4 while $\epsilon \leq \epsilon_{max}$ do
5     if $\epsilon$-Robinson$(A, \epsilon)$ is TRUE then
6         return: $\epsilon$
7     else
8         $\epsilon$ is updated to the next value in $\Delta$
```

We now analyze the complexity of Algorithm 8.3. As for the complexity analysis in Chapter 5 and 6, we assume that $A \in \mathcal{S}^n$ is a nonnegative symmetric matrix, given as adjacency list of an undirected weighted graph $G = (V = [n], E)$. So G is the support graph of A, whose edges are the pairs $\{x, y\}$ such that $A_{xy} > 0$ with edge weight A_{xy}, and $N(x) = \{y \in V : A_{xy} > 0\}$ is the neighborhood of $x \in V$. We assume that each vertex $x \in V = [n]$ is linked to the list of vertex/weight pairs (y, A_{xy}) for its neighbors $y \in N(x)$ and we let m denote the number of nonzero entries of A.

We may also assume that the support graph G associated to A is connected, as this implies that $m \geq n - 1$ and lead to a more compact complexity bound. Furthermore, this is a reasonable and smart preprocessing step to speed up the algorithm in practice, by splitting the problem in smaller subproblems. Hence, when we say that A is connected, we actually mean the support graph G of A. Then the following holds.

8.3.2 Lemma. *Algorithm 8.3 applied to an $n \times n$ symmetric nonnegative and connected matrix $A \in \mathcal{S}^n$ with m nonzero entries runs in $O(m^3 n \log n)$ time.*

Proof. The algorithm consists of two main tasks: compute the ordered set Δ and run Algorithm 8.2 for each $\epsilon \in \Delta$.

(1) To compute Δ we need to compare the distinct values taken by the entries of A, which can be done in $O(n^2)$ time. Moreover, as $|\Delta| = O(m^2)$, the set Δ can be sorted in $O(m^2 \log m)$.

(2) The most expensive task in Algorithm 8.2 is to run the ϵ-SFS algorithm. It is easy to see that the complexity of Algorithm 8.1 is the same as for the classic SFS algorithm (Algorithm 6.1), as the only difference is the definition of the similarity partition ψ_p^ϵ, which does not affect the overall

complexity of the SFS routine. Furthermore, as already mentioned earlier in Subsection 8.3.2, checking if π is a 2-ϵ-Robinson ordering can be done in $O(n^2)$ by computing $(A_\pi)^*$. Hence, using the result in Theorem 6.5.10, we can claim that Algorithm 8.2 runs in $O(n^2 + mn \log n)$ time. In the worst case, we repeat Algorithm 8.2 at most $O(|\Delta|) \leq O(m^2)$ times. Therefore, the final complexity of Algorithm 8.2 is $O((n^2 + mn \log n)m^2)$.

The overall complexity complexity of Algorithm 8.3 is given by the sum of the above two tasks, i.e., $O(m^2 \log m + n^2m^2 + m^3n \log n)$. Since we assumed A to be connected, then the above complexity is bounded by $O(m^3n \log n)$. □

Note that to speed up the heuristic one could simply fix the number of sweeps in the ϵ-multisweep algorithm to a constant factor, in which case the overall complexity would be $O(m^3 \log n)$.

8.3.4 Example

We discuss an example showing how Algorithm 8.3 works concretely on the example in (8.9). The set Δ (containing ϵ_{opt}) is:

$$\Delta = \left\{0, \frac{1}{2}, 1, \frac{3}{2}, \dots, \frac{21}{2}, 11\right\}.$$

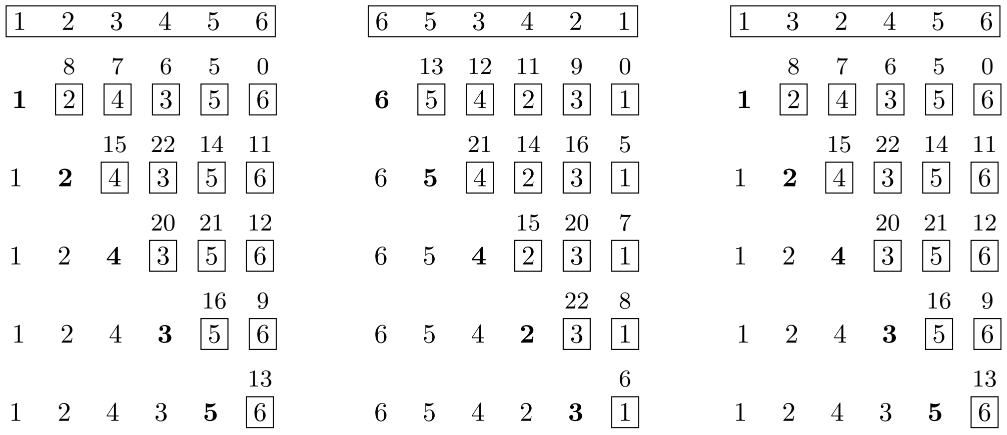
We start exploring the set Δ from the smallest value.

First iteration ($\epsilon = 0$) The first three sweeps of Algorithm 8.2 are given in Figure 8.1. After the third sweep Algorithm 8.2 will loop between permutations returned in the first two sweeps. Hence, the final ordering is $\pi = (1, 2, 4, 3, 5, 6)$. We cannot claim that it is 2-ϵ-Robinson, because the matrix $(A_\pi)^*$ has distance to the original matrix A equal to $\epsilon^* = \frac{7}{2} > \epsilon$.

$$(A_\pi)^* = \begin{array}{c|cccccc} & \mathbf{1} & \mathbf{2} & \mathbf{4} & \mathbf{3} & \mathbf{5} & \mathbf{6} \\ \hline \mathbf{1} & * & 8 & 7 & 6 & 5 & 0 \\ \mathbf{2} & & * & 18.5 & 18.5 & 14 & 10 \\ \mathbf{4} & & & * & 20 & 16 & 10 \\ \mathbf{3} & & & & * & 21 & 12 \\ \mathbf{5} & & & & & * & 13 \\ \mathbf{6} & & & & & & * \end{array}$$

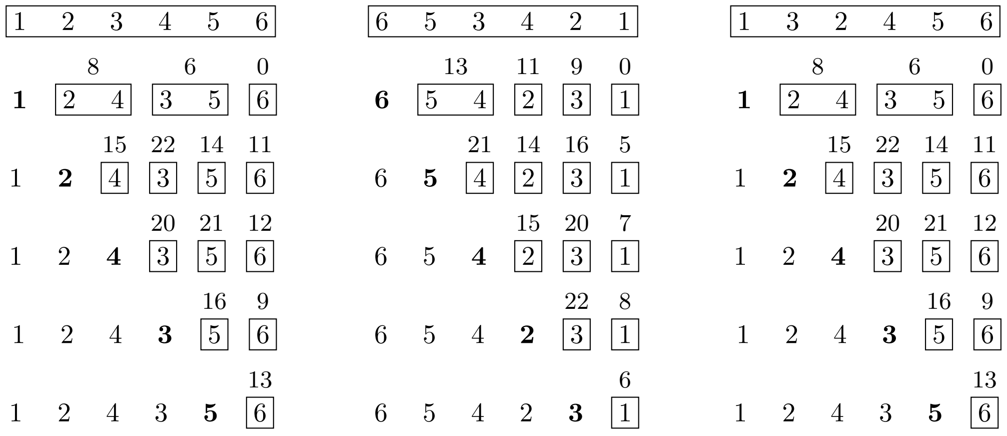
Second iteration $\left(\epsilon = \frac{1}{2}\right)$ The first three sweeps of Algorithm 8.2 are given in Figure 8.2 at page 163. After the third sweep Algorithm 8.2 will loop again between the first two permutations, returning the same order as the fourth iteration.

Third iteration ($\epsilon = 1$) The first sweep of Algorithm 8.2 is given in Figure 8.3. The ordering returned after the first sweep is $\pi = (1, 2, 3, 4, 5, 6)$, and the matrix $(A_\pi)^*$ is the same as A^* at page 154. Since $\epsilon^* = \|A_\pi - (A_\pi)^*\| = 1$ and $\epsilon^* \leq \epsilon$, then π is 2ϵ-Robinson and we stop Algorithm 8.3. In this case $\epsilon^* = \epsilon_{opt} = 1$ is the optimal solution of the l_∞-FITTING-BY-ROBINSONIAN problem (8.3). Note that the queues of the ϵ-SFS algorithm change with respect of the given ϵ.



(a) first sweep (b) second sweep (c) third sweep

Figure 8.1: Iterations of ϵ-SFS for $\epsilon = 0$: in bold the pivot which is chosen at the current iteration; above the blocks, the similarity between the new pivot and the vertices in the classes of the queue.



(a) first sweep (b) second sweep (c) third sweep

Figure 8.2: Iterations of ϵ-SFS for $\epsilon = \frac{1}{2}$: in bold the pivot which is chosen at the current iteration; above the blocks, the similarity between the new pivot and the vertices in the classes of the queue.

$$\begin{array}{cccccc}
\multicolumn{6}{c}{\boxed{1\quad 2\quad 3\quad 4\quad 5\quad 6}}\\
 & & 8 & & 5 & 0\\
\mathbf{1} & \multicolumn{3}{c}{\boxed{2\quad 3\quad 4}} & \boxed{5} & \boxed{6}\\
 & & 22 & 15 & 14 & 11\\
1 & \mathbf{2} & \boxed{3} & \boxed{4} & \boxed{5} & \boxed{6}\\
 & & & 20 & 16 & 9\\
1 & 2 & \mathbf{3} & \boxed{4} & \boxed{5} & \boxed{6}\\
 & & & & 21 & 12\\
1 & 2 & 3 & \mathbf{4} & \boxed{5} & \boxed{6}\\
 & & & & & 13\\
1 & 2 & 3 & 4 & \mathbf{5} & \boxed{6}
\end{array}$$

Figure 8.3: Iterations of ϵ-SFS for $\epsilon = 1$: in bold the pivot which is chosen at the current iteration; above the blocks, the similarity between the new pivot and the vertices in the classes of the queue.

8.4 Conclusions and future work

In this chapter we discussed how to solve the seriation problem when the similarity matrix A is not Robinsonian, by finding a 'close' Robinsonian approximation. After a short overview of the l_∞-FITTING-BY-ROBINSONIAN problem and ϵ-ROBINSONIAN-RECOGNITION problem, we introduced the ϵ-Similarity-First Search algorithm (ϵ-SFS), an extension of the SFS algorithm presented in Chapter 6 to the case when the input matrix is not Robinsonian. We then discussed a multisweep algorithm to recognize ϵ-Robinsonian matrices based on ϵ-SFS. We could not prove any immediate result on the quality of the solution returned by Algorithm 8.3. Nevertheless, we think that the ϵ-SFS multisweep algorithm illustrates well the potential of the SFS algorithm. Indeed, the main difference between the classic SFS algorithm and the ϵ-SFS algorithm is given by the different definitions of Similarity Partition (see Definition 6.3.2) and ϵ-similarity partition (see Definition 8.3.1). Hence, in principle, one could define, e.g, a k-similarity partition, where the neighborhood $N(p)$ of the pivot p is partitioned in classes consisting of k similar vertices. In any case, the structure of the SFS multisweep algorithm would remain unchanged. Therefore, the SFS algorithm could be used to develop several different heuristics for the seriation problem, not only the one discussed in this chapter.

An open question is whether it can be proven that the error made by our algorithm can actually be bounded by a constant approximation factor or not. A possible approach could be, for example, to relax the concept of anchor introduced in Chapter 6 to the notion of ϵ-anchor (analogously to the relaxation done for Robinson ordering and of SFS), and then extend the results in Chapter 6.

9 Computational experiments

In this final chapter we give some computational experiments regarding the performance of Algorithm 5.5, Algorithm 6.3 and Algorithm 8.3. In Section 9.1 we discuss how to design the experiments, i.e., how to generate seriation instances and which parameters to take into account to compare the performance of the algorithms. In Section 9.2 we give the results of the experiments for Robinsonian matrices. In Section 9.3 we give the results of the experiments for non-Robinsonian matrices. Finally, in Section 9.4 we conclude the chapter commenting the performance of our new Robinsonian recognition algorithms.

9.1 Design of experiments

The main goal of this chapter is to give some insights on how the algorithms introduced in this thesis perform in practice. In order to do so, we define five parameters aimed to represent some structural important properties of a matrix (e.g., density or number of distinct values). We then generate several scenarios by choosing different combinations of the parameters, and we compare the algorithms on such different scenarios in order to see if the performance of the algorithms is affected by special structures of the data. As benchmark, we will use the spectral algorithm of Atkins *et al.* [5] discussed in Subsection 3.2.3. Our motivation for this choice is based on the fact that it is easy to implement and, to the best of our knowledge, the other recognition algorithms presented in Subsection 3.2.2 are not implemented or not available.

A crucial point in the experiments is to decide how to generate a Robinsonian matrix and how to generate a non-Robinsonian matrix. In what follows, we will denote by A a Robinsonian matrix and by A_ϵ a non-Robinsonian matrix, respectively. As already discussed in previous chapters, when we refer to Robinsonian matrices we mean in fact Robinsonian similarity matrices.

Our experiments rely on the main assumption that all the entries of the matrix are integers, which is made for the sake of ease and without loss of generality. The algorithms were implemented in C++ and the experiments were run on a 2,7 GHz Intel Core i5 computer with 16 GB of RAM. For an efficient data structure, we relied on the linear algebra library *Armadillo*, which allows eigenvalues/eigenvectors computation for (sparse) matrices [104].

The rest of the section is organized as follows: in Subsection 9.1.1 we discuss the choice of the parameters to generate instances; in Subsection 9.1.2 we show how to generate Robinson(ian) matrices; in Subsection 9.1.3 we show how to generate non-Robinsonian matrices, by adding an artificial error in the original Robinsonian structure.

9.1.1 Parameters

We use the following five parameters (in bold) to tune the structure of the Robinsonian matrices and thus to create different instances of the seriation problem.

1. The size $\mathbf{n}$ of the matrix, i.e., the number of rows and columns.
2. The density $\mathbf{d} \in [0, 1]$ of the matrix, i.e., the percentage of nonzero entries (with respect to $\mathbf{n}^2$).
3. The largest value $\boldsymbol{\alpha_L}$ of the matrix. For $\boldsymbol{\alpha_L} = 0$ we generate 0/1 matrices.
4. The probability $\mathbf{p}_e \in [0, 1]$ that an entry is affected by error.
5. The intensity $\mathbf{e} \in [0, 1]$ of the error. This parameter reflects the magnitude of the error for the entries affected by error. For a given $\mathbf{e}$, the error ϵ is then defined by:
$$\epsilon = \lfloor \mathbf{e} \cdot \boldsymbol{\alpha_L} \rfloor. \tag{9.1}$$

The parameters and the corresponding range of values are summarized in Table 9.1. Note that these parameters are only used to control and diversify the generation process. Hence, when analyzing the experimental results, we will classify the matrices according to their real density, number of distinct values, etc., and not according to the parameters.

In Subsection 9.1.2 we show how to generate a Robinsonian matrix A for many combinations of ($\mathbf{n}$, $\mathbf{d}$, $\boldsymbol{\alpha_L}$). Given such a matrix, in Subsection 9.1.3 we show how to create a non-Robinsonian matrix A_ϵ for all possible combinations of ($\mathbf{p}_e$, $\mathbf{e}$), by adding to A an error ϵ defined as in (9.1).

Table 9.1: Range of values for each parameter considered in the experiments.

symbols	**description**	**range**
$\mathbf{n}$	number of rows/columns	$[100, 1000]$
$\mathbf{d}$	density, i.e., percentage of nonzero entries	$[0.1, 1]$
$\boldsymbol{\alpha_L}$	largest value	$[0, 800]$
$\mathbf{p}_e$	probability that an entry is affected by error	$\{0.1, 0.3\}$
$\mathbf{e}$	intensity of the error (in percentage)	$\{0.05, 0.1\}$

9.1.2 Robinsonian matrices generation

In this subsection we analyze how to generate a Robinson matrix given some fixed parameters ($\mathbf{n}$, $\mathbf{d}$, $\boldsymbol{\alpha_L}$). Once we create a Robinson matrix, the corresponding Robinsonian matrix is obtained by randomly permuting its rows and columns.

In order to diversify the Robinson structure, we discuss four different procedures to generate random Robinson matrices. Note that, since we are interested in creating symmetric matrices, we actually consider only the upper triangular entries when generating Robinson matrices. Furthermore, as the entries on the main diagonal do not play a role, we always set them to the largest value $\boldsymbol{\alpha_L}$. In the beginning of the generation process, the matrix A has size $\mathbf{n} \times \mathbf{n}$ with all (off-diagonal) zero entries.

Generation 1 We compute the number of (upper-diagonal) nonzero entries, i.e.,

$$m = \mathbf{d} \cdot \frac{\mathbf{n}^2 - \mathbf{n}}{2}.$$

We generate m random integer numbers in the interval $[1, \boldsymbol{\alpha_L}]$ and we sort them for decreasing values. Then, we place the m values in the matrix diagonal by diagonal, starting from the first upper diagonal (of length $n-1$) to the last upper diagonal (of length 1). To further randomize the process, we place the values on each diagonal randomly. Because the m values are originally sorted and because we fill the matrix diagonal by diagonal moving away from the main diagonal, it is easy to see that we obtain a Robinson matrix by construction. With this generation we directly control the density and the number of distinct elements of the matrix. An example of a Robinson matrix generated with this procedure is shown in Figure 9.1a.

Generation 2 We generate $\mathbf{n}$ random integers $x_1, \ldots, x_n$ in the interval $[0, \boldsymbol{\alpha_L}]$ and we sort them for decreasing order. Then, we compute the distance matrix D where $D_{ij} = |x_i - x_j|$. As already mentioned in Subsection 2.1, such a matrix is a Robinson dissimilarity matrix by construction. Finally, we obtain the corresponding Robinson similarity $A = \boldsymbol{\alpha_L} J_n - D$. Note that, differently from the above generation, in this case we cannot directly control the sparsity of the matrix. Nevertheless, it represents a simple and intuitive way to generate Robinson matrices. An example of a Robinson matrix generated with this procedure is shown in Figure 9.1b.

For the last two generations, we generate matrices with density $\mathbf{d}$ by opportunely defining the bandwidth b of the matrix. Specifically, we fix a threshold $\Theta = \mathbf{d} \cdot \mathbf{n}^2$, which is the 'desired' number of nonzero entries of the matrix. The idea is, for some fixed b, to compute the number of nonzero entries m_b that a matrix with bandwidth b would have, which is simply given by:

$$m_b = 2\sum_{i=1}^{n} \min\{b, \mathbf{n} - i\}. \tag{9.2}$$

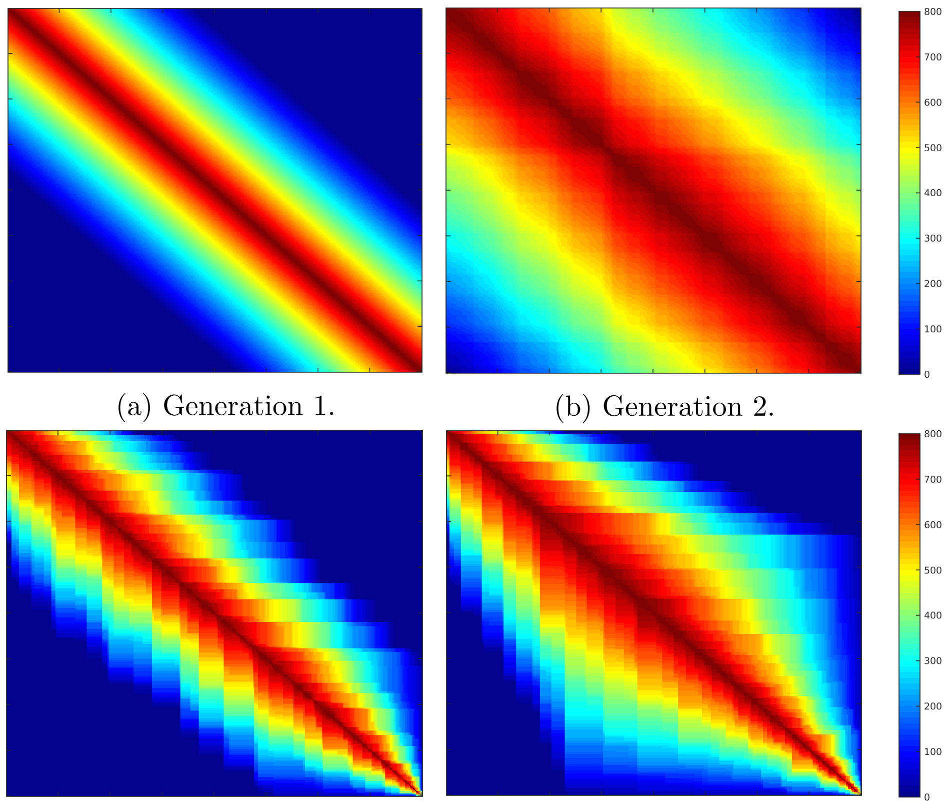
We start with $b = \mathbf{d} \cdot \mathbf{n}$, and we distinguish two cases:

- If $m_b < \Theta$ then we increase $b = b + 1$;
- If $m_b > \Theta$ then we decrease $b = b - 1$;

Once b is updated, the number of nonzero entries m_b is computed again as in (9.2), and the above process is repeated until m_b converges to the desired number of nonzero entries Θ.

Generation 3 Given the bandwidth b as in the procedure described above, for the i-th row we generate a random integer number $b_i \in [2, b]$, which will represent the number of off-diagonal entries in the i-th row (recall that we are in the upper triangular part). We then generate $\min\{b_i, \mathbf{n} - i\}$ random integer values in the range $[1, \boldsymbol{\alpha_L}]$, which we sort and we place on i-th row, starting from entry $A_{i,i+1}$. The matrix obtained with this process might be non-Robinson (the Robinson property might be violated in the columns). Hence, we modify the entries in order to get the final Robinson matrix. Specifically, starting from the upper-right corner, we set $A_{ij} = \max\{A_{i-1,j}, A_{i,j+1}\}$ for each $i = 1, \ldots, \mathbf{n}$ and $j = \mathbf{n}, \ldots, i+1$. We repeat the above procedure for each $i = 1, \ldots, \mathbf{n}$. With this generation we directly control the density and the number of distinct elements of the matrix. An example of a Robinson matrix generated with this procedure is shown in Figure 9.1c.

Generation 4 Given the bandwidth b as in the procedure described above, this last generation is basically identical to Generation 3, with the only difference that this time, we generate $\min\{b_i + i, \mathbf{n} - i\}$ random values in the range $[1, \boldsymbol{\alpha_L}]$ for each row $i = 1, \ldots, \mathbf{n}$. In this way, we try to generate non-banded matrices. With this generation we directly control the number of distinct elements of the matrix but more indirectly the sparsity of the matrix (as we increase the bandwidth row by



(a) Generation 1. (b) Generation 2.

(c) Generation 3. (d) Generation 4.

Figure 9.1: Different Robinson matrices generated for $\mathbf{n} = 400, \boldsymbol{\alpha_L} = 800$ and $\mathbf{d} = 0.6$.

row). An example of a Robinson matrix generated with this procedure is shown in Figure 9.1d.

To summarize, we generate four Robinsonian matrices (one for each generation type) for every triple ($\mathbf{n}$, $\mathbf{d}$, $\boldsymbol{\alpha_L}$), where we choose:

i) $\mathbf{n} = \{100, 200, \dots, 900, 1000\}$;

ii) $\mathbf{d} = \{0.1, 0.2, \dots, 0.9, 1\}$;

iii) $\boldsymbol{\alpha_L} = \{0, 5, 10, 20, 50, 100, 150, 200, 400, 600, 800\}$.

For $\boldsymbol{\alpha_L} = 0$ we generate 0/1 matrices. Computing instances for all the possible combinations of the above parameters, we get approximately 4000 instances which we have to test with all the three algorithms.

9.1.3 Error generation

Given a Robinsonian matrix $A \in \mathcal{S}^n$ generated as in Subsection 9.1.2, in this subsection we show how to create a non-Robinsonian matrix for some fixed parameters $(\mathbf{p}_e, \mathbf{e})$. When talking about 'error', we refer to the fact that the matrix is not Robinsonian. Therefore, we way that the Robinsonian structure is affected by error. Hence, let $\epsilon = \lfloor \mathbf{e} \cdot \boldsymbol{\alpha_L} \rfloor > 0$ as in (9.1) denote the maximum error. The idea is to generate a perturbation matrix $\Xi = (\xi_{ij}) \in \mathcal{S}^n$ as follows. In the beginning, Ξ contains only zero entries. Then, we scan the (off-diagonal) upper entries and we decide with probability $\mathbf{p}_e$ if the entry A_{ij} will be affected by error, for each $i = 1, \dots, n$, $j = i+1, \dots, n$. If this is the case, we generate a random perturbation $\xi_{ij} = \xi_{ji} \in [1, \epsilon]$. Finally, we set $A_\epsilon = A + \Xi$.

As already shown in Table 9.1, we choose the parameters as follows:

i) $\mathbf{p}_e = \{0.1,\ 0.3\}$;

ii) $\mathbf{e} = \{0.05,\ 1\}$.

Hence, for each Robinsonian matrix generated as described in Subsection 9.1.2, we create four non-Robinsonian matrices, given by all the possible combinations of parameters $(\mathbf{p}_e, \mathbf{e})$. As we will see, the computational time of the algorithms presented in Chapters 5 and 8 is significantly affected by the number of distinct values. Hence, for the generation of non-Robinsonian matrices, we will perturb only Robinsonian matrices with $\boldsymbol{\alpha_L} \leq 200$. This leads to roughly 9000 instances in total. Note that if, for some parameters $(\mathbf{n}, \mathbf{d}, \boldsymbol{\alpha_L})$, we obtain the perturbation $\epsilon = 0$ (e.g., for $\mathbf{e} = 0.05$ and $\boldsymbol{\alpha_L} = 5$), then the corresponding matrix A_ϵ is in fact Robinsonian and thus ignored and not included in further experiments.

9.2 Results with Robinsonian matrices

In this section we discuss the computational results of the recognition algorithms when applied to a Robinsonian matrix generated as described in Subsection 9.1.2. This corresponds to verifying the Robinsonian property of an input matrix, i.e., find a Robinson ordering. We will discuss in Section 9.3 the case when the recognition algorithms are applied to non-Robinsonian matrices. In order to understand how the structure of the matrix affects the performance of the algorithms, we distinguish different scenarios depending on the number of distinct values and the number of nonzero entries (i.e., the density of the matrix). As already mentioned earlier, this distinction is made on the actual structure of the matrix, and not based on the values of parameters $(\mathbf{n}, \mathbf{d}, \boldsymbol{\alpha_L})$, which are defined only for the sake of diversifying the generation of Robinsonian matrices. This is the reason why, in this section, we denote them by (n, d, L), and not in bold as before.

Specifically, we grouped the matrices generated in three classes, according to their number of distinct values as follows:

1. matrices with *low* number of distinct values, i.e., ≤ 50;
2. matrices with *medium* number of distinct values, i.e., > 50 and ≤ 200;
3. matrices with with *high* number of distinct values, i.e., > 200.

On top of the above groups, we also distinguish the matrices according to their density as follows:

1. *sparse* matrices, i.e., with a density at most 30%;
2. *normal* matrices, i.e., with a density between 30% (excluded) and 70%;
3. *dense* matrices, i.e., with a density strictly bigger than 70%.

For the sake of abbreviation, we use the following notation for the recognition algorithms:

(i) we denote by **spectral** the algorithm of Atkins *et al.* [5] presented in Subsection 3.2.3;

(ii) we denote by **LBFS** the Lex-BFS-based algorithm presented in Chapter 5 (Algorithm 5.5);

(iii) we denote by **SFS** the SFS multisweep algorithm presented in Chapter 6 (Algorithm 6.3).

Hence, for each combination of $n \leq 1000$, number of distinct values (i.e., *low, medium, high*) and density (i.e., *sparse, normal, dense*), we have a group of matrices with the same structure. Then, we run each recognition algorithm on all matrices in the same group and we take the average computational time for each algorithm, as shown in Table 9.2.

For example, for *dense* matrices of size $n = 1000$ and *low* number of distinct values, the average time of **spectral** it is 663,46 milliseconds, for **SFS** it is 642,58 milliseconds and for **LBFS** is 8,05 seconds (see bottom-left entries in Table 9.2).

As we will discuss later, for verifying Robinsonian matrices, **LBFS** is much slower compared to **spectral** and **SFS**, and it gets worse for increasing number of distinct values. Hence, since we have to repeat the experiments for several thousands of matrices, **LBFS** would slow down the whole simulation process. For this reason, although in absolute value the time required by **LBFS** is still reasonable, we decided to use **LBFS** only to recognize matrices with at most 200 distinct values.

To help the reading of Table 9.2, we provide some additional charts in Figures 9.2, 9.3 and 9.4, which were constructed using some data in Table 9.2.

Each figure consists of two lines of charts. The first line (on top) is a graphical representation of the values in Table 9.2 for some structured matrices. For example, in Figure 9.2a we have the (average) time performance of a *sparse* matrix with *low* number of distinct values for each $n = 100, \dots, 1000$. In these first charts, **LBFS** performance does not appear, because, as mentioned before, it is significantly slower compared with **spectral** and **SFS**.

In the second line we give additional bar charts, where each chart represents the number of instances (in percentage) for which each algorithm is faster than the other two. Each chart on the bottom has one-to-one correspondence with the chart immediately above. For example, in Figure 9.2d we can deduce that, for *sparse* matrices with *low* number of distinct values and (say) $n = 100$, **Atkins** is the fastest algorithm for 20% of instances, **SFS** for 70% of instances and **LBFS** for the remaining instances. The idea of these last charts is to give some insights on the single (non-averaged) performance of the algorithms.

We comment below on the computational results for each algorithm.

Spectral algorithm The algorithm performs extremely well when applied to our Robinsonian instances. Most of the time it is faster than the other two algorithms, and it is able to return a Robinson ordering in less than one second also for dense 1000×1000 matrices (i.e., approximately one million of nonzero entries). The most time consuming instance was a 1000×1000 matrix with 6 distinct values and 100% density, which required around 2,4 seconds. In comparison, for the same instance **SFS** required 0,9 seconds, and **LBFS** 3,6 seconds, with the depth of the recursion tree equal to 4. One of the reasons why the algorithm is so fast is also because, in *Armadillo*, it is possible to generate the first k eigenvalues (and corresponding eigenvectors) of a matrix instead of computing all n of them. Because in our case we only need $k = 2$ eigenvalues/eigenvectors, the generation of the Fiedler vector can be even 10 times faster than computing all n of eigenvalues/eigenvectors. However, we encountered a numerical problem for some instances produced with the Generation methods 3 and 4. Specifically, as mentioned in Subsection 3.2.3, if the Fiedler vector has repeated values, then we recourse on the submatrix corresponding to the indices of the Fiedler vector with the same value. The issue is that, due to numerical errors in computing eigenvalues of the matrix, two entries of the Fiedler vector might be considered equal even though they are not (or vice versa). Consequently, we could apply recursion on the 'wrong' submatrix, which might lead to a non-Robinson ordering. Indeed, some permutations returned by **spectral** are not Robinson orderings. Note that this mistake is not a consequence of the choice of computing only k eigenvalues above mentioned.

SFS multisweep algorithm For Robinsonian matrices, the algorithm is comparable with **spectral**, and it even outperforms it, e.g., for sparse matrices with size less than 300 (see Table 9.2 and Figure 9.2). The most time consuming instance was a 1000×1000 matrix with 600 distinct values and 99% density, which required around 1,5 seconds. In comparison, for the same instance **spectral** required 1,2 seconds (while **LBFS** was not run for more than 200 distinct values). Therefore, the maximum time required by **SFS** is lower the the maximum time required by **spectral** (1,2 seconds versus 3,6 seconds). If we take the single computations in the bar charts, we see that most of the time **spectral** solves the single instances faster than **SFS**. However, because the curve in the plots are close to each other, this suggests that the time performance are not so different in absolute value. Differently from **spectral**, this **SFS** algorithm is combinatorial, which means that numerical errors cannot occur. Indeed, all the permutations returned are Robinson orderings. It is interesting to notice that the maximum number of sweeps (needed for terminating the SFS multisweep algorithm) that we encountered throughout all the instances was 4 (while in theory we can only show that the algorithm terminates, in the worst case, after $n - 1$ sweeps). This is a really small number, especially considering matrices of size 1000. The reason why it is so low is that, if the matrix has many distinct values, then the similarity partition (see Definition 6.3.2) at each iteration will consist of many blocks, and thus we will have less ties. Hence, once an anchor is found, the SFS algorithm almost reduces to sort an n dimensional vector for decreasing values. On the other hand, if the matrix has few distinct values, we have more ties. However, because more entries assume the same value, the matrix admits more Robinson orderings, which means that after few sweeps one of these orderings could be more likely encountered.

As we will see also for **LBFS**, reordering the matrix according to a given linear order is one of the most time consuming tasks of the **SFS** algorithm. Hence, if there exists a particular subclass of matrices for which the number of sweeps is tight (i.e., $n - 1$ for a $n \times n$ matrix), then the performance of **SFS** could be significantly worse than for **spectral**. Nevertheless, this is not the case for the instances generated as in Subsection 9.1.2.

LBFS-based algorithm The algorithm is without doubts the worst performing among the three recognition algorithms applied to Robinsonian matrices. As already mentioned before, we were not able to solve instances with more than 200 rows/columns in less than a minute. Furthermore, in general, the algorithm is 10 times slower than the other two. For example, to verify a 1000×1000 Robinsonian matrix with *medium* number of distinct values it needs, in average, almost one minute, while the other algorithms take around half a second. The most time consuming instance was a 1000×1000 matrix with 200 distinct values and 100% density, which required around 1,72 minutes with the depth of the recursion tree equal to 99. In comparison, for the same instance, both **spectral** and **SFS** required only 0,5 seconds. Also for **LBFS**, as for **SFS**, the main bottleneck is to reorder the matrix according to a given linear order. As discussed in Theorem 5.4.4, this permits a linear time implementation of Lex-BFS multisweep algorithms. Hence, for each level of the recursion tree, we reorder the matrix three times according to the 3 sweeps algorithm of Corneil in [33]. However, if the depth of the recursion tree is large (which is often the case), then we have to reorder the matrix many times, which explains why the algorithm becomes really time consuming. Furthermore, we found many instances where the depth of the recursion tree is actually equal to the number of distinct values (and larger than n), which explains why we could not deal with matrices with more than 200 distinct values within one minute. This is a crucial difference with respect to **SFS**, where we do not decompose our original matrix in 0/1 matrices. To conclude, we could not find any significant correlation between the number of distinct values and the depth of the recursion tree (or the size of the common refinement).

Large instances For **SFS** and **spectral** we repeated the experiments for Robinsonian matrices with size $n \geq 1000$. We generate four Robinsonian matrices (one for each generation type) for every triple ($\mathbf{n}$, $\mathbf{d}$, $\boldsymbol{\alpha_L}$), where we choose:

i) $\mathbf{n} = \{1000, 2000, \ldots, 9000, 10000\}$;

ii) $\mathbf{d} = \{0.1, 0.3, 0.5, 0.7, 0.9\}$;

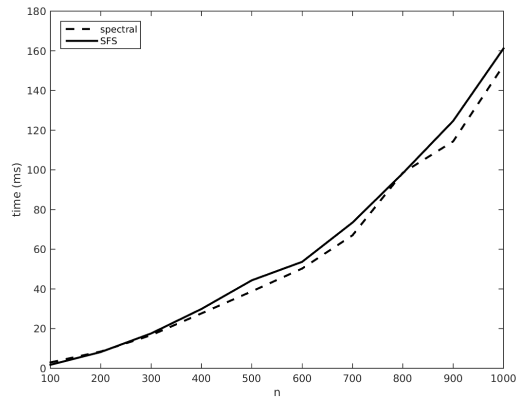
iii) $\boldsymbol{\alpha_L} = \{0, 50, 100, 200, 400, 600, 800\}$.

Therefore, the total number of large instances generated is now around 1400, instead of 4000 in the previous experiments. In fact, we repeated the experiments only for a subclass of matrices in order not to slow down the whole experiment process, as reading the data takes more than 5 minutes per matrix for large matrices. Analogously as above, we show in Table 9.3 the average computational time of **spectral** and **SFS** for matrices grouped according to their density, number of distinct values and size. To help the reader in visualizing the data in Table 9.3, we give some additional charts in Figures 9.5, 9.6 and 9.7. Analyzing the data, **SFS** seems to outperform **spectral** for increasing size. For example, for *dense* matrices of size 10000 with *high* number of distinct values, the average running time for **spectral** is around 7 minutes, against an average running time of 1 minute for **SFS**. The most time consuming instance for **spectral** was a 1000×1000 matrix with 50 distinct values and 95% density, which required 13 minutes. In comparison, for the same instance **SFS** required 1,5 minutes. The most time consuming instance for **SFS** was a 1000×1000 matrix with 2 distinct values (i.e., a 0/1 matrix) and 90% density, which required 14,5 minutes [1]. In comparison, for the same instance **spectral** required 3,5 minutes. To conclude, the results seems to suggest a really good performance of **SFS** for large instances, which makes the algorithm suitable to solve real world problems of large dimension.

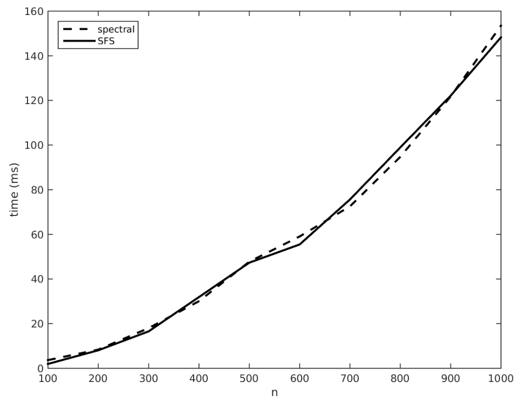
[1]If the input matrix has only 2 distinct values, there is not need to sort the neighborhood of the pivot $N(p)$ to create the similarity partition. However, this additional check was missing during the experiments, leading to high computational times for 0/1 matrices.

Table 9.2: (Average) Time performance of the algorithms to verify Robinsonian matrices (in milliseconds).

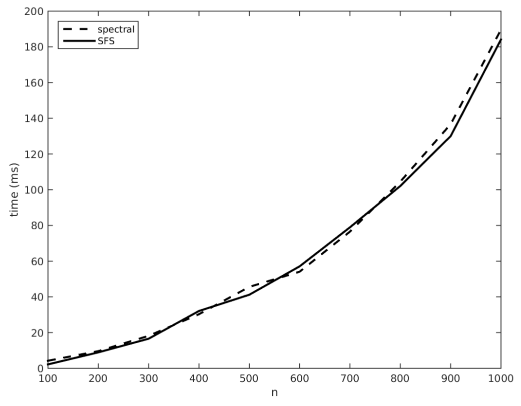
	# distinct values	**low** (≤ 50)			**medium** (> 50 and ≤ 200)			**high** (≥ 200)		
# nonzero entries	**algorithms** / **n**	**spectral**	**SFS**	**LBFS**	**spectral**	**SFS**	**LBFS**	**spectral**	**SFS**	**LBFS**
sparse (≤ 30 %)	**100**	2,98	1,78	10,57	3,68	1,97	58,85	4,24	2,20	-
	200	8,48	8,22	36,99	8,38	8,08	211,08	9,62	8,93	-
	300	16,69	17,58	83,08	18,00	16,55	513,76	18,18	16,58	-
	400	27,68	29,91	153,23	30,06	31,92	953,13	30,30	32,10	-
	500	38,78	44,35	209,87	47,77	47,33	1382,98	45,60	41,20	-
	600	50,28	53,66	277,90	59,06	55,47	1771,93	54,10	57,10	-
	700	67,02	73,45	383,13	72,54	75,64	2437,52	76,55	78,96	-
	800	98,54	98,29	526,48	94,76	98,96	3236,95	104,52	102,09	-
	900	114,36	124,67	616,90	121,75	122,12	4103,76	136,70	130,02	-
	1000	152,63	161,15	904,72	153,52	148,28	5047,28	189,63	184,12	-
normal (> 30 % and ≤ 70%)	**100**	3,16	4,65	26,25	3,46	5,20	196,26	3,41	5,04	-
	200	11,04	18,58	108,28	12,96	19,92	942,65	14,43	20,08	-
	300	25,62	40,91	252,98	29,46	44,37	2098,60	30,71	45,09	-
	400	49,50	76,23	459,03	55,82	74,65	3833,16	56,85	79,34	-
	500	73,35	108,69	645,23	84,66	113,71	5659,31	84,77	110,84	-
	600	108,05	139,40	893,37	126,33	153,15	7437,49	126,89	148,99	-
	700	143,32	186,48	1247,81	164,40	196,33	10402,90	172,27	195,22	-
	800	193,45	253,49	1646,54	232,95	246,19	13920,20	253,77	255,05	-
	900	254,46	307,13	2131,64	317,26	309,65	17909,20	310,84	326,79	-
	1000	331,47	408,70	2856,86	383,54	376,66	22601,10	442,26	499,45	-
dense (> 70 %)	**100**	3,87	6,81	66,58	3,89	7,72	493,64	3,89	7,78	-
	200	16,37	27,38	285,67	16,08	30,01	2126,32	16,95	31,57	-
	300	38,64	61,59	633,54	40,14	65,96	4904,51	38,32	69,41	-
	400	77,00	112,23	1165,52	76,81	114,90	9114,09	77,66	121,97	-
	500	122,27	158,87	1691,87	122,57	163,62	13693,00	114,96	161,89	-
	600	174,42	211,88	2349,12	173,31	210,19	18455,80	171,59	225,39	-
	700	273,01	291,58	3364,06	248,08	286,44	25932,80	245,26	299,84	-
	800	359,28	379,78	4493,35	339,09	373,69	34891,70	344,47	397,55	-
	900	489,78	487,85	5854,02	450,70	466,22	45060,20	450,22	519,41	-
	1000	663,46	642,58	8046,78	588,68	579,59	58410,50	707,10	775,99	-



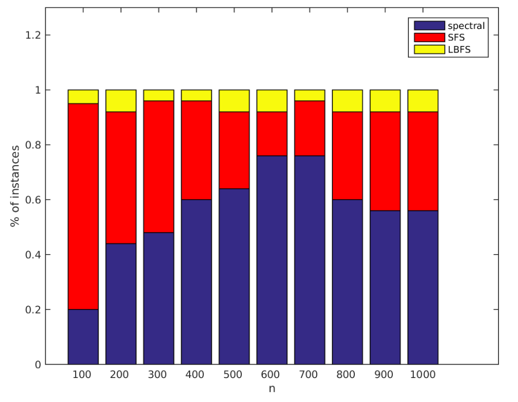
(a) sparse - low



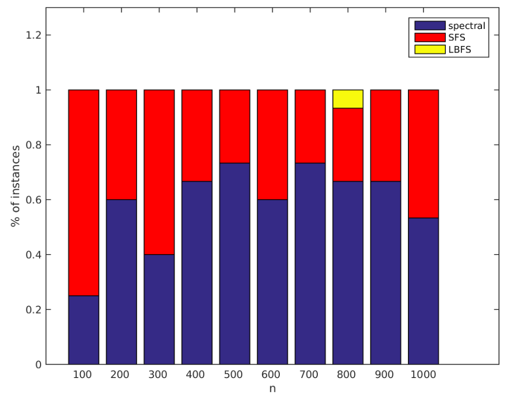
(b) sparse - medium



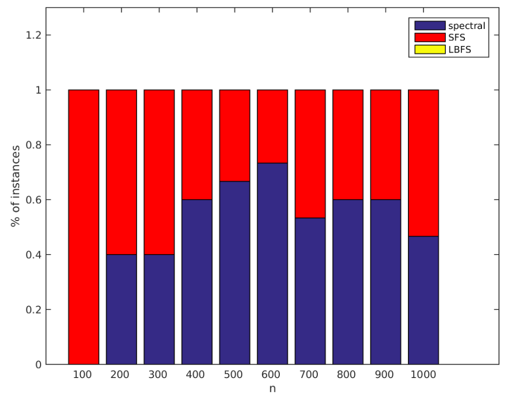
(c) sparse - high



(d) sparse - low

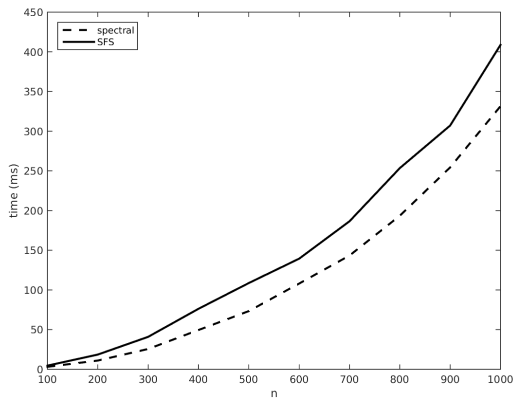


(e) sparse - medium

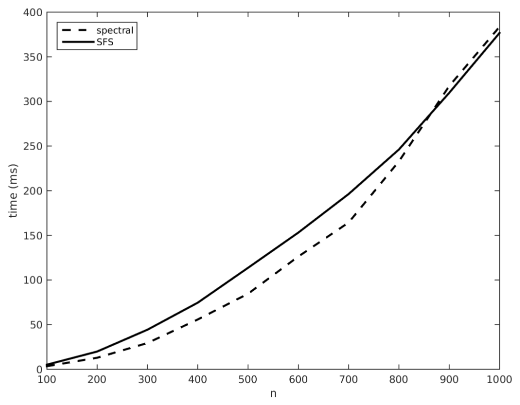


(f) sparse - high

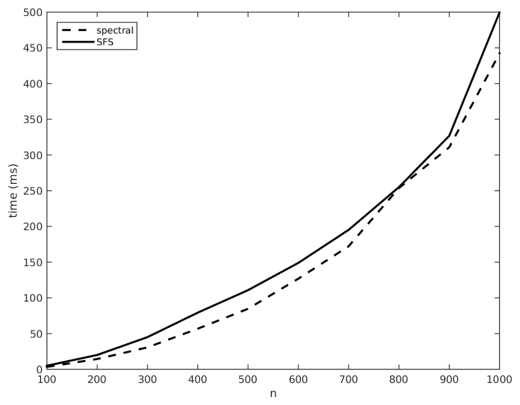
Figure 9.2: Time performance of the algorithms for *sparse* Robinsonian matrices.



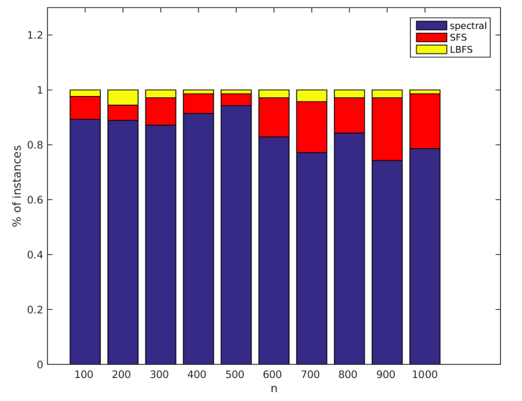
(a) normal - low



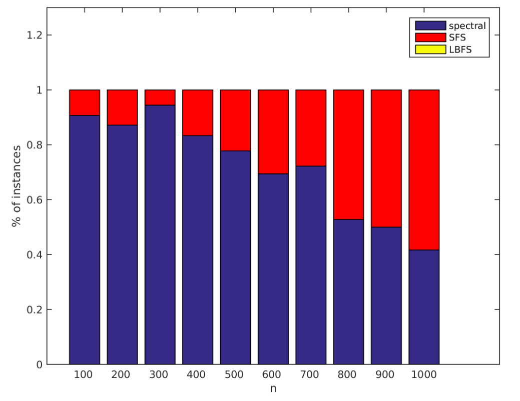
(b) normal - medium



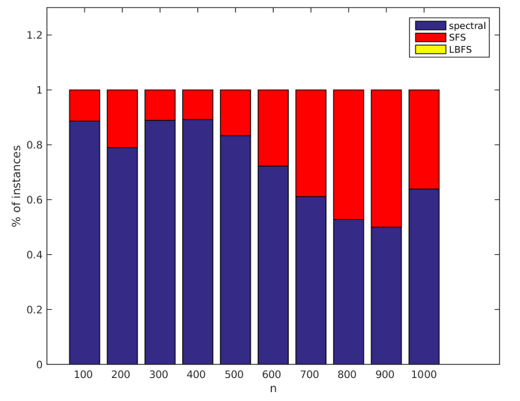
(c) normal - high



(d) normal - low

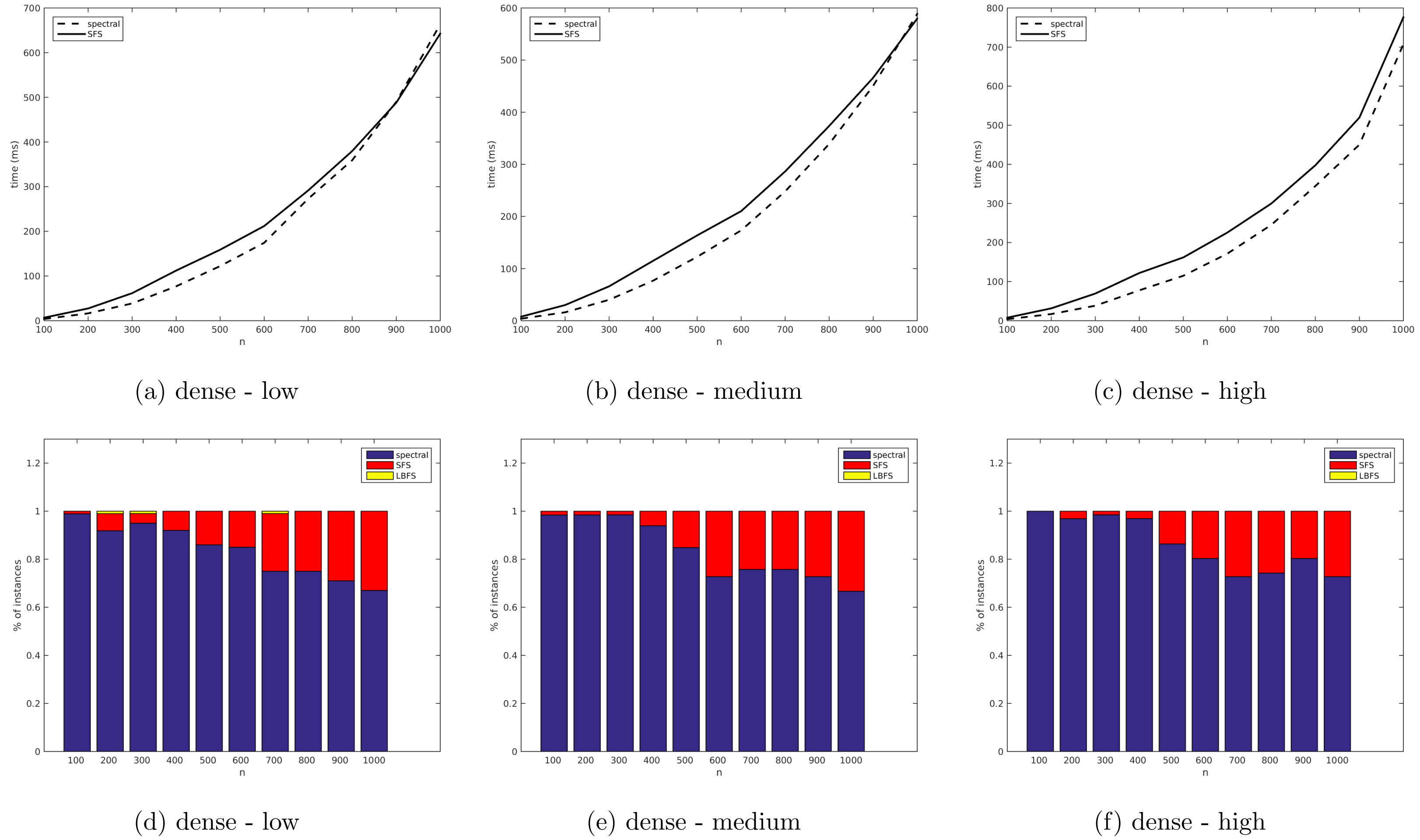


(e) normal - medium



(f) normal - high

Figure 9.3: Time performance of the algorithms for *normal* Robinsonian matrices



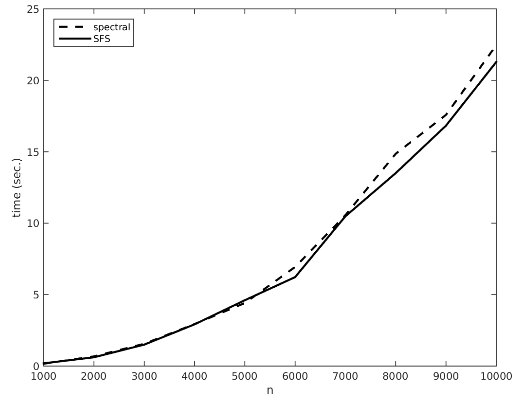
(a) dense - low (b) dense - medium (c) dense - high

(d) dense - low (e) dense - medium (f) dense - high

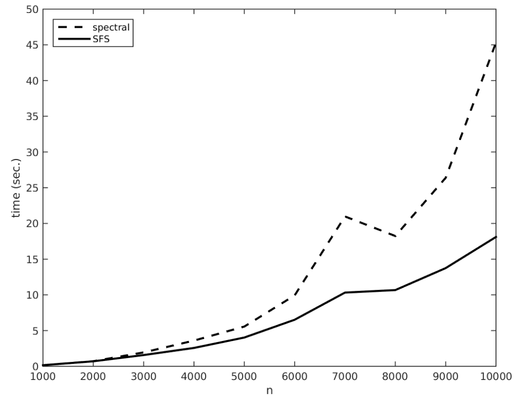
Figure 9.4: Time performance of the algorithms for *dense* Robinsonian matrices

Table 9.3: (Average) Time performance of the algorithms to verify Robinsonian matrices (in seconds) - large instances.

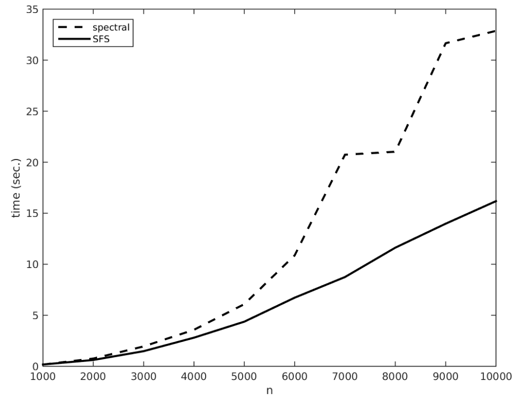
	# distinct values	**low** (≤ 50)			**medium** (> 50 and ≤ 200)			**high** (≥ 200)		
# nonzero entries	**algorithms** / **n**	**spectral**	**SFS**	**LBFS**	**spectral**	**SFS**	**LBFS**	**spectral**	**SFS**	**LBFS**
sparse (≤ 30 %)	**1000**	0,16	0,19	-	0,16	0,16	-	0,17	0,18	-
	2000	0,68	0,62	-	0,72	0,7	-	0,76	0,62	-
	3000	1,56	1,5	-	1,95	1,58	-	1,95	1,48	-
	4000	2,94	2,92	-	3,6	2,57	-	3,58	2,81	-
	5000	4,41	4,61	-	5,56	4,03	-	6,09	4,38	-
	6000	6,94	6,23	-	9,93	6,52	-	10,87	6,72	-
	7000	10,56	10,48	-	20,98	10,32	-	20,73	8,75	-
	8000	14,86	13,5	-	18,24	10,67	-	21,03	11,63	-
	9000	17,58	16,83	-	26,38	13,75	-	31,66	13,97	-
	10000	22,46	21,28	-	45,32	18,11	-	32,87	16,18	-
normal (> 30 % and ≤ 70%)	**1000**	0,32	0,4	-	0,45	0,41	-	0,45	0,46	-
	2000	1,53	1,8	-	2,2	1,67	-	1,99	1,71	-
	3000	4,42	4,77	-	5,49	3,77	-	5,74	3,64	-
	4000	9,13	9,46	-	13,04	6,33	-	14,22	6,54	-
	5000	17,08	16,45	-	26,85	10,55	-	26,33	10,77	-
	6000	29,09	27,48	-	44,08	16,76	-	43,07	18,11	-
	7000	43,05	45,63	-	85,31	24,65	-	68,86	21,71	-
	8000	72,48	58,42	-	88,91	31,54	-	86,72	30,49	-
	9000	92,18	95,53	-	151,81	36,85	-	116,02	36,87	-
	10000	111,08	116,67	-	190,55	48,09	-	155,1	43,41	-
dense (> 70 %)	**1000**	0,62	0,67	-	0,62	0,6	-	0,6	0,63	-
	2000	3,3	2,95	-	3,59	2,26	-	3,62	2,38	-
	3000	10,46	8,43	-	11,65	4,99	-	11,61	5,51	-
	4000	25,64	16,75	-	27,53	9,38	-	26,62	9,92	-
	5000	43,85	29,4	-	51,63	15,22	-	51,03	15,89	-
	6000	104,47	59,28	-	101,14	22,69	-	92,41	26,09	-
	7000	121,14	91,75	-	166,53	38,52	-	142,65	31,19	-
	8000	220,08	129,7	-	219,71	40,28	-	216,43	43,31	-
	9000	284,63	175,07	-	331,37	52,81	-	293,18	52,44	-
	10000	383,98	248,97	-	423,32	65,31	-	411,29	64,93	-



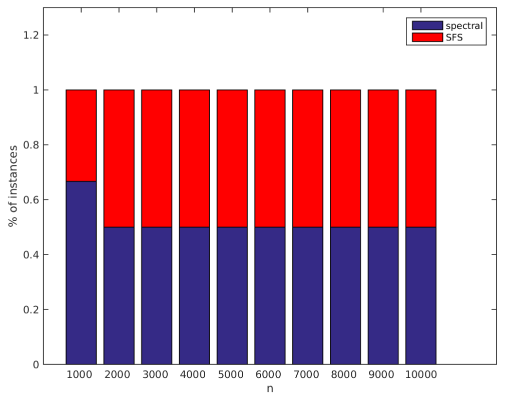
(a) sparse - low



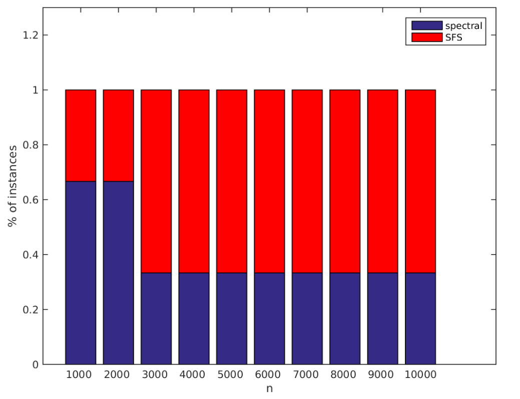
(b) sparse - medium



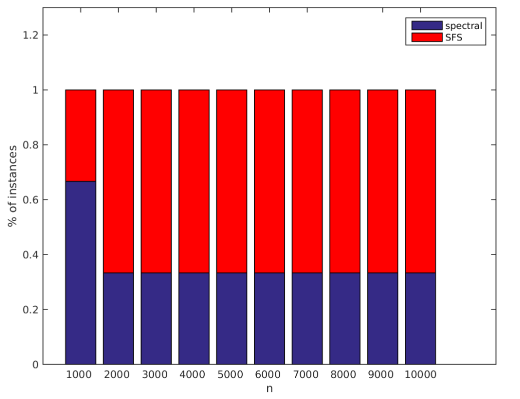
(c) sparse - high



(d) sparse - low

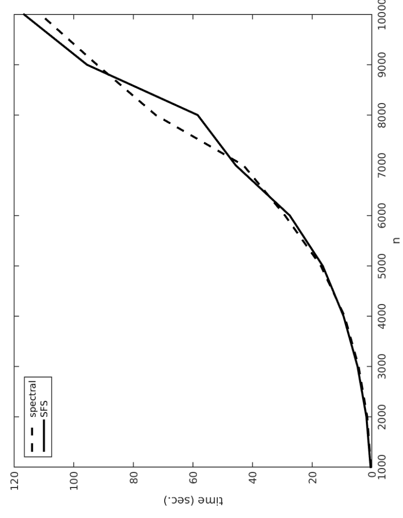


(e) sparse - medium

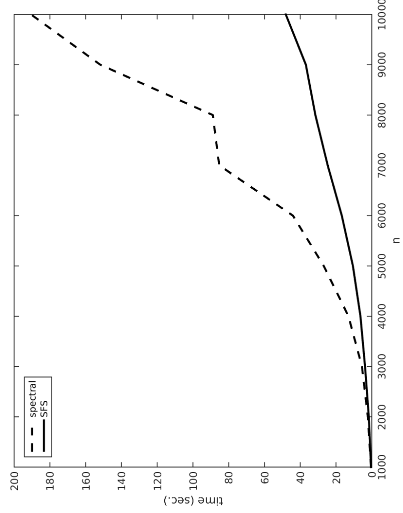


(f) sparse - high

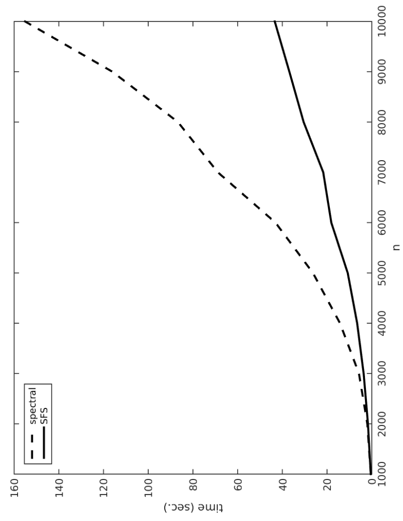
Figure 9.5: Time performance of the algorithms for *sparse* Robinsonian matrices (large instances).



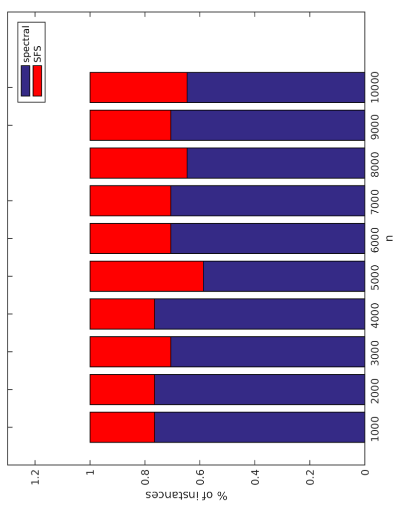
(a) normal - low



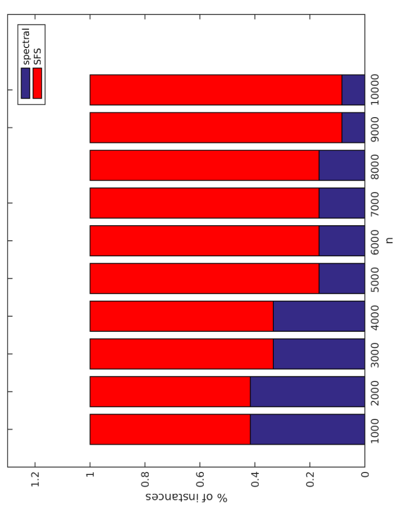
(b) normal - medium



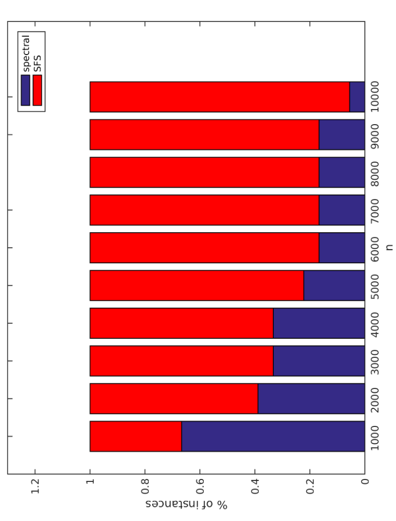
(c) normal - high



(d) normal - low

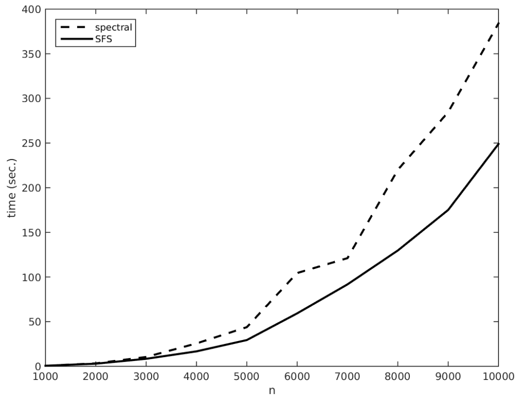


(e) normal - medium

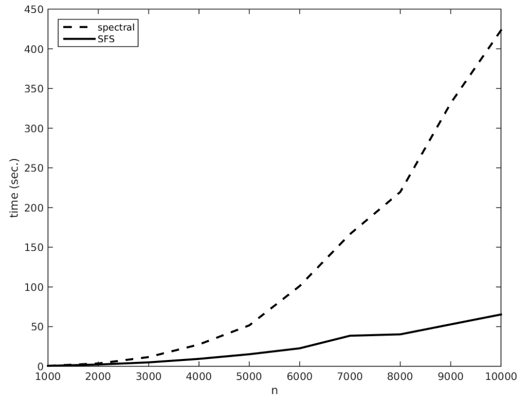


(f) normal - high

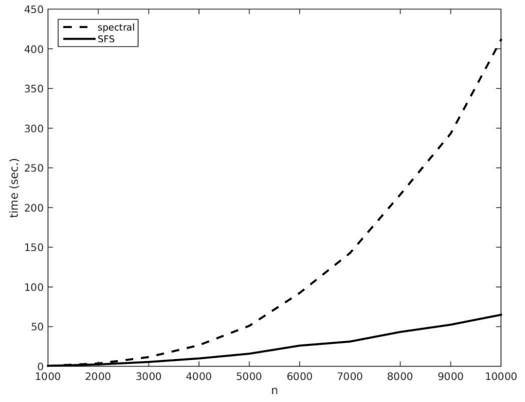
Figure 9.6: Time performance of the algorithms for *normal* Robinsonian matrices (large instances)



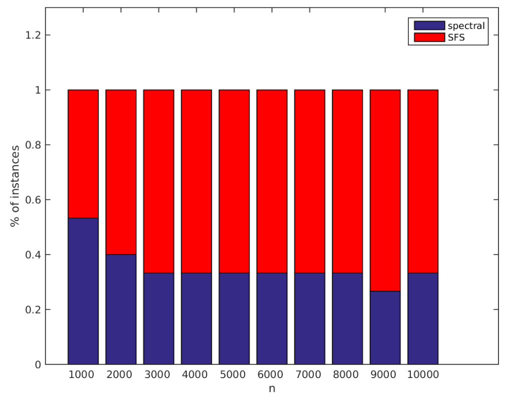
(a) dense - low



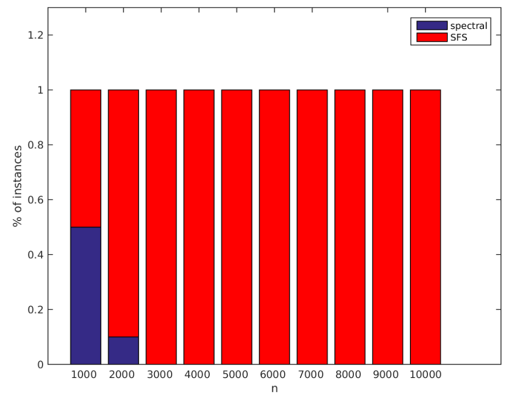
(b) dense - medium



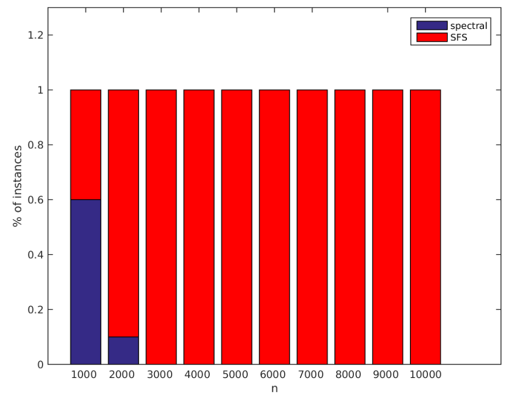
(c) dense - high



(d) dense - low



(e) dense - medium



(f) dense - high

Figure 9.7: Time performance of the algorithms for *dense* Robinsonian matrices (large instances)

9.3 Results with non-Robinsonian matrices

In this section we discuss the computational results for the seriation problem when the data does not have the Robinsonian structure. We then say that the matrix is affected by error. The results are divided in two parts: in the first part we give the computational times of the algorithms for verifying non-Robinsonian matrices; in the second part we give instead some results related to the quality of the solution returned by the algorithms with respect to Robinsonian approximation.

As already discussed in Subsection 9.1.3, we run the experiments for non-Robinsonian matrices with at most 200 distinct values, as we needed to test around 8000 instances and **SFS** and **LBFS** needed more than a minute per instance to terminate.

Non-Robinsonian verification In what follows, we will use the same notation as in Section 9.2 to refer to the algorithms. In Section 9.2 we studied the performance of the algorithms to verify the Robinsonian structure, i.e., given a Robinsonian matrix, return a Robinson ordering. Here, instead, we study which algorithm is faster in detecting non-Robinsonian matrices, i.e., given a non-Robinsonian matrix, find an obstruction to the Robinsonian structure which permits to state that the matrix is not Robinsonian. Analogously as above, we show in Table 9.4 the average computational time of **spectral**, **SFS** and **LBFS** for matrices grouped according to their density, number of distinct values and size. Analyzing the data, **LBFS** outperforms **SFS** almost in all the groups of matrices. For example, to check if a 1000×1000 matrix with *medium* number of distinct values is non-Robinsonian, **LBFS** needs in average 2 seconds, while **SFS** takes 53 seconds. Although faster than **SFS**, **LBFS** is still not comparable with **spectral**, which is often three times faster than **LBFS** and even ten times faster than **LBFS**. The most time consuming instance for **spectral** was a 1000×1000 matrix with 23 distinct values and 100% density, which required around 1,7 seconds. In comparison, for the same instance, **LBFS** required 2 seconds, while **SFS** required 2,4 seconds. The most time consuming instance for **LBFS** was a 1000×1000 matrix with 221 distinct values and 100% density, which required around 5 seconds, with a depth of the recursion tree equal to 2. In comparison, for the same instance, **spectral** required only 0,5 seconds, while **SFS** required 1,7 seconds. It is worth to notice that the largest depth of the recursion tree for **LBFS** was 20. Nevertheless, for almost 99% of instances, the depth was at most 5. Finally, the most time consuming instance for **SFS** was a 1000×1000 matrix with 22 distinct values and 100% density, which required around 7 minutes, with a total number of 999 sweeps. In comparison, for the same instance, **spectral** required only 0,5 seconds, while **LBFS** required 2,3 seconds.

It is important to notice that, in order to improve the efficiency of **SFS**, instead of checking if every sweep is a Robinson ordering, we actually implement an additional check in Algorithm 6.3, which has been used also in [51]. Specifically, every time we compute an SFS ordering σ_i with $i \geq 1$, we check if $\sigma_i = \sigma_{i-2}$. If this is the case, then we stop. In fact, if σ_{i-2} is Robinson, then in view of Lemma 6.4.2 we have that $\sigma_{i-1} = \overline{\sigma}_{i-2}$ and $\sigma_i = \overline{\sigma}_{i-1} = \sigma_{i-2}$, and thus σ_i is also a Robinson ordering. On the other hand, if σ_{i-2} is not Robinson, then it is easy to see that $\sigma_{i-1} = \sigma_{i+1}$. Hence, the SFS multisweep algorithm will loop between σ_{i-2} and σ_i until $n-1$ sweeps are reached. Therefore, we stop it immediately, without performing other sweeps. This caution permits to bound the number of sweeps, which in almost 93% of instances is less than 20. Nevertheless, there exist instances where the SFS orderings do not loop, reaching the maximum number of sweeps (i.e., $n-1$). Quite interestingly, the computational time for **SFS** seems to decrease for increasing number of distinct values. For example, if we look at 1000×1000 *normal* matrices, **SFS** needed in average 6,5 seconds for matrices with less than 50 distinct values, 2,5 seconds for matrices with at least 51 and at most 200 distinct values, and 0,8 seconds for matrices with at least 201 distinct values. The same behavior can be observed for dense matrices. To conclude, the results seem to suggest that **spectral** or **LBFS** would me more suitable than **SFS** to detect non-Robinsonian matrices. Nevertheless, given the numerical errors encountered in **spectral** for the computation of eigenvalues (see Section 9.2), an interesting approach to solve seriation would be to first run **spectral**, and then to use the order returned by **spectral** to break ties in the first sweep of **SFS**. Because the number of SFS sweeps highly depends on the initial order of the vertices, the order returned by **spectral** could be thus used as preprocessing step to obtain a smaller number of SFS sweeps.

Robinsonian approximation In what follows, we will use the same notation as in Section 9.2 to refer to the algorithms. The algorithm **LBFS** is the same as described in Chapter 5, with the only difference that we do not stop the algorithm in case one of the routines does not successfully terminates. Furthermore, when referring to **SFS**, this time we mean the ϵ-SFS multisweep algorithm presented in Chapter 8 (Algorithm 8.3). As this last algorithm involves computing the set Δ of all the differences of the values of the matrix (see Subsection 8.3.3) and repeats the SFS multisweep algorithm several times. In this section we will focus more on the quality of the solution returned by the algorithms rather than their time performance (as **spectral** largely outperforms both **SFS** and **LBFS**).

Recall that we denote by $\mathbf{p}_e$ the probability that an entry is affected by error and by $\mathbf{e}$ is the intensity of such error. Then, given a Robinsonian matrix A with largest value $\boldsymbol{\alpha_L}$ generated as described in Subsection 9.1.2, we obtain a matrix A_ϵ by perturbing the entries of the original Robinsonian matrix. Namely, $(A_\epsilon)_{ij} = A_{ij} + \xi_{ij}$, where $\xi_{ij} \in [1, \epsilon]$ with $\epsilon = \lfloor \mathbf{e} \cdot \boldsymbol{\alpha_L} \rfloor$.

Hence, ϵ represents an upper bound on the optimal error ϵ_{opt} of l_∞-FITTING-BY-ROBINSONIAN (8.3). We distinguish the results by the magnitude of the

Table 9.4: (Average) Time performance of the algorithms to recognize non-Robinsonian matrices (in milliseconds).

	# distinct values	**low** (≤ 50)			**medium** (> 50 and ≤ 200)			**high** (≥ 200)		
# nonzero entries	**algorithms** / **n**	**spectral**	**SFS**	**LBFS**	**spectral**	**SFS**	**LBFS**	**spectral**	**SFS**	**LBFS**
sparse (≤ 30 %)	**100**	3,38	3,19	3,97	3,65	3,26	8,70	2,45	3,53	24,71
	200	7,95	12,81	13,46	8,26	13,13	17,29	8,94	14,44	28,50
	300	18,33	26,74	26,72	14,66	27,85	38,77	15,61	27,91	54,74
	400	30,24	46,61	55,89	29,49	48,00	62,50	26,61	47,86	66,35
	500	39,21	71,35	73,08	43,32	74,78	86,73	39,56	75,01	123,39
	600	51,73	101,66	102,86	60,42	107,97	124,79	52,85	108,44	143,53
	700	72,44	139,13	139,61	74,04	144,80	164,43	71,49	147,85	205,54
	800	128,95	179,96	184,48	95,14	194,42	178,50	93,55	193,13	180,51
	900	127,64	225,54	240,81	127,92	244,19	262,55	121,03	249,03	296,24
	1000	163,96	296,93	325,84	160,30	301,74	300,25	167,75	302,62	331,94
normal (> 30 % and ≤ 70%)	**100**	3,46	8,31	9,10	3,29	8,56	13,54	3,39	8,91	21,55
	200	11,33	75,93	31,70	11,88	32,17	36,02	11,89	33,41	39,96
	300	24,06	66,82	66,93	25,77	71,00	72,04	25,00	69,49	74,70
	400	52,86	124,91	136,50	53,85	126,70	133,11	53,05	126,13	132,55
	500	77,73	199,92	194,99	72,52	568,96	190,62	79,54	205,13	189,14
	600	118,72	281,08	279,01	121,22	285,78	273,46	123,48	281,48	267,90
	700	158,05	1429,76	384,47	162,14	908,15	379,71	156,37	401,34	374,50
	800	240,88	2026,66	561,99	238,48	2168,65	516,81	214,83	523,01	500,15
	900	305,65	4710,79	689,49	327,93	686,22	682,06	334,72	673,17	654,22
	1000	367,81	6486,19	883,41	392,26	2548,73	871,23	404,05	825,35	895,90
dense (> 70 %)	**100**	3,84	23,48	14,68	3,99	16,59	27,57	4,17	14,73	44,46
	200	13,57	211,20	52,92	14,11	97,96	76,25	14,53	83,24	125,05
	300	33,08	1099,70	118,34	35,26	354,78	149,28	34,08	315,20	190,26
	400	73,82	3693,10	243,17	68,34	2050,41	270,97	74,06	472,07	343,39
	500	114,81	8455,90	366,58	110,63	2971,21	409,58	109,28	1777,49	534,60
	600	180,06	18100,80	553,47	167,60	6716,78	601,30	173,72	3711,67	664,55
	700	280,26	24185,30	785,59	240,16	16905,70	842,70	247,22	9713,86	1023,74
	800	404,85	59160,50	1147,25	352,20	22751,90	1166,28	326,20	22186,60	1363,94
	900	533,38	88776,30	1492,49	488,54	34155,10	1632,73	443,13	28937,10	1621,75
	1000	679,39	129063,00	1941,58	621,02	53091,40	2088,94	618,87	40281,20	2213,26

error ϵ and by its frequency [2]. Specifically, we say that an error ϵ is:

(i) *small* if $\epsilon \leq 0.05 \cdot \alpha_L$;

(ii) *large* if $\epsilon > 0.05 \cdot \alpha_L$;

(iii) *rare* if it occurs with probability $\mathbf{p}_e$ at most 10%;

(iv) *frequent* if it occurs with probability $\mathbf{p}_e$ higher than 10%.

In order to assess the quality of the solutions returned (i.e., a permutation π) by the algorithms, we use two criteria:

1) We evaluate the objective function of 2-SUM in (7.2), i.e.,

$$\sum_{i,j=1}^{n} A_{\pi(i),\pi(j)}|i-j|^2. \tag{9.3}$$

In view of Theorem 7.2.6, if A is Robinsonian then a Robinson ordering is optimal for (9.3). Hence, the idea is to check which permutations lead to the minimum value for the objective function in (9.3).

2) We compute the best Robinson approximation matrix $((A_\epsilon)_\pi)^*$ of $(A_\epsilon)_\pi$ as in (8.10) and the optimal value ϵ^*_π of the corresponding l_∞-FITTING-BY-ROBINSON problem (8.4), i.e.,:

$$\epsilon^*_\pi = \|(A_\epsilon)_\pi - ((A_\epsilon)_\pi)^*\|_\infty. \tag{9.4}$$

This gives an upper bound on the optimal value ϵ_{opt} of l_∞-FITTING-BY-ROBINSONIAN (8.3). Hence, the idea is to check which permutation π leads to the closest Robinsonian approximation of the non-Robinsonian matrix A_ϵ, i.e., the smallest value in (9.4). To do so, we consider, for each permutation π, the (approximation) ratio $\frac{\epsilon^*_\pi}{\epsilon}$. The reason we use ϵ and not ϵ_{opt} is that ϵ_{opt} is not known, while ϵ can be computed as $\epsilon = \|A - A_\epsilon\|_\infty$. Then, the closer the ratio is to one, the better the quality of the permutation π, because it means that the original perturbation was retrieved. Recall that $\epsilon > 0$ by construction, because instances with null error were discarded (see Subsection 9.1.3). Note that the ratio can be zero, because there are cases where $\epsilon^*_\pi = 0$ (see Figure 9.9), i.e., when A_ϵ is still Robinsonian.

[2] In fact ϵ is recomputed afterwards with the value of the actual error, i.e., $\epsilon = \|A - A_\epsilon\|_\infty$.

Analogously to Section 9.2, we group the matrices for each possible combination of error magnitude (*small*, *large*) and error frequency (*rare*, *frequent*), thus four possible scenarios.

The results corresponding to the first criterion are illustrated in Figure 9.8. For each algorithm, we denote the number of instances (in percentage) for which a given algorithm achieves the best value in (7.2). For example, for matrices with *small* and *rare* error (see Figure 9.8a) and $n = 100$, in roughly 70% of instances, **spectral** achieves a strictly better solution in (9.3) with respect to the other two algorithms and, in roughly 10% of instances, **SFS** achieves a strictly better solution in (9.3) with respect to the other two algorithms. Furthermore, in roughly 10% of instances, both **spectral** and **SFS** achieve solutions with the same value in (9.3), which is strictly better than the solution of **LBFS**. Finally, in the remaining 10% of instances, all three algorithms achieves solutions with the same value in (9.3).

The results corresponding to the second criterion are illustrated in Figures 9.9 and 9.10. In this case, the bar chart represents the distribution of the (approximation) ratio ϵ^*_π/ϵ. For example, for matrices with *small* and *rare* error, the distribution of the ratio for **spectral** is given in Figure 9.9a. Hence, we can see that in 50% of the instances, the ratio is 1, in 18% of instances it is 2 and so on. The same reasoning can be done for **SFS** and **LBFS**.

We comment below on the computational results for each algorithm.

Spectral algorithm Also when dealing with non-Robinsonian matrices, **spectral** performs really good with respect to both criteria. If we look at Figure 9.8, for any scenario, in more than 80% of cases it achieves the minimum value in 2-SUM (9.3). Furthermore, it returns also a close Robinsonian approximation. In fact, for 80% of instances, the approximation ratio with respect to ϵ is at most 5, and for 70% of instances it is at most 2. However, in the worst case, the largest approximation error is up to 25 times bigger than ϵ (for matrices with small errors, see Figure 9.9).

ϵ-SFS multisweep algorithm The algorithm does not perform as good as **spectral**, but it outperforms **LBFS**. In all scenarios, for around 20% of instances, it achieves the minimum value in 2-SUM (9.3), and the rate of success seems to increase for increasing sizes n of the matrix (see Figure 9.8). The Robinsonian approximation is not as close as **spectral**. In fact, in this case the approximation ratio with respect to ϵ is at most 5 for 70-80% of instances, while it is at most 2 for only 20-30% of the instances. However, we can see that the largest approximation error with respect to ϵ is only 18 (for matrices with small and frequent error, see Figure 9.9), while for **spectral** it was 25. Furthermore, for matrices with large error, the largest approximation error is 9, while in **spectral** it is 11 (see Figure 9.10).

LBFS-based algorithm For non-Robinsonian matrices, **LBFS** has the worst performance among the three algorithms with respect to both criteria. For any scenario, it achieves the minimum value for in 2-SUM (9.3) most of the time together with the other two companion algorithms, and rarely alone (see Figure 9.8). The largest approximation error returned by the algorithm with respect to ϵ is equal to 22 (for matrices with small and frequent error, see Figure 9.9), which is in between the one of **spectral** (25) and the one of **SFS** (18).

9.4 Conclusions

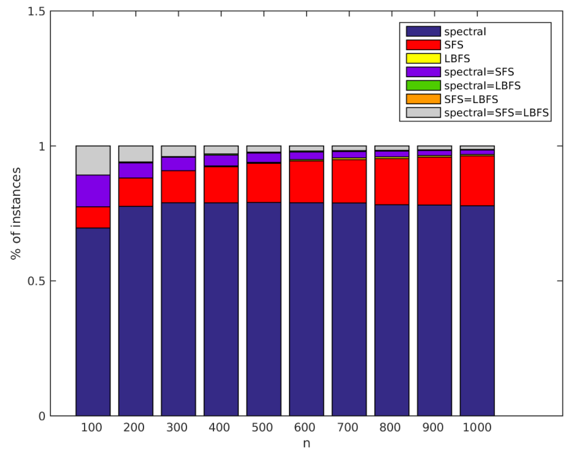
In this chapter we have discussed the performance of Algorithm 5.5, Algorithm 6.3 and Algorithm 8.3 with respect to the spectral algorithm introduced by Atkins *et al.* [5]. We generated several classes of Robinsonian matrices, and we perturbed them in order to obtain instances for the seriation problem. We used four different generation methods and five different parameters to vary the structure of the matrices. Of course, the matrices obtained represent a subclass of Robinsonian matrices. Nevertheless, the results can give an indicative idea on how the algorithms work.

For the recognition of Robinsonian matrices, the SFS multisweep algorithm presented in Chapter 6 (Algorithm 6.3) is much faster than the Lex-BFS based algorithm presented in Chapter 5 (Algorithm 5.5) when applied to Robinsonian matrices. To give an idea, in average the spectral and the SFS multisweep algorithms run in less than a second even for 1000×1000 dense matrices, while the Lex-BFS based algorithm in comparison needs more than 8 seconds. One of the most time consuming tasks of the Lex-BFS based algorithm is reordering the matrix according to a given linear order, which we have to compute three times per level graph. If the depth of the recursion tree is large, the multiplication factor in the complexity (see Theorem 5.4.4) leads to significantly slow computational time (up to 2 minutes) compared with the other two algorithms, which instead terminate in few seconds or even milliseconds (see Table 9.2). Also the SFS multisweep algorithm faces the same issue of reordering the matrix for each sweep. However, because the number of sweeps is at most 4 for all the instances, the matrix is reordered only at most four times. However, if for some 'bad' instances the number of sweeps would be larger, this would affect the time performance of the SFS multisweep algorithm as well. This is the case, e.g., when the input matrix is non-Robinsonian. Here, the SFS multisweep algorithm performs worse compared with the Lex-BFS based algorithm and the spectral algorithm, as in some instances, $n-1$ sweeps are required, leading to high computational times. Nevertheless, the SFS multisweep algorithm does not encounter numerical errors as in the generation of the Fiedler vector for the spectral algorithm, which, as we have seen, can lead to non-Robinson orderings even if the matrix is Robinsonian. Furthermore, when the input matrix is Robinsonian, the SFS multisweep algorithm performance compares well with respect to the spectral algorithm in [5], which is a numerical method that can be implemented extremely fast due to state-of-the-art algebraic libraries to compute eigenvalues/eigenvectors of (sparse) matrices (e.g., *Armadillo* [104] for C++). From the experiments for large instances given in Table 9.3, the SFS multisweep algorithm significantly outperforms the spectral algorithm in all the instances (on average), which makes it suitable for practical applications.

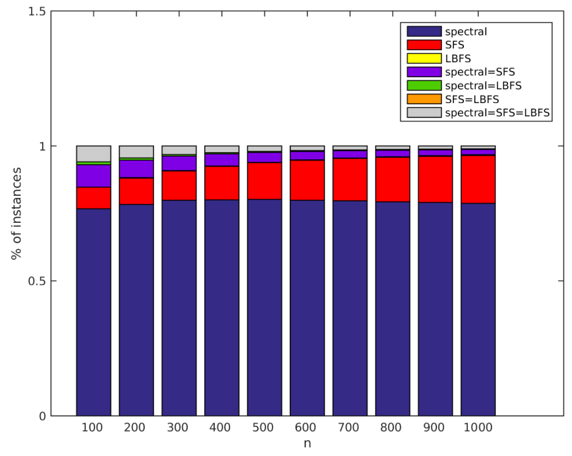
In addition, for the seriation problem when the similarity matrix is not Robinsonian, we compared the algorithms in terms of the quality of their returned permutation, i.e., the evaluation in the 2-SUM problem (9.3) and the value of l_∞-FITTING-BY-ROBINSONIAN (8.3), which reflects how close is the non-Robinsonian matrix to a Robinsonian matrix. In both cases, the ϵ-SFS multisweep algorithm introduced in Chapter 8 (Algorithm 8.3) outperforms the Lex-BFS based algorithm. However, this time the ϵ-SFS multisweep algorithm performs a bit worse compared with the spectral algorithm, which often leads to a better permutation. Nevertheless, the work done in Chapter 8 is preliminary and its theoretical foundation has not yet been investigated as was done for the SFS multisweep algorithm in Chapter 6.

In summary, among the two main recognition algorithms introduced in this thesis, the SFS multisweep algorithm (presented in Chapter 6) seems more promising than the Lex-BFS-based algorithm (presented in Chapter 5). To some extent, in fact, the SFS multisweep algorithm represents a 'smarter' evolution of the Lex-BFS-based algorithm, in the sense that, with (almost) its same subroutines, it is able to recognize a Robinsonian matrix without the need to decompose it over its level graphs. The computational experiments show that, when the input matrix is Robinsonian, the SFS multisweep algorithm is more efficient than the Lex-BFS-based algorithm and that, on medium size matrices, it is comparable with the spectral method of Atkins *et al.* in [5], which is a common and extremely fast algorithm used to solve the seriation problem. Furthermore, the SFS multisweep algorithm outperforms the spectral algorithm for large instances with a number of rows/columns $n \geq 1000$ (tested with n up to 10000). Moreover, it does not encounter numerical errors as in the computation of the Fiedler vector for the spectral algorithm. However, when the input matrix is non-Robinsonian, then the SFS multisweep algorithm is slower (with respect to the other two algorithms), due to the high number of sweeps. Therefore, a possible heuristic to solve seriation would be to combine the spectral algorithm and the SFS multisweep algorithm as follows. We first run the spectral algorithm as preprocessing step, and we use its linear order returned as initial ordering for the SFS multisweep algorithm. The idea behind this heuristic is that, independently of the fact that the input matrix is Robinsonian or not, the linear order returned by the spectral algorithm represents a good 'Robinson ordering candidate'. Therefore, this approach could potentially lead to a smaller number of SFS sweeps for the recognition of Robinsonian matrices, and thus would make SFS potentially suitable for applications of seriation in real world problems.

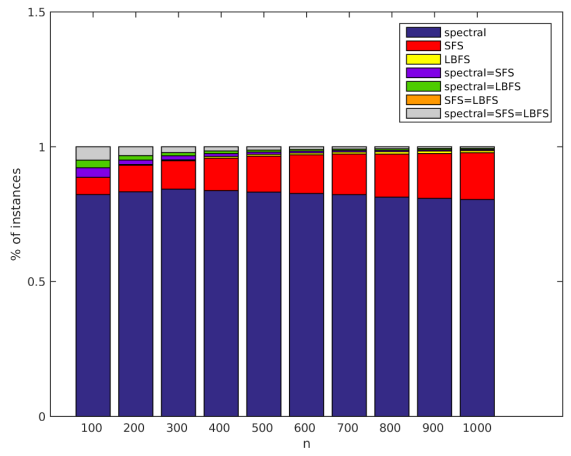
To conclude, we believe that the SFS multisweep algorithm can be a powerful algorithm to solve the seriation problem and that the SFS algorithm can be used as basic flexible routine to solve other combinatorial problems. In fact, as already discussed in Chapter 6, one could try to extend some famous graph classes recognizable using Lex-BFS (e.g., interval and chordal graphs) to the corresponding ‘weighted version’. Because SFS is the generalization of Lex-BFS to weighted graphs, it could be potentially used for the recognition of these new classes of matrices.



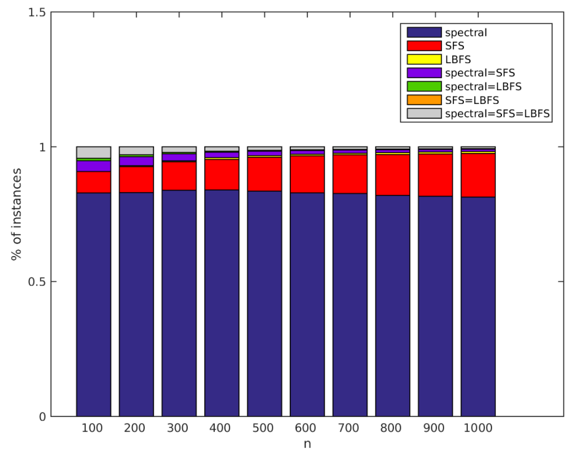
(a) small - rare



(b) small - frequent

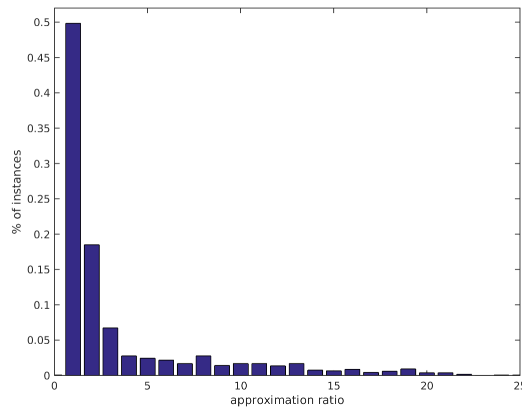


(c) large - rare

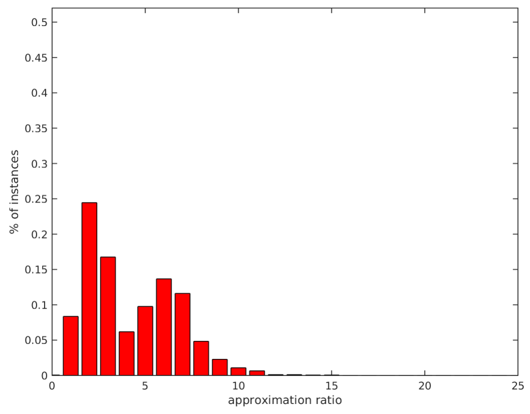


(d) large - frequent

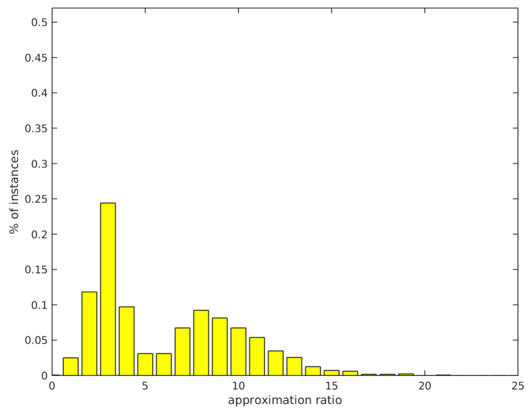
Figure 9.8: Algorithms performance in terms of minimum value attained for 2-SUM (9.3) for non-Robinsonian matrices.



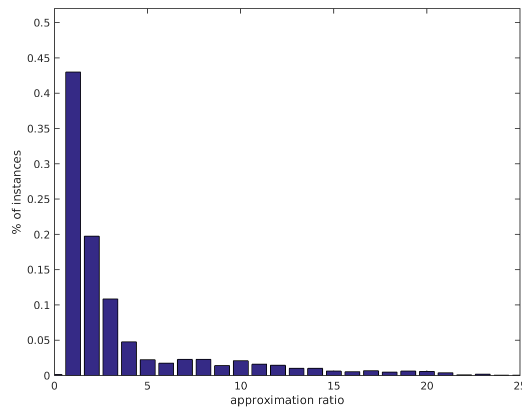
(a) small - rare (**spectral**)



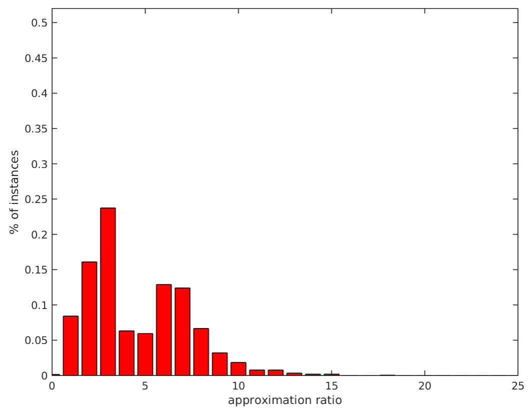
(b) small - rare (**SFS**)



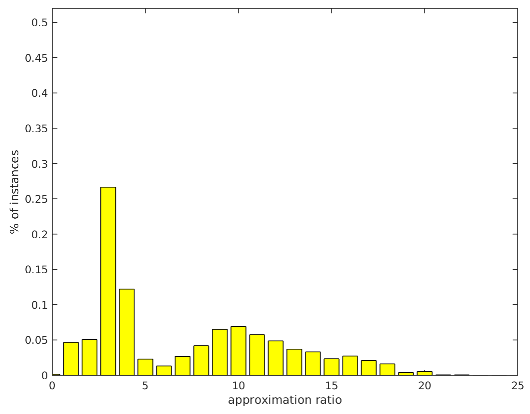
(c) small - rare (**LBFS**)



(d) small - frequent (**spectral**)

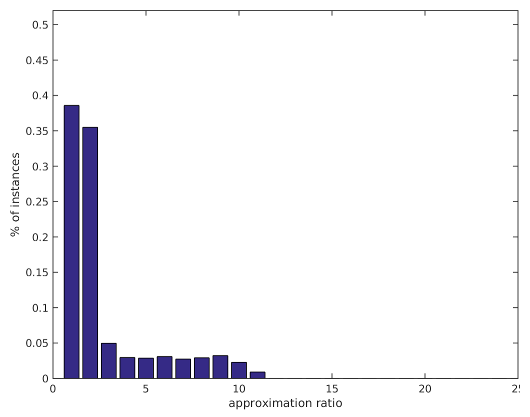


(e) small - frequent (**SFS**)

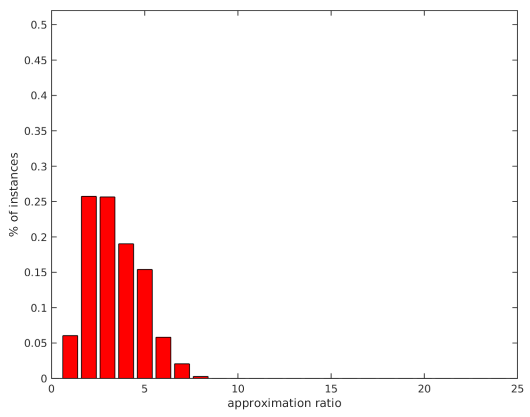


(f) small - frequent (**LBFS**)

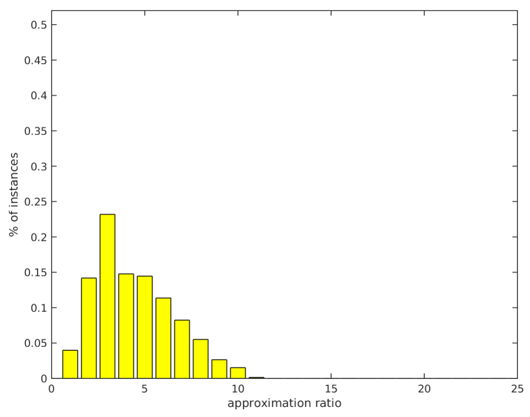
Figure 9.9: Algorithms performance in terms of approximation ratio between ϵ_{π}^{*} and ϵ for non-Robinsonian matrices with *small* error.



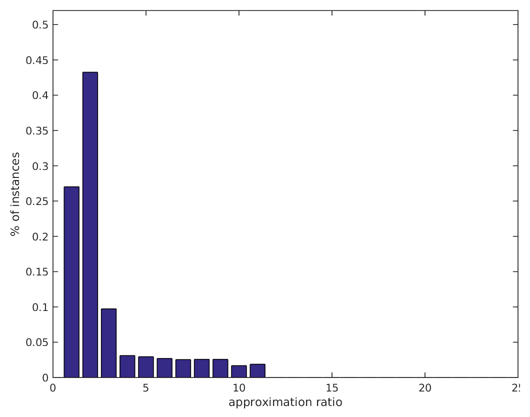
(a) large - rare (**spectral**)



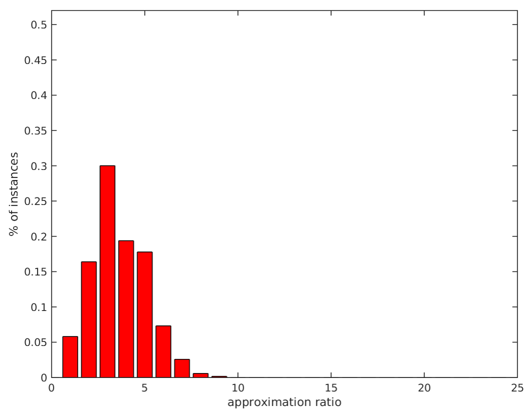
(b) large - rare (**SFS**)



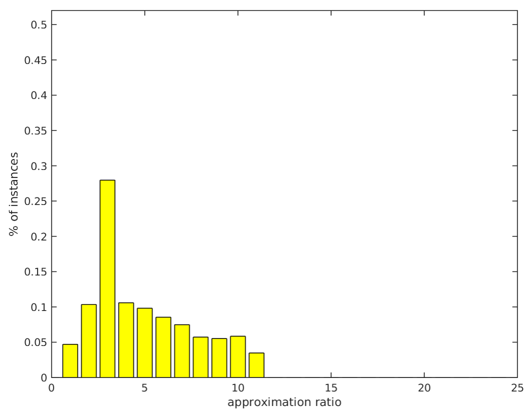
(c) large - rare (**LBFS**)



(d) large - frequent (**spectral**)



(e) large - frequent (**SFS**)



(f) large - frequent (**LBFS**)

Figure 9.10: Algorithms performance in terms of approximation ratio between ϵ^*_π and ϵ for non-Robinsonian matrices with *large* error.

Bibliography

[1] A. Abbott. Sequence analysis: New methods for old ideas. *Annual Review of Sociology*, 21:93–113, 1995.

[2] R. Agarwala, V. Bafna, M. Farach, B. Narayanan, M. Paterson, and M. Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, pages 365–372. Society for Industrial and Applied Mathematics, 1996.

[3] F. Annexstein and R. Swaminathan. On testing consecutive–ones property in parallel. *Discrete Applied Mathematics*, 88(13):7–28, 1998.

[4] P. Arabie and L.J. Hubert. The bond energy algorithm revisited. *IEEE Transactions on Systems, Man and Cybernetics*, 20(1):268–274, 1990.

[5] J.E. Atkins, E.G. Boman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28:297–310, 1998.

[6] J-C. Aude, Y. Diaz-Lazcoz, J-J. Codani, and J-L. Risler. Applications of the pyramidal clustering method to biological objects. *Computers and Chemistry*, 23(3–4):303–315, 1999.

[7] S.T. Barnard, A. Pothen, and H.D. Simon. A spectral algorithm for envelope reduction of sparse matrices. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Supercomputing '93, pages 493–502, 1993.

[8] J.P. Barthélemy and F. Brucker. Np-hard approximation problems in overlapping clustering. *Journal of Classification*, 18(2):159–183, 2001.

[9] E. Becker, B. Robisson, C-E Chapple, A. Gunoche, and C. Brun. Multifunctional proteins revealed by overlapping clustering in protein interaction network. *Bioinformatics*, 28(1):84–90, 2012.

[10] P. Bertrand and J. Diatta. Prepyramidal clustering and robinsonian dissimilarities: one-to-one correspondences. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(4):290–297, 2013.

[11] P. Bertrand and M.F. Janowitz. Pyramids and weak hierarchies in the ordinal model for clustering. *Discrete Applied Mathematics*, 122(13):55–81, 2002.

[12] R. Bevern, C. Komusiewicz, H. Moser, and R. Niedermeier. Measuring indifference: Unit interval vertex deletion. In *Graph Theoretic Concepts in Computer Science: 36th International Workshop*, volume 6410 of *Lecture Notes in Computer Science*, pages 232–243. Springer Berlin Heidelberg, 2010.

[13] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Revista Facultad de Ciencias Exactas, Puras y Aplicadas Universidad Nacional de Tucuman, Serie A*, 5:147–151, 1946.

[14] K.S. Booth and G.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.

[15] A. Brandstädt, F.F. Dragan, and F. Nicolai. LexBFS-orderings and powers of chordal graphs. *Discrete Mathematics*, 171(13):27–42, 1997.

[16] A. Bretscher, D.G. Corneil, M. Habib, and C. Paul. A simple linear time LexBFS cograph recognition algorithm. *SIAM Journal on Discrete Mathematics*, 22:1277–1296, 2008.

[17] M.J. Brusco. A branch-and-bound algorithm for fitting anti-robinson structures to symmetric dissimilarity matrices. *Psychometrika*, 67(3):459–471, 2002.

[18] M.J. Brusco and S. Stahl. *Branch-and-Bound Applications in Combinatorial Data Analysis*. Statistics and Computing. Springer New York, 2005.

[19] E.R. Burkard, S.E. Karisch, and F. Rendl. Qaplib – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, 1997.

[20] R.E. Burkard, E. Çela, V.M. Demidenko, N.N. Metelski, and G.J. Woeginger. A unified approach to simple special cases of extremal permutation problems. *Optimization*, 44(2):123–138, 1998.

[21] R.E. Burkard, E. Çela, P.M. Pardalos, and L.S. Pitsoulis. The quadratic assignment problem. In *Handbook of Combinatorial Optimization*, pages 1713–1809. Springer US, 1999.

[22] R.E. Burkard, E. Çela, G. Rote, and G.J. Woeginger. The quadratic assignment problem with a monotone anti-monge and a symmetric Toeplitz matrix: Easy and hard cases. *Mathematical Programming*, 82(1-2):125–158, 1998.

[23] R.E. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems.* Society for Industrial and Applied Mathematics, 2009.

[24] G. Caraux and S. Pinloche. PermutMatrix: a graphical environment to arrange gene expression profiles in optimal linear order. *Bioinformatics*, 21(7):1280–1281, 2005.

[25] E. Çela, V.G. Deineko, and G.J. Woeginger. A new tractable case of the QAP with a Robinson matrix. In *Combinatorial Optimization and Applications*, volume 9486 of *Lecture Notes in Computer Science*, pages 709–720. Springer International Publishing, 2015.

[26] E. Çela, V.G. Deineko, and G.J. Woeginger. Well-solvable cases of the QAP with block-structured matrices. *Discrete Applied Mathematics*, 186:56–65, 2015.

[27] C.H. Chen. Generalized association plots: information visualization via iteratively generated correlation matrices. *Statistica Sinica*, 12:7–29, 2002.

[28] V. Chepoi and B. Fichet. Recognition of Robinsonian dissimilarities. *Journal of Classification*, 14(2):311–325, 1997.

[29] V. Chepoi, B. Fichet, and M. Seston. Seriation in the presence of errors: Np-hardness of l_∞-fitting Robinson structures to dissimilarity matrices. *Journal of Classification*, 26(3):279–296, 2009.

[30] V. Chepoi and M. Seston. Seriation in the presence of errors: A factor 16 approximation algorithm for l_∞-fitting Robinson structures to distances. *Algorithmica*, 59(4):521–568, 2011.

[31] G. Christopher, M. Farach, and M. Trick. The structure of circular decomposable metrics. In *Algorithms ESA '96*, volume 1136 of *Lecture Notes in Computer Science*, pages 486–500. Springer Berlin Heidelberg, 1996.

[32] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2009.

[33] D.G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004.

[34] D.G. Corneil. Lexicographic breadth first search - A survey. In *Graph-Theoretic Concepts in Computer Science*, volume 3353 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2005.

[35] D.G. Corneil, H. Kim, S. Natarajan, S. Olariu, and A.P. Sprague. Simple linear time recognition of unit interval graphs. *Information Processing Letters*, 55(2):99–104, 1995.

[36] D.G. Corneil, E. Khler, and J.M. Lanlignel. On end-vertices of lexicographic breadth first searches. *Discrete Applied Mathematics*, 158(5):434–443, 2010.

[37] D.G. Corneil, S. Olariu, and L. Stewart. The ultimate interval graph recognition algorithm? In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 175–180. Society for Industrial and Applied Mathematics, 1998.

[38] D.G. Corneil, S. Olariu, and L. Stewart. The LBFS structure and recognition of interval graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1905–1953, 2009.

[39] P. Crescenzi, D.G. Corneil, J. Dusart, and M. Habib. New trends for graph search. PRIMA Conference in Shanghai, June 2013. available at `http://math.sjtu.edu.cn/conference/Bannai/2013/data/20130629B/slides1.pdf`.

[40] F. Critchley and B. Fichet. The partial order by inclusion of the principal classes of dissimilarity on a finite set, and some of their basic properties. In *Classification and Dissimilarity Analysis*, pages 5–65. Springer New York, 1994.

[41] V.G. Deineko and G.J. Woeginger. A solvable case of the quadratic assignment problem. *Operations Research Letters*, 22(1):13–17, 1998.

[42] V.M. Demidenko, G. Finke, and V.S. Gordon. Well solvable cases of the quadratic assignment problem with monotone and bimonotone matrices. *Journal of Mathematical Modelling and Algorithms*, 5(2):167–187, 2006.

[43] X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM Journal on Computing*, 25(2):390–403, 1996.

[44] J. Díaz, J. Petit, and M.J. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[45] E. Diday, Y. Lechevallier, M. Schader, P. Bertrand, and B. Burtschy. New approaches in classification and data analysis. In *Studies in Classification, Data Analysis, and Knowledge Organization*. Springer, 1994.

[46] C. Ding and X. He. Linearized cluster assignment via spectral ordering. In *Proceedings of the 21st International Conference on Machine learning*, ICML '04. ACM Press, 2004.

[47] M. Dom. Algorithmic aspects of the consecutive-ones property. *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59, 2009.

[48] M. Dom, J. Guo, and R. Niedermeier. Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *Journal of Computer and System Sciences*, 76(34):204–221, 2010.

[49] C. Durand and B. Fichet. Orders and overlapping clusters in pyramids. In *Multidimensional Data Analysis*, pages 201–234. DSWO Press, 1986.

[50] C. Durand and B. Fichet. One-to-one correspondences in pyramidal representation: a unified approach. In *Classification and Related Methods of Data Analysis*, volume 1136, pages 85–90. 1988.

[51] J. Dusart and M. Habib. A new LBFS-based algorithm for cocomparability graph recognition. *Discrete Applied Mathematics*, to appear, 2015.

[52] D. Earle. *Dendrogram seriation in data visualisation: algorithms and applications*. PhD thesis, National University of Ireland Maynooth, 2010.

[53] B.S Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. Wiley Series in Probability and Statistics, 5th edition, 2011.

[54] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1):155–179, 1995.

[55] F. Fogel, A. d' Aspremont, and M. Vojnovic. Serialrank: Spectral ranking using seriation. In *Advances in Neural Information Processing Systems 27*, pages 900–908. 2014.

[56] F. Fogel, R. Jenatton, F. Bach, and A. d'Aspremont. Convex relaxations for permutation problems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1465–1488, 2015.

[57] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[58] A. George and A. Pothen. An analysis of spectral envelope reduction via quadratic assignment problems. *SIAM Journal on Matrix Analysis and Applications*, 18(3):706–732, 1997.

[59] P.C. Gilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

[60] J.Y. Goulermas, A. Kostopoulos, and T. Mu. A new measure for analyzing and fusing sequences of objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(5):833–848, 2015.

[61] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(12):59–84, 2000.

[62] M. Hahsler. An experimental comparison of seriation methods for one-mode two-way data. *European Journal of Operational Research*, to appear, 2016.

[63] M. Hahsler, K. Hornik, and C. Buchta. Getting things in order: An introduction to the R package seriation. *Journal of Statistical Software*, 25(1), 2008.

[64] M.T. Hajiaghayi and Y. Ganjali. A note on the consecutive ones submatrix problem. *Information Processing Letters*, 83(3):163–166, 2002.

[65] T.C. Havens and J.C. Bezdek. An efficient formulation of the improved visual assessment of cluster tendency (iVAT) algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):813–822, 2012.

[66] P. Hell and J. Huang. Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs. *SIAM Journal on Discrete Mathematics*, 18(3):554–570, 2005.

[67] P. Hell, R. Shamir, and R. Sharan. A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM Journal on Computing*, 31(1):289–305, 2002.

[68] L. Hubert and P. Arabie. The analysis of proximity matrices through sums of matrices having (anti-)robinson forms. *British Journal of Mathematical and Statistical Psychology*, 47(1):1–40, 1994.

[69] L. Hubert, P. Arabie, and J. Meulman. *Combinatorial Data Analysis.* Society for Industrial and Applied Mathematics, 2001.

[70] L. Hubert, P. Arabie, and J. Meulman. *The Structural Representation of Proximity Matrices with MATLAB.* Society for Industrial and Applied Mathematics, 2006.

[71] L. Hubert and J. Schultz. Quadratic assignment problem as a general data analysis strategy. *British Journal of Mathematical and Statistical Psychology*, 29(2):190–241, 1976.

[72] C.J. Jardine, N. Jardine, and R. Sibson. The structure and construction of taxonomic hierarchies. *Mathematical Biosciences*, 1(2):173–179, 1967.

[73] S.C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254.

[74] M. Kalmanson. Edgeconvex circuits and the traveling salesman problem. *Canadian Journal of Mathematics*, 27(5):1000–1010, 1975.

[75] D.G. Kendall. Incidence matrices, interval graphs and seriation in archeology. *Pacific Journal of Mathematics*, 28(3):565–570, 1969.

[76] T.C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76, 1957.

[77] M. Laurent and M. Seminaroti. A Lex-BFS-based recognition algorithm for Robinsonian matrices. In *Algorithms and Complexity: Proceedings of the 9th International Conference CIAC 2015*, volume 9079 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 2015.

[78] M. Laurent and M. Seminaroti. The quadratic assignment problem is easy for Robinsonian matrices with Toeplitz structure. *Operations Research Letters*, 43(1):103–109, 2015.

[79] M. Laurent and M. Seminaroti. Similarity-First Search: a new algorithm with application to Robinsonian matrix recognition. *arXiv:1601.03521*, 2016.

[80] J.M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[81] I. Liiv. Seriation and matrix reordering methods: An historical overview. *Statistical Analysis and Data Mining*, 3(2):70–91, 2010.

[82] P.J. Looges and S. Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993.

[83] A. Louis, E. Ollivier, J-C. Aude, and J-L. Risler. Massive sequence comparisons as a help in annotating genomic sequences. *Genome Research*, 11(7):1296–1303, 2001.

[84] J.F. Marcotorchino. Seriation problems:an overview. *Applied Stochastic Models and Data Analysis*, 7(2):139–151, 1991.

[85] D. Mavroeidis and E. Bingham. Enhancing the stability and efficiency of spectral ordering with partial supervision and feature selection. *Knowledge and Information Systems*, 23(2):243–265, 2010.

[86] W.T. McCormick, P.J. Schweitzer, and T.W. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 20(5):993–1009, 1972.

[87] B.G. Mirkin. *Mathematical hierarchies and biology : DIMACS workshop, November 13-15, 1996.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science. R.I. American Mathematical Society, 1997.

[88] B.G. Mirkin and S.N. Rodin. *Graphs and Genes.* Biomathematics. Springer, 1984.

[89] B.S. Mirkin. *Mathematical classification and clustering.* Nonconvex optimization and its applications. Kluwer Academic Publishers, 1996.

[90] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[91] S. Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21–25, 1991.

[92] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[93] W.M.F. Petrie. Sequences in prehistoric remains. *Journal of the Anthropological Institute of Great Britain and Ireland*, 29(3–4):295–301, 1899.

[94] P. Préa and D. Fortin. An optimal algorithm to recognize Robinsonian dissimilarities. *Journal of Classification*, 31:1–35, 2014.

[95] G.P. Quinn and M.J. Keough. *Experimental design and data analysis for biologists.* Cambridge University Press, 2002.

[96] Richard R. Streng. Classification and seriation by iterative reordering of a data matrix. In *Classification, Data Analysis, and Knowledge Organization: Models and Methods with Applications*, pages 121–130. Springer Berlin Heidelberg, 1991.

[97] F.S. Roberts. Indifference graphs. In *Proof Techniques in Graph Theory: Proceedings of the Second Ann Arbor Graph Theory Conference*, pages 139–146. Academic Press, 1969.

[98] F.S. Roberts. On the compatibility between a graph and a simple order. *Journal of Combinatorial Theory, Series B*, 11(1):28–38, 1971.

[99] F.S. Roberts. *Graph theory and its applications to problems of society.* Society for Industrial and Applied Mathematics, 1978.

[100] W.S. Robinson. A method for chronologically ordering archaeological deposits. *American Antiquity*, 16(4):293–301, 1951.

[101] D.J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597–609, 1970.

[102] D.J. Rose and R.E. Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of 7th Annual ACM Symposium on Theory of Computing*, STOC '75, pages 245–254. ACM, 1975.

[103] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[104] C. Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. 2010.

[105] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer Berlin Heidelberg, 2003.

[106] M. Seston. *Dissimilarités de Robinson: Algorithmes de Reconnaissance et d'Approximation.* PhD thesis, Université de la Méditerranée, 2008.

[107] K. Simon. A new simple linear algorithm to recognize interval graphs. In *Computational geometry-methods, algorithms and applications*, volume 553 of *Lecture Notes in Computer Science*, pages 289–308. Springer Berlin Heidelberg, 1991.

[108] A. Strehl and J. Ghosh. Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing*, 15(2), 2003.

[109] Y.-J. Tien, T.-S. Lee, H.-M. Wu, and C.-H. Chen. Methods for simultaneously identifying coherent local clusters with smooth global patterns in gene expression profiles. *BMC Bioinformatics*, 9(1):1–16, 2008.

[110] D. Tsafrir, I. Tsafrir, L. Ein-Dor, O. Zuk, D. Notterman, and E. Domany. Sorting points into neighborhoods (SPIN): data analysis and visualization by ordering distance matrices. *Bioinformatics*, 21(10):2301–2308, 2005.

[111] F. Wauthier, M. Jordan, and N. Jojic. Efficient ranking from pairwise comparisons. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *ICML '13*, pages 109–117. JMLR Workshop and Conference Proceedings, 2013.

[112] H.-M. Wu, Y.-J. Tien, and C.-H. Chen. GAP: A graphical environment for matrix visualization and cluster analysis. *Computational Statistics and Data Analysis*, 54(3):767–778, 2010.

List of symbols and abbreviations

Symbols

$[n]$	This is defined as $\{1, \dots, n\}$.
$\mathcal{P}_n$	The set of all possible permutations of $[n]$.
$\mathcal{S}^n$	The set of symmetric $n \times n$ matrices.
A	Similarity matrix.
D	Dissimilarity matrix.
$A[U]$	This is the principal submatrix of a matrix A indexed by the subset U.
$\mathrm{Tr}(A)$	The trace of a matrix A.
$\langle A, B\rangle = \mathrm{Tr}(A^\mathsf{T} B)$	The trace inner product between two symmetric matrices A and B.
e	The all-ones vector.
J_n	The $n \times n$ all-ones matrix.
$\mathbb{N}$	The set of nonnegative integers.
$\mathbb{R}^n$	The set of real vectors.
$\mathbb{R}^{n\times n}$	The set of $n \times n$ real matrices.
π	Permutation.
Π	Permutation matrix.
$A_\pi = \Pi A \Pi^\mathsf{T}$	This is the matrix obtained by symmetrically permuting the rows and columns of A according to a permutation π.
$\mathcal{T}$	PQ-Tree.
$\Phi = \psi \wedge \phi$	Common Refinement between two compatible weak linear orders ψ and ϕ.
$\mathcal{L}$	Similarity layer structure.
$G^{(\ell)}$	The ℓ-th level graph of a (similarity) matrix A.
$G^{(1)}$	The support graph of a (similarity) matrix A.

α_ℓ	The ℓ-th distinct value of a (similarity) matrix A.
α_L	The value of the largest entry in a (similarity) matrix A.
L_A	The Laplacian matrix of $A \in \mathcal{S}^n$.
y_F	Fiedler vector.
$O(\cdot)$	The big O notation.
$\|A\|_\infty$	The l_∞-norm of a matrix A.
$\|A\|_2$	The l_2-norm of a matrix A.
$\|A\|_p$	The l_p-norm of a matrix A.
ϵ^*	The optimal value of l_∞-FITTING-BY-ROBINSON (8.4).
ϵ_{opt}	The optimal value of l_∞-FITTING-BY-ROBINSONIAN (8.3).

Abbreviations

BFS	This is the abbreviation for Breadth-First Search.
DFS	This is the abbreviation for Depth-First Search.
Lex-BFS	This is the abbreviation for Lexicographic Breadth-First Search.
SFS	This is the abbreviation for Similarity-First Search.
QAP	This is the abbreviation for Quadratic Assignment Problem.
C1P	This is the abbreviation for Consecutive Ones Property
PAL	This is the abbreviation for ‘Path Avoiding Lemma’

Index

ϵ-
- anchor, 164
- multisweep, 187
- Robinson ordering, 154, 155, 159
- Robinsonian, 154, 156
  - recognition, 156, 160
- SFS, 158
  - multisweep, 158
- SFS_+, 158
- similarity partition, 158

l_∞-FITTING-BY-ROBINSON, 150, 152
l_∞-FITTING-BY-ROBINSONIAN, 149, 154, 156, 186
2-SUM, 3, 133, 186
3-vertex condition, 26, 91

adjacency
- extended matrix, 24, 37, 114
- list, 24, 53, 62, 75, 121, 122
- matrix, 24, 26, 112, 145

BFS, 46
- layers, 110

blocks, 24
- end, 63
- inner, 63

C1P, 18, 19, 24, 26, 37, 39, 40, 62
classification, 1, 35
clique, 24, 40, 56, 112
- condition, 26
- maximal, 25, 40

clustering, 1, 4
- hierarchical, 34, 35
- pyramidal, 35
- spectral, 43

common refinement, 21, 63, 64, 70, 72

dendrogram, 34, 35
DFS, 46

Fiedler
- value, 41, 133
  - simple, 42
- vector, 41, 42, 172

graph
- chordal, 25, 45, 55, 103
- indifference, 26
- intersection, 40
- interval, 25, 40, 55, 93
- level, 41, 64, 73, 74
- proper interval, 26
- support, 30, 40, 75, 161
- threshold, 38
- unit interval, 26, 37, 56, 57, 114

hierarchy, 33
hypergraph, 27
- ball, 38

  incidence matrix, 27, 39
  interval, 27, 38

LBFS, 45, 171, 173, 188
Lex-BFS, 6, 46, 47, 54, 55, 57, 66, 97
  good, 49, 56
  layers, 56
  multisweep, 53, 54
  ordering, 47, 49, 55
  pivot, 47, 51
  slice, 48, 49, 51, 55
  sweep, 69
  ties, 48, 49
Lex-BFS$_*$, 55, 128
Lex-BFS$_+$, 49, 54, 55, 107
linear order, 20
  compatible, 21
  concatenation, 20, 62, 91
  restriction, 20, 91
  reversal, 20, 62, 91

matrix
  anti-R, 31
  anti-Robinson, 31
  dissimilarity, 18
  interval-cut, 133
  Laplacian, 41, 42
  left-down monotonic, 32
  monotonically
    nondecreasing, 132
    nonincreasing, 132
  proximity, 17
  R, 30
  right-up monotonic, 33
  similarity, 3, 18
  Toeplitz, 134, 144
  vertex-clique incidence, 25, 40

neighborhood, 24, 51, 53, 123
  (weighted graph), 91, 97, 99, 121, 158
  closed, 24
  condition, 26, 69

P-node, 24, 77, 79
partition
  ordered, 21, 91, 97, 99
  refinement, 22, 49, 97, 99, 121
  similarity, 98, 99, 121
permutation, 19, 131
  identity, 19, 132, 133, 144
  matrix, 19, 37, 131
PQ-tree, 23, 26, 40, 43, 70, 77
pyramid, 35

Q-node, 24, 77, 79
QAP, 131, 135, 144, 148

Robinson
  condition, 116
  dissimilarity, 4, 144
  ordering, 4, 30, 31, 40, 43, 63, 70, 77, 92, 94, 110, 115, 120, 159
  similarity, 4, 30, 42, 144, 150
  violation, 5
Robinsonian
  approximation, 147, 152
  dissimilarity, 4, 31, 38, 39, 127, 144
    strongly, 31
  matrix, 5, 31, 167, 170
  recognition, 6, 71, 75, 113
  similarity, 4, 30, 42, 101, 114, 115, 120, 144, 149
    binary, 36
    strongly, 30

seriation, 2, 3, 29, 33, 133
  criterion, 3, 5
  measure, 3
SFS, 97, 99, 106
  3-good, 116, 118, 119
  good, 106–108, 111, 116, 118, 120
  multisweep, 113, 120, 171, 172
  ordering, 98, 99, 106, 111, 120
  pivot, 98, 121
  slice, 99, 106, 111
  sweep, 113, 120

ties, 99, 101, 122
SFS$_+$, 101, 107, 113, 122
similarity
  layers, 110, 111, 116
spectral algorithm, 41, 165, 171, 172, 187
straight enumeration, 27, 61–64, 68, 72, 74, 77

vertex
  admissible, 55
  anchor, 94, 106, 110
  good, 49, 56
  homogeneous, 92, 100
  intercepts a path, 54
  miss a path, 54
  path avoiding a, 92, 96, 103
  path missing a, 92
  pivot, 121
  simplicial, 25, 54, 55
  unvisited, 46
  valid, 55, 93
  valid (weighted graph), 93, 94
vertices
  opposite anchors, 94, 96, 106, 107
  undistinguishable, 24, 61, 69
  unrelated, 54
  unvisited, 46, 50, 98, 121

weak linear order, 21, 91
  compatible, 21, 22, 63, 64, 66, 70, 72, 74
  concatenation, 21, 73
  restriction, 21, 63
  reversal, 21, 63, 79