A Lex-BFS-based recognition algorithm for Robinsonian matrices

Monique Laurent\textsuperscript{1,2} and Matteo Seminaroti\textsuperscript{1}

\textsuperscript{1}Centrum Wiskunde & Informatica (CWI), Science Park 123, 1098 XG Amsterdam, The Netherlands
\textsuperscript{2}Tilburg University, P.O. Box 90153, 5000 LE Tilburg, The Netherlands

Abstract

Robinsonian matrices arise in the classical seriation problem and play an important role in many applications where unsorted similarity (or dissimilarity) information must be reordered. We present a new polynomial time 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 and is based essentially on lexicographic breadth-first search (Lex-BFS), using a divide-and-conquer strategy. When applied to a non-negative symmetric $n \times n$ matrix with $m$ nonzero entries and given as a weighted adjacency list, it runs in $O(d(n+m))$ time, where $d$ is the depth of the recursion tree, which is at most the number of distinct nonzero entries of $A$.

Keywords: Robinson (dis)similarity; unit interval graph; Lex-BFS; seriation; partition refinement; straight enumeration

1 Introduction

An important question in many classification problems is to find an order of a collection of objects respecting some given information about their pairwise (dis) similarities. The classic seriation problem, introduced by Robinson \cite{34} for chronological dating, asks to order objects in such a way that similar objects are ordered close to each other, and it has many applications in different fields (see \cite{25} and references therein).

A symmetric matrix $A = (A_{ij})_{i,j=1}^{n}$ is a Robinson similarity matrix if its entries decrease monotonically in the rows and columns when moving away from the main diagonal, i.e., if $A_{ik} \leq \min\{A_{ij}, A_{jk}\}$ for all $1 \leq i \leq j \leq k \leq n$. Given a set of $n$ objects to order and a symmetric matrix $A = (A_{ij})$ which represents their pairwise correlations, the seriation problem asks to find (if it exists) a permutation $\pi$ of $[n]$ so that the permuted matrix $A_\pi = (A_{\pi(i)\pi(j)})$ is a Robinson matrix. If such a permutation exists then $A$ is said to be a Robinsonian similarity, otherwise we say that data is affected by noise. The definitions extend to dissimilarity matrices: $A$ is a Robinson(ian) dissimilarity precisely when $-A$ is a Robinson(ian) similarity. Hence results can be directly transferred from one class to the other one.

Robinsonian matrices play an important role in several hard combinatorial optimization problems and recognition algorithms are important in designing heuristic and approximation.
consecutive ones property (C1P). A binary matrix has the C1P if its columns can be reordered in such a way that the ones are consecutive in each row. Then a graph $G$ is an interval graph if and only if its vertex-clique incidence matrix has C1P, where the rows are indexed by the vertices and the columns by the maximal cliques of $G$. A hypergraph $H = (V,E)$ is a generalization of the notion of graph where elements of $E$, 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 corresponding hyperedge contains the corresponding vertex. Then $H$ is an interval hypergraph if its incidence matrix has C1P, i.e., if its vertices can be ordered in such a way that hyperedges are intervals.

Given a dissimilarity matrix $A \in S^n$ and a scalar $\alpha$, the threshold graph $G_\alpha = (V,E_\alpha)$ has edge set $E_\alpha = \{(x,y) : A_{xy} \leq \alpha\}$ and, for $x \in V$, the ball $B(x,\alpha) := \{y \in V : A_{xy} \leq \alpha\}$ consists of $x$ and its neighbors in $G_\alpha$. Let $B$ denote the collection of all the balls of $A$ and $H_B$ denote the corresponding ball hypergraph, with vertex set $V = [n]$ and with $B$ as set of hyperedges. One can also build the intersection graph $G_B$ of $B$, where the balls are the vertices and connecting two vertices if the corresponding balls intersect. Most of the existing algorithms are then based on the fact that a matrix $A$ is Robinsonian if and only if the ball hypergraph $H_B$ is an interval hypergraph or, equivalently, if the intersection graph $G_B$ is an interval graph (see $[28,30]$).

Testing whether an $m \times n$ binary matrix with $f$ ones has C1P can be done in linear time $O(n + m + f)$ (see the first algorithm of Booth and Leuker $[3]$ based on PQ-trees, the survey $[14]$ and further references therein). Mirkin and Rodin $[28]$ 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 $[3]$ to check whether the incidence matrix has C1P. Later, Chepoi and Fichet $[5]$ introduced a simpler algorithm that, using a divide-an-conquer strategy and sorting the entries of $A$, improved the running time to $O(n^3)$. The same sorting preprocessing was used by Seson $[36]$, who improved the algorithm to $O(n^2 \log n)$ by constructing paths in the threshold graphs of $A$. Very recently, Préa and Fortin $[30]$ presented a more sophisticated $O(n^2)$ algorithm, which uses the fact that the maximal cliques of the graph $G_B$ are in one-to-one correspondence with the row/column indices of $A$. Roughly speaking, they use the algorithm from Booth and Leuker $[3]$ to compute a first PQ-tree which they update throughout the algorithm.

A numerical spectral algorithm was introduced earlier by Atkins et al. $[1]$ for checking whether a similarity matrix $A$ is Robinsonian, based on reordering the entries of the Fiedler eigenvector of the Laplacian matrix associated to $A$, and it runs in $O(n(T(n) + n \log n))$ time, where $T(n)$ is the complexity of computing (approximately) the eigenvalues of an $n \times n$ symmetric matrix.

In this paper we introduce a new combinatorial algorithm to recognize Robinsonian matrices, based on characterizing them in terms of straight enumerations of unit interval graphs. Unit interval graphs are a subclass of interval graphs, where the intervals labeling the vertices are required to have unit length. As is well known, they can be recognized in linear time $O(|V| + |E|)$ (see e.g. $[8]$, $[22]$ and references therein). Many of the existing algorithms are based on the
equivalence between unit interval graphs and proper interval graphs (where the intervals should be pairwise incomparable) (see [31,33]). Unit interval graphs have been recently characterized in terms of *straight enumerations*, which are special orderings of the classes of the ‘undistinguishability’ equivalence relation, calling two vertices undistinguishable if they have the same closed neighborhoods (see [9]). This leads to alternative recognition algorithms (see [8,9]), which we will use as main building block in our algorithm. Our algorithm relies indeed on the fact that a similarity matrix $A$ is Robinsonian if and only if its level graphs (the analogues for similarities of the threshold graphs for dissimilarities) admit pairwise compatible straight enumerations (see Theorem 7).

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 and no preprocessing to order the data is needed. In fact, the most difficult task carried out by our algorithm is a lexicographic breadth-first search (abbreviated Lex-BFS), which is a variant of the classic breadth-first search (BFS), where the ties in the search are broken by giving preference to those vertices whose neighbors have been visited earliest (see [35] and [19]). Following [8], we in fact use the variant Lex-BFS+ introduced by [37] to compute straight enumerations. 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 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 15). Furthermore, we can return all the permutations reordering $A$ as a Robinson matrix using a PQ-tree data structure on which we perform only a few simple operations (see Section 4.3).

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 [11]. Alternatively one could check whether the incidence matrix $M$ of the ball hypergraph of $A$ has C1P, using the Lex-BFS based algorithm of [19], in time $O(r + c + f)$ 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 $L$.

This paper is an extended version of work to appear in the proceedings of the 9th International Conference on Algorithms and Complexity (CIAC 2015) [23].

Contents of the paper

Section 2 contains preliminaries about weak linear orders, straight enumerations and unit interval graphs. In Section 3 we characterize Robinsonian matrices in terms of straight enumerations of unit interval graphs. In Section 4 we introduce our recursive algorithm to recognize Robinsonian matrices, and then we discuss the complexity issues and explain how to return all the permutations reordering a given similarity matrix as a Robinson matrix. The final Section 5 contains some questions for possible future work. Finally, in Appendix A we illustrate the algorithm on an example.
2 Preliminaries

Throughout $S^n$ denotes the set of symmetric $n \times n$ matrices. Given a permutation $\pi$ of $[n]$ and a matrix $A \in S^n$, $A_\pi := (A_{\pi(i)\pi(j)})_{i,j=1}^n \in S^n$ is the matrix obtained by permuting both the rows and columns of $A$ simultaneously according to $\pi$. For $U \subseteq [n]$, $A[U] = (A_{ij})_{i,j \in U}$ is the principal submatrix of $A$ indexed by $U$. As we deal exclusively with Robinson(ian) similarities, when speaking of a Robinson(ian) matrix, we mean a Robinson(ian) similarity matrix.

An ordered partition $\{B_1,\ldots, B_p\}$ of a finite set $V$ corresponds to a weak linear order $\psi$ on $V$ (and vice versa), 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 $i < j$. Then we also use the notation $\psi = (B_1,\ldots, B_p)$ and $B_1 <_\psi \cdots <_\psi B_p$. When all classes $B_i$ are singletons then $\psi$ is a linear order (i.e., total order) of $V$.

The reversal of $\psi$ is the weak linear order, denoted $\overrightarrow{\psi}$, of the reversed ordered partition $(B_p,\ldots, B_1)$. For $U \subseteq V$, $\psi[U]$ denotes the restriction of the weak linear order $\psi$ to $U$. 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 $\psi_1$ and $\psi_2$ 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 $V_1 \cup V_2$.

The following notions of compatibility and refinement will play an important role in our treatment. Two weak linear orders $\psi_1$ and $\psi_2$ on the same set $V$ are said to be compatible if 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 $\psi_1 \wedge \psi_2$ on $V$ defined by $x =_{\psi_1} y$ if $x =_{\psi_2} y$ for all $\ell \in \{1, 2\}$, and $x <_{\psi_\ell} y$ if $x \leq_{\psi_\ell} y$ for all $\ell \in \{1, 2\}$ with at least one strict inequality.

We will use the following fact, whose easy proof is omitted.

**Lemma 1.** Let $\psi_1,\ldots, \psi_L$ be weak linear orders on $V$. Then $\psi_1,\ldots, \psi_L$ are pairwise compatible if and only if there exists a linear order $\pi$ on $V$ which is compatible with each of $\psi_1,\ldots, \psi_L$ or, equivalently, with their common refinement $\psi_1 \wedge \cdots \wedge \psi_L$.

In what follows $V = [n] = \{1,\ldots, n\}$ is the vertex set of a graph $G = (V,E)$, whose edges are pairs $\{x, y\}$ of distinct vertices $x, y \in V$. For $x \in V$, its closed neighborhood is the set $N[x] = \{x\} \cup \{y \in V : \{x, y\} \in E\}$. Two vertices $x, y \in V$ are undistinguishable if $N[x] = N[y]$. This 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 $G$. A straight enumeration of $G$ is then a linear order $\phi = (B_1,\ldots, B_p)$ 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 (see [21]). The blocks $B_1$ and $B_p$ are called the end blocks of $\phi$ and $B_i$ (with $1 < i < p$) are its inner blocks. Having a straight enumeration is a strong property, and not all graphs have one. In fact, this notion arises naturally in the context of unit interval graphs as recalled below.

A graph $G = (V = [n], E)$ is called an interval graph if its vertices can be mapped to intervals $I_1,\ldots, I_n$ of the real line such that, for distinct vertices $x, y \in V$, $\{x, y\} \in E$ if and only if $I_x \cap I_y \neq \emptyset$. Such a set of intervals is called a realization of $G$, and it is not unique. If the graph $G$ admits a realization by unit intervals, then $G$ is said to be a unit interval graph.

Interval graphs and unit interval graphs play an important role in many applications in different fields. Many NP-complete graph problems can be solved in polynomial time on interval graphs (this holds e.g. for the bandwidth problem [27]). However, there are still problems which remains NP-hard also for interval graphs (this holds e.g. for the minimal linear arrangement problem [7]). It is well known that interval graphs and unit interval graphs can be recognized in $O(m + n)$ time [3,8,10,12,13,19,22,26,37]. For a more complete overview on linear recognition
algorithms for unit interval graphs, see [8] and reference therein. Most of the above mentioned
algorithms are based on the equivalence between unit interval graphs and proper interval graphs
(i.e., graphs admitting a realization by pairwise incomparable intervals) or indifference graphs
[31]. Furthermore, there exist several equivalent characterizations for unit interval graphs. The
following one in terms of straight enumerations will play a central role in our paper.

Theorem 2 (Unit interval graphs and straight enumerations). 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 linear ordering of the connected
components combined with any possible orientation of the straight enumeration of each con-
nected component induces a straight enumeration of \( G \). The next theorem summarizes several
known characterizations for unit interval graphs, combining results from [9, 17, 26, 29, 31, 32].
Recall that \( K_{1,3} \) is the graph with one degree-3 vertex connected to three degree-1 vertices (also
known as claw).

Theorem 3. 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 ordering \( \pi \) of \( V \) such that, for all \( x,y,z \in V \),

\[
x <_\pi y <_\pi z, \quad \{x,z\} \in E \implies \{x,y\}, \{y,z\} \in E.
\]

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

(v) (Clique condition) There is a linear ordering \( \pi \) of \( V \) such that the vertices contained in
any maximal clique of \( G \) are consecutive with respect to \( \pi \).

3 Robinsonian matrices and unit interval graphs

In this section we characterize Robinsonian matrices in terms of straight enumerations of unit
interval graphs. We focus first on binary Robinsonian matrices. We may view any symmetric
binary matrix with all diagonal entries equal to 1 as the extended adjacency matrix of a graph.
The equivalence between binary Robinsonian matrices and indifference graphs (and thus with
unit interval graphs) was first shown by Roberts [31]. Furthermore, as observed, e.g., by Corneil
et al. [9], the “neighborhood condition” for a graph is equivalent to its extended adjacency matrix
having C1P. Hence we have the following equivalence between Robinsonian binary matrices and
unit interval graphs, which also follows as a direct application of Theorem 3(ii).

Lemma 4. Let \( G = (V,E) \) be a graph and \( A_G \) be its extended adjacency matrix. Then, \( A_G \) is
a Robinsonian similarity if and only if \( G \) is a unit interval graph.

The next result characterizes the linear orders that reorder the extended adjacency matrix
\( A_G \) as a Robinson matrix in terms of the straight enumerations of \( G \). It is simple but will play
a central role in our algorithm for recognizing Robinsonian similarities.
Theorem 5. Let $G = (V, E)$ be a graph. A linear order $\pi$ of $V$ reorders $A_G$ as a Robinson matrix if and only if there exists a straight enumeration of $G$ whose corresponding weak linear order $\psi$ is compatible with $\pi$, i.e., satisfies:

$$\forall x, y \in V \text{ with } x \neq \psi y \quad x \prec_\pi y \iff x < \psi y.$$  \hfill (2)

Proof. Assume that $\pi$ 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 holds for $\pi$ and that each block of $G$ is an interval w.r.t. $\pi$. Therefore the order $\pi$ induces an order $\psi$ of the blocks: $B_1 < \psi \ldots < \psi B_p$, 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, $\psi$ is compatible with $\pi$ by construction. Moreover, $\psi$ 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 $\pi$.

Conversely, assume that $B_1 < \psi \ldots < \psi B_p$ is a straight enumeration of $G$ and let $\pi$ be a linear order of $V$ which is compatible with $\psi$, i.e., satisfies (2). We show that $\pi$ reorders $A_G$ as a Robinson matrix. That is, we show that if $x < \pi y$, $A_G(x, y) = 1$, then (2) holds. Indeed, if $x < \pi y$, then $x \prec_\pi y$ and thus $x \prec_\psi y$. Assume now that $x \prec_\pi y$ and $y \prec_\pi z$. In other words, $x < \pi y < \pi z$. We show that $x < \psi y < \psi z$. As $\psi$ is a straight enumeration, $x \prec_\pi y$ and thus $y \prec_\pi x$. However, the converse is not true. It is easy to build an example where $G$ is not Robinsonian although the extended adjacency matrix of each level graph is 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 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 [33, Theorem 4.4]). However, he does not give any algorithmic insight on how to find such orders.

Combining the links between binary Robinsonian matrices and unit interval graphs (Lemma 4) and between reorderings of binary Robinsonian matrices and straight enumerations of unit interval graphs (Theorem 5) together with the decomposition result of Lemma 6, we obtain the following characterization of Robinsonian matrices.

**Theorem 7.** 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 $G(1), \ldots, G(L)$ whose corresponding weak linear orders $\psi_1, \ldots, \psi_L$ are pairwise compatible.

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

**Proof.** Observe first that if assertion (ii) holds then (i) follows directly using the result of Lemma 1. We now prove (ii). Assume that $A$ is Robinsonian and let $\pi$ a linear order of $V$ that reorders $A$ as a Robinson matrix. Then $A_\pi$ is Robinson and thus, by Lemma 6, each permuted matrix $(A_{G(\ell)})_\pi$ is a Robinson matrix. Then, applying Theorem 5 for each $\ell \in [L]$, there exists a straight enumeration of $G(\ell)$ whose corresponding weak linear ordering $\psi_\ell$ is compatible with $\pi$. We can thus conclude that the common refinement of $\psi_1, \ldots, \psi_L$ is compatible in view of Lemma 6. 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 with $\pi$ and their common refinement is compatible with $\pi$. Then, by Theorem 5, $\pi$ reorders simultaneously each $A_{G(\ell)}$ as a Robinson matrix and thus $A_\pi$ is Robinson, which shows that $A$ is Robinsonian. \hfill \Box

4 The algorithm

We describe here our algorithm for recognizing whether a given symmetric nonnegative matrix $A$ is Robinsonian. First, we introduce an algorithm which either returns a permutation reordering $A$ as a Robinson matrix or states that $A$ is not a Robinsonian matrix. Then, we show how to modify it in order to return all the permutations reordering $A$ as a Robinson matrix.

4.1 Overview of the algorithm

The algorithm is based on Theorem 7. The main idea 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. As above, $L$ denotes the number of distinct nonzero entries of $A$ and throughout $G(\ell) = (V = [n], E_\ell)$ is the $\ell$-th level graph, whose edges are the pairs $\{x, y\}$ with $A_{xy} \geq \alpha_\ell$, for $\ell \in [L]$.

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 $\psi$ of $V$. Roughly speaking, $G$ will correspond to a level graph $G(\ell)$ of $A$ (in fact, to a connected component of it), while $\psi$ 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 $\psi$ 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: CO-Lex-BFS (see Algorithm 1), a variation of Lex-BFS, which finds and orders the connected components of the level graphs;
Then there exist
\[ x \in V \]
(i) Assume first that Algorithm 1 successfully terminates and returns the linear ordering
\[ \phi \] 
Proof. Let \( \psi \) be ordered along the already detected components in a compatible way with respect to \( \psi \).
\[ \text{Lemma 9.} \]
\[ \text{Consider a graph} \ G = (V,E) \text{and a weak linear order} \ \psi \text{ of} \ V. \text{If} \ G \text{has a straight enumeration} \ \phi \text{ compatible with} \ \psi \text{ then there exists an ordering} \ V_1,\ldots,V_c \text{ of the connected components of} \ G \text{ which is compatible with} \ \psi, \text{ i.e., such that} \ V_1 \leq_\psi \ldots \leq_\psi V_c. \]
\[ \text{Algorithm} \ 1 \text{ is based on the following observations. When the vertex} \ v \text{ in the set} \ S \text{ at line} 10 \text{ (which represents the current set of unvisited vertices with a tie, known as slice in the Lex-BFS) has label} \ \emptyset, \text{ it means that} \ v \text{ is not contained in the current component} \ V_\omega, \text{ so a new component containing} \ v \text{ is opened. Every time a connected component} \ V_\omega \text{ is detected we check if it can be ordered along the already detected components in a compatible way with} \ \psi. \text{ For this we distinguish two cases:} \]
1. if \( V_\omega \) meets more than one block of \( \psi \) (i.e., if \( B_\omega^\text{min} \prec_\psi B_\omega^\text{max} \)), we check if all the inner blocks between \( B_\omega^\text{min} \) and \( B_\omega^\text{max} \) are contained in \( V_\omega \). If this is not the case, then the algorithm stops. Moreover the algorithm also stops if both \( V_\omega \) and \( V_{\omega-1} \) meet exactly the same two blocks, i.e., \( B_\omega^\text{min} = B_{\omega-1}^\text{min} \) and \( B_\omega^\text{max} = B_{\omega-1}^\text{max} \). In both cases it is indeed not possible to order the components in a compatible way with \( \psi \).
2. if \( V_\omega \) meets only one block \( B_k \) of \( \psi \) (i.e., if \( V_\omega \subseteq B_k \)) and if moreover this block \( B_k \) is the first block of the previous connected component \( V_{\omega-1} \) (i.e., \( B_k = B_{\omega-1}^\text{min} \)), then we swap \( V_{\omega-1} \) and \( V_\omega \) in order to make the ordering of the components compatible with \( \psi \). The ordering \( \sigma \) 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 \( V_\omega \) and \( V_{\omega-1} \) are contained in \( B_k \), then we do not need to do this swap, i.e., the two components \( V_\omega \) and \( V_{\omega-1} \) can be ordered arbitrarily.

The next lemma shows the correctness of Algorithm 1.
\[ \text{Lemma 9. Let} \ G = (V,E) \text{ be a graph and let} \ \psi \text{ be a weak linear order of} \ V. \]
(i) If Algorithm 1 successfully terminates then the returned order \( V_1,\ldots,V_c \) of the components satisfies \( V_1 \leq_\psi \ldots \leq_\psi V_c \).
(ii) If Algorithm 1 stops then no ordering of the components exists that is compatible with \( \psi \).
\[ \text{Proof.} \]
(i) Assume first that Algorithm 1 successfully terminates and returns the linear ordering \( V_1,\ldots,V_c \) of the components. Suppose for contradiction that \( V_{\omega-1} \nleq_\psi V_\omega \) 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
Algorithm 1: CO-Lex-BFS($G, \psi, \tau$)

**Input:** a graph $G = (V, E)$, a weak linear order $\psi = (B_1, \ldots, B_p)$ of $V$, a linear order $\tau$ of $V$ compatible with $\psi$

**Output:** a linear order $\sigma$ of $V$ and a linear order $(V_1, \ldots, V_c)$ of the connected components of $G$ compatible with $\psi$ and $\sigma$, or STOP (no such linear order of the components exists)

1. mark all the vertices as unvisited
2. $u$ is the first vertex appearing in $\tau$
3. $\text{label}(u) = |V|$
4. $\omega = 1$
5. $V_\omega, B^\min_\omega, B^\max_\omega = \emptyset$
6. foreach $v \in V \setminus u$ do
   7. $\text{label}(v) = \emptyset$
7. end
8. for $i = |V|, \ldots, 1$ do
   9. let $S$ be the set of unvisited vertices with the lexicographically largest label
   10. pick the vertex $v$ in $S$ appearing first in $\tau$ and mark it as visited
11. $\sigma(v) = |V| + 1 - i$
12. if $\text{label}(v) = \emptyset$ then
   13. if $V_\omega \subseteq B^\min_\omega$ then
   14. swap $V_\omega$ and $V_{\omega - 1}$ and modify $\sigma$ accordingly
   15. else
   16. if $B^\min_\omega < \psi B^\max_\omega$ or if there exists a block $B$ of $\psi$ such that $B \not\subseteq V_\omega$ and $B^\min_\omega < \psi B < \psi B^\max_\omega$ then
   17. stop (no ordering of the components compatible with $\psi$ exists)
   18. end
19. end
20. $\omega = \omega + 1$
21. $V_\omega = \emptyset$
22. end
23. $V_\omega = V_\omega \cup \{v\}$
24. $B^\min_\omega$ is the first block in $\psi$ which meets $V_\omega$
25. $B^\max_\omega$ is the last block in $\psi$ which meets $V_\omega$
26. foreach unvisited vertex $w$ in $N(v)$ do
   27. append $i$ to $\text{label}(w)$
28. end
29. end
30. return $(V_1, \ldots, V_c)$ and $\sigma$, or STOP

component $V_{\omega - 1}$. Then, $z \leq \psi y$ (for if not the algorithm would have selected $y$ before $z$ when opening the component $V_{\omega - 1}$). 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 $B^\min_{\omega - 1}$ and $B^\max_{\omega - 1}$ 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 $B \subseteq B_i = B^\min_{\omega - 1}$ and then the algorithm would have swapped $V_\omega$ and $V_{\omega - 1}$. Hence $\psi[V_\omega]$ has at least two blocks and $B^\min_\omega \leq \psi B_i < \psi B_j \leq \psi B^\max_{\omega - 1}$, which is again a contradiction since the algorithm would have stopped.
(ii) Assume now that the algorithm stops when dealing with the component \(V_\omega\). Then \(\psi[V_\omega]\) has at least two blocks. Suppose first that the algorithm stops because \(B_{\omega_1}^{\min} \prec_\psi B_{\omega_1}^{\max}\). Then clearly one cannot have \(V_{\omega-1} \prec_\psi V_\omega\). We show that we also cannot have \(V_\omega \prec_\psi V_{\omega-1}\). For this assumption for contradiction that \(V_\omega \prec_\psi V_{\omega-1}\). Let \(y\) be the first selected vertex in \(V_\omega\) and let \(x\) be the first vertex selected in \(V_{\omega-1}\). Then, \(y \in B_{\omega-1}^{\min}\), \(x \preceq_\psi y\) (for if not the algorithm would have considered the component \(V_\omega\) before \(V_{\omega-1}\)), and thus \(B_{\omega-1}^{\min} \preceq_\psi B_{\omega}^{\min}\). If \(B_{\omega-1}^{\min} \prec_\psi B_{\omega}^{\min}\) then the algorithm would have stopped earlier when examining \(V_{\omega-1}\), since \(B_{\omega-1}^{\min} \prec_\psi B_{\omega}^{\min}\) 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 \(x \prec_\psi z\), which contradicts \(V_\omega \prec_\psi V_{\omega-1}\). Suppose now that the algorithm stops because \(B_{\omega}^{\min} \prec_\psi B_{\omega}^{\max}\) 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 \(x \prec_\psi z\), which contradicts \(V_\omega \prec_\psi V_{\omega-1}\). Suppose now that the algorithm stops because \(B_{\omega}^{\min} \prec_\psi B_{\omega}^{\max}\) 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 \(x \prec_\psi z\), which contradicts \(V_\omega \prec_\psi V_{\omega-1}\).

\[\square\]

4.1.2 Straight enumerations

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 \(\sigma_\omega\) is a suitable given order of \(V_\omega\) (namely, \(\sigma_\omega = \sigma[V_\omega]\), where \(\sigma\) is the vertex order returned by CO-Lex-BFS(G, \(\psi, \tau\)). This routine is essentially the 3-sweep unit interval graph recognition algorithm of Corniel [8] which, briefly, computes three times a Lex-BFS (each is named a sweep) and uses the vertex ordering coming from the previous sweep to break ties in the search for the next sweep. The only difference of Straight enumeration\((G[V_\omega], \sigma_\omega)\) with respect to Corniel’s algorithm is that we save the first sweep, because we use the order \(\sigma_\omega\) returned by CO-Lex-BFS. We now describe the routine Straight enumeration which is based on the algorithms of [8], [3] and [6], §2. Below, \(\deg_G(v)\) denotes the degree of the vertex \(v\) in \(G\).

Basically, after the last sweep of Lex-BFS, for each vertex \(v\) we define the leftmost vertex \(l_{mn}(v)\) and the rightmost vertex \(r_{mn}(v)\), according to \(\sigma^{++}\), that are adjacent to \(v\). Checking whether \(r_{mn}(v) - l_{mn}(v) = \deg_G(v)\) corresponds exactly to checking whether the neighborhood condition holds for node \(v\). The vertices with the same leftmost and rightmost vertex are then indistinguishable vertices, and they form a block of \(G\). The order of the blocks follows the vertex order \(\sigma^{++}\).

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). The next lemma shows that the subroutine Straight enumeration correctly chooses the orientation of the straight enumeration of a connected graph with respect to a given weak linear order \(\psi\).

**Lemma 10.** Let \(\psi = (B_1, \ldots, B_p)\) be a weak linear order and \(\tau\) be a linear order of \(V\) compatible with \(\psi\). Let \(G\) be a connected graph and let \(\sigma\) be the vertex ordering returned by CO-Lex-BFS(G, \(\psi, \tau\)). Finally, let \(\phi = (C_1, \ldots, C_q)\) be the straight enumeration returned by Straight enumeration\((G, \sigma)\). Assume that there is no edge between \(B_1\) and \(B_p\) and that Algorithm 3 successfully terminates. Then \(\psi\) and \(\phi\) are not compatible.

**Proof.** Let \(a\) be the first vertex of \(\sigma\). Then \(a \in B_1\), because in Algorithm 1 the first vertex of \(\sigma\) is the first vertex of \(\tau\), which is compatible with \(\psi\). We first claim that \(\{a, y\} \notin E\). For this we will use the following well known property of the Lex-BFS order \(\sigma\): if \(\sigma\) is a Lex-BFS ordering of an arbitrary graph \(G = (V, E)\) then, for all vertices \(u \prec_\sigma v \prec_\sigma w\) such that \(\{u, w\} \in E\) and \(\{u, v\} \notin E\), there exists a node \(x \prec_\sigma u\) with \(\{x, v\} \in E\) ( [4], [18]). Applying this property
Algorithm 2: \textit{Straight enumeration}(G, \sigma)

\begin{verbatim}
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 $i = 0$ \hspace{1cm} (index of the blocks of $\psi$)
4 $L = R = 0$ \hspace{1cm} (dummy variables to record the current block $B_i$)
5 for $v = 1, \ldots, |V|$ do
6 \hspace{1cm} $l_{mn}(v) = \min\{u : u \in N[v]\}$ \hspace{1cm} (leftmost vertex adjacent to $v$)
7 \hspace{1cm} $r_{mn}(v) = \max\{u : u \in N[v]\}$ \hspace{1cm} (rightmost vertex adjacent to $v$)
8 \hspace{1cm} if $r_{mn}(v) - l_{mn}(v) \neq \deg_G(v)$ then
9 \hspace{1.5cm} \textbf{stop} \hspace{1cm} ($G$ is not a unit interval graph)
10 \hspace{1cm} else
11 \hspace{1.5cm} if $l_{mn}(v) = L$ and $r_{mn}(v) = R$ then
12 \hspace{2cm} $C_i = C_i \cup \{v\}$.
13 \hspace{1.5cm} else
14 \hspace{2cm} $L = l_{mn}(v)$
15 \hspace{2cm} $R = r_{mn}(v)$
16 \hspace{2cm} $i = i + 1$
17 \hspace{2cm} $C_i = \{v\}$
18 \hspace{1.5cm} end
19 \hspace{1cm} end
20 end
21 return $\phi = (C_1, \ldots, C_q)$ or STOP
\end{verbatim}


to $u = a$ and $w = y$ that are respectively the first and the last nodes of $\sigma$ we deduce that if $\{a, y\} \in E$ then $a$ is adjacent to all nodes, which contradicts the assumption that there is no edge between $B_1$ and $B_p$. Let $y$ be the last vertex of $\sigma^{++}$. Then $y \in C_q$ and, in view of [8, Fact 4], $y$ is also the last vertex of $\sigma$. Indeed, if $\{a, y\} \in E$ then, using the above property of $\sigma$, we can conclude that $a$ is adjacent to all nodes of $G$, which would contradict the assumption that there is no edge between $B_1$ and $B_p$.

Suppose now, for contradiction, that $\psi = (B_1, \ldots, B_p)$ and $\overline{\phi} = (C_1, \ldots, C_1)$ are compatible. We distinguish two cases, depending whether $y$ belongs to $B_1$. Consider first the case when $y \notin B_1$. Then $a <_\psi y$. As $\overline{\phi}$ is compatible with $\psi$, we have $a \leq_\psi y$ and thus $a \in C_q$, implying that $\{a, y\} \in E$, contradicting our previous claim.

We now assume that $y \in B_1$. Let $C_k$ the block containing $a$. So $k \leq q - 1$ as $\{a, y\} \notin E$. Moreover, $C_q \cup C_{q-1} \cup \ldots \cup C_{k+1} \subseteq B_1$. Indeed, for any vertex $x \in C_q \cup C_{q-1} \cup \ldots \cup C_{k+1}$, $x <_\psi a$, which implies $x \leq_\psi a$ and thus $x \in B_1$. We now use the fact that, since $G$ is connected, each block $C_i$ is adjacent to its neighboring blocks $C_{i+1}$ and $C_{i-1}$. In particular, there is a path from $a$ to $y$ which is contained in $B_1$. Consider such a shortest path $(a = a_0, a_1, \ldots, a_r = y)$ where all nodes belong to $B_1$. As the path is shortest, the nodes come in that order in $\sigma$, i.e., $a = a_0 <_\sigma a_1 <_\sigma \ldots <_\sigma a_r = \sigma y$. Consider now a vertex $x \in B_p$. Say, $a_i <_\sigma x <_\sigma a_{i+1}$ for some $0 \leq i \leq r - 1$. As $a_i$ is adjacent to $a_{i+1}$, it follows that $a_i$ is adjacent to $x$, which contradicts again the assumption that no node of $B_1$ is adjacent to a node of $B_p$. \qed

If $G$ is connected, its straight enumeration $\phi$ is unique up to reversal and the 3-sweep Lex-BFS algorithm implicitly returns it correctly orientated with respect to $\psi$. On the other hand,
the following holds for all $x, y$

Assume first that the algorithm terminates and returns $\Phi$. We show that $\Phi = W \psi$ is clearly compatible with $\phi$.

**Proof.** The proof is by induction on the number of blocks of $\psi$.

1. $B^\text{max}$ is the last block of $\psi$ meeting $C_1$
2. if there exists a block $B$ of $\psi$ such that $B <_\psi B^\text{max}$ and $B \nsubseteq C_1$ then
   3. stop
   (\(\psi\) and $\phi$ are not compatible)
3. else
   4. $W = V \setminus C_1$
   5. $\Phi = (\psi[C_1], \text{Refine}(\psi[W], \phi[W], \tau[W]))$
6. end
7. return $\Phi$ or STOP

**Lemma 11.** If Algorithm 3 successfully terminates then the returned $\Phi$ is the common refinement of $\psi$ and $\phi$. If Algorithm 3 stops then $\psi$ and $\phi$ are not compatible.

**Proof.** The proof is by induction on the number $q$ of blocks of $\phi$. If $q = 1$ then $\phi = (V)$ is clearly compatible with $\psi$ 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 terminates and returns $\Phi$. We show that $\Phi = \psi \land \phi$, i.e., that the following holds for all $x, y \in V$:

$$
\begin{align*}
x = \phi y & \iff x = \psi y \text{ and } x = \phi y, \\
x <_\phi y & \iff x \leq \psi y \text{ and } x \leq \phi y \text{ with at least one strict inequality.}
\end{align*}
$$

(3)

If $x, y \in C_1$ then $x = \phi y$ and (3) holds since $\Phi[C_1] = \psi[C_1]$. If $x, y \in V \setminus C_1$, then (3) holds by the induction assumption. Suppose now $x \in C_1$ and $y \in V \setminus C_1$. Then $x <_\phi y$ and $x <_\psi y$. We show that $x \leq \psi y$ holds. For this let $B_i$ (resp., $B_j$) be the block of $\psi$ containing $x$ (resp., $y$). Then $B_i \leq_\psi B^\text{max}$ since $B_i$ meets $C_1$ as $x \in C_1$. Moreover, $B^\text{max} \leq_\psi B_j$, which implies $x \leq_\psi y$. Indeed, if one would have $B_j <_\psi B^\text{max}$, then the algorithm would have stopped since $B_j \nsubseteq C_1$ as $y \in B_j \setminus C_1$. Assume now that the algorithm stops. If it stops because there is a block $B <_\psi B^\text{max}$ such that $B \nsubseteq C_1$, then pick elements $x \in B \setminus C_1$ and $y \in C_1 \cap B^\text{max}$ so that $y <_\phi x$ and $x <_\psi y$, which shows that $\psi$ and $\phi$ are not compatible. If the algorithm stops because the recursive routine $\text{Refine}(\psi[W], \phi[W], \tau[W])$ stops then, by the induction assumption, $\psi[W]$ and $\phi[W]$ are not compatible and thus $\psi$ and $\phi$ neither. This concludes the proof.

\[\square\]
4.1.4 Main algorithm

We can now describe our main algorithm $\text{Robinson}(A, \psi, \tau)$. Given a nonnegative matrix $A \in S^n$, a weak linear order $\psi$ and an order $\tau$ of $V = [n]$ that are compatible, it either returns a weak linear order $\Phi$ of $V$ compatible with $\psi$ and with straight enumerations of the level graphs of $A$, or it indicates that such $\Phi$ does not exist. The idea behind the algorithm is the following. We use the subroutines $\text{CO-Lex-BFS}$ and $\text{Straight enumeration}$ to order the components and compute the straight enumerations of the level graphs of $A$, and we refine them using the subroutine $\text{Refine}$. 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 4 below.

**Algorithm 4: $\text{Robinson}(A, \psi, \tau)$**

- **input:** a weak linear order $\psi$ of $V = [n]$, a linear order $\tau$ of $V$ compatible with $\psi$, a nonnegative matrix $A \in S^n$
- **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. $\text{CO-Lex-BFS}(G, \psi, \tau)$ 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. $a_{\min}$ is the smallest entry of $A[V_\omega]$
6. **if** $a_{\min} > 0$ **then**
7. $A[V_\omega] := A[V_\omega] - a_{\min}J$ and $G[V_\omega]$ is its updated support
8. **end**
9. $\phi_\omega = \text{Straight enumeration}(G[V_\omega], \sigma[V_\omega])$ (if $G[V_\omega]$ is a unit interval graph)
10. $\Phi_\omega = \text{Refine}(\psi[V_\omega], \phi_\omega)$ (if $\psi[V_\omega]$ and $\phi_\omega$ are compatible)
11. $a'_{\min}$ is the smallest nonzero entry of $A[V_\omega]$
12. $A'[V_\omega]$ is obtained from $A[V_\omega]$ by setting its entries with value $a'_{\min}$ to zero
13. **if** $A'[V_\omega]$ is diagonal **then**
14. $\Phi = (\Phi, \Phi_\omega)$
15. **else**
16. $\tau_\omega$ is a linear order of $V_\omega$ compatible with $\Phi_\omega$
17. $\Phi = (\Phi, \text{Robinson}(A'[V_\omega], \Phi_\omega, \tau_\omega))$
18. **end**
19. **end**
20. **return:** $\Phi$ or STOP

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

For each component $V_\omega$, we compute the straight enumeration $\phi_\omega$ of $G[V_\omega]$ (if it exists, else we stop), and we find the common refinement $\Phi_\omega$ of $\psi[V_\omega]$ and $\phi_\omega$ (if it exists, else we stop). 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 $\Phi_\omega$ after $\Phi_{\omega-1}$ in $\Phi$. Otherwise, we make a recursive call, where the the input of the recursive routine is the matrix $A'[V_\omega]$, $\Phi_\omega$, and $\tau_\omega$. If the algorithm successfully terminates, then the concatenation $(\phi_1, \ldots, \phi_c)$ will represent a straight enumeration of $G$, and $\Phi = (\Phi_1, \ldots, \Phi_c)$ will represent the common refinement of this straight enumeration with the given weak linear order $\psi$ and with the level graphs of $A$.

The final algorithm is Algorithm 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, and whose leaves (i.e. the pruned nodes) are the subproblems whose corresponding submatrices are diagonal.

**Algorithm 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. $\tau = (1, 2, \ldots, n)$
3. $\Phi = \text{Robinson}(A, \psi, \tau)$
4. $\pi$ is a linear order of $V$ compatible with $\Phi$
5. return: $\pi$ or “$A$ not Robinsonian”

The correctness of Algorithm 5 follows directly from the correctness of Algorithm 4, which is shown by the next theorem. Indeed, assume that Algorithm 4 is correct. Then, if Algorithm 5 terminates then it computes a weak linear order $\Phi$ compatible with straight enumerations of the level graphs of $A$ and thus the returned order $\pi$ orders $A$ as a Robinson matrix in view of Theorem 7 (ii). On the other hand, if Algorithm 5 stops then Algorithm 4 stops with the input $(A, \psi = (V))$. Then no weak linear order $\Phi$ exists which is compatible with straight enumerations of the level graphs of $A$ and thus, in view of Theorem 7 (i), $A$ is not Robinsonian.

**Theorem 12.** Consider a weak linear order $\psi$ of $V = [n]$ and a nonnegative matrix $A \in \mathcal{S}^n$ ordered compatibly with $\psi$.

(i) If Algorithm 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 $\Phi$ is compatible with each of them and with $\psi$.

(ii) If Algorithm 4 stops then there do not exist straight enumerations of the level graphs of $A$ that are pairwise compatible and compatible with $\psi$.

**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 successfully and returns Φ. Then G is the support of A, CO-Lex-BFS(G, ψ, τ) orders the components of G as V₁ ≤ψ ... ≤ψ Vc, and Φ = (Φ₁, ..., Φc) where each Φᵦ = Φ[Vᵦ] is built as the common refinement of ψ[Vᵦ] and a straight enumeration of G[Vᵦ]. 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 3 stops. If it stops when applying CO-Lex-BFS(G, ψ, τ), 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. If the algorithm stops when applying Straight Enumeration to G[Vᵦ] then no straight enumeration of Gᵦ exists. Else, if the algorithm stops when applying Refine to (ψ[Vᵦ], φω) then ψ[Vᵦ] and the straight enumeration φω of G[Vᵦ] are not compatible. In both cases, no straight enumeration of G exists that is compatible with ψ.

We now assume that Theorem 12 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, Φ = (Φ₁, ..., Φc) after ordering the components compatibly with ψ as V₁ ≤ψ ... ≤ψ Vc and where each Φᵦ is constructed by computing a straight enumeration φᵦ of G[Vᵦ] compatible with ψ, Φᵦ = φᵦ ∧ ψ[Vᵦ], and thus Φᵦ = Robinson(A'[Vᵦ]τᵦ, Φᵦ, τᵦ), where A'[Vᵦ] is obtained from A[Vᵦ] by setting to 0 its entries with smallest nonzero value. By the induction assumption, Φᵦ is compatible with straight enumerations of the level graphs of the matrix A'[Vᵦ] and with Φᵦ. This implies that Φᵦ is compatible with ψ[Vᵦ] and with the straight enumeration φᵦ of G[Vᵦ], and thus with straight enumerations of all the level graphs of A[Vᵦ], as desired. Finally assume that the algorithm stops. If it stops when applying Robinson(A'[Vᵦ]τᵦ, Φᵦ, τᵦ) then, by the induction assumption, no straight enumerations of the level graphs of A'[Vᵦ] exist that are compatible with Φᵦ = ψ[Vᵦ] ∧ φᵦ, and thus no straight enumerations of the level graphs of A[Vᵦ] exist that are compatible with ψ[Vᵦ] and the same holds in turn for the full matrix A. The same conclusion holds if the algorithm stops at another step, which concludes the proof.

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 1 and 2 and then we derive the complexity of the final Algorithm 4. 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 = |E₁| ≥ |E₂| ≥ ... ≥ |EL| for the level graphs of A. We assume that A is a nonnegative symmetric matrix, which is given as an adjacency list of an undirected weighted graph, where each vertex x ∈ 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 since we are given a weak linear order ψ and a linear order τ of V that are compatible, the blocks of ψ are intervals of the order τ and thus one can check whether a given set C ⊆ V is contained in a block B of ψ in O(|C|) operations (simply by comparing each element of C to the endpoints of the interval B). Furthermore, the size of any block of ψ is simply given by the difference between its extremities (plus one).
Lemma 13. Algorithm \[7\] runs in \(O(|V| + |E|)\) time.

Proof. It is well known that Lex-BFS can be implemented in linear time \(O(|V| + |E|)\). In our implementation of Algorithm \[7\] we will follow the linear time implementation of Corneil \[8\], which uses the data structure based on the paradigm of “partitioning” presented in \[19\]. In order to carry out the other operations about the components of \(G\) we maintain a doubly linked list, where each node of the list represents a connected component \(V_\omega\) of \(G\) and it has a pointer to the connected component \(V_{\omega-1}\) ordered immediately before \(V_\omega\) and to the connected component \(V_{\omega+1}\) ordered immediately after \(V_\omega\). 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 \(V_\omega\), the first block \(B_\omega^{\min}\) and the last block \(B_\omega^{\max}\) in \(\psi\) meeting \(V_\omega\). These two blocks \(B_\omega^{\min}\) and \(B_\omega^{\max}\) can be found in time \(O(|V|)\) as follows. First one finds the smallest element \(v_{\min}\) (resp. the largest element \(v_{\max}\)) of \(V_\omega\) in the order \(\tau\), which can be done in \(O(|V_\omega|)\). Then, \(B_\omega^{\min}\) is the block of \(\psi\) containing \(v_{\min}\), which can be found in \(O(|V_\omega|)\). Analogously for \(B_\omega^{\max}\), which is the block of \(\psi\) containing \(v_{\max}\). Checking whether \(V_\omega\) is contained in the block \(B_\omega^{\min}\) can be done in \(O(|V_\omega|)\) (since \(B_\omega^{\min}\) is an interval). In order to check whether all the inner blocks between \(B_\omega^{\min}\) and \(B_\omega^{\max}\) are contained in \(V_\omega\) we proceed as follows. Let \(B_\omega\) be the union of these inner blocks, which is an interval of \(\tau\). 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 \(B_\omega\) are equal. For this we check first whether \(V_\omega \setminus (B_\omega^{\min} \cup B_\omega^{\max})\) is contained in \(B_\omega\) (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 \[7\] is \(O(|V| + |E|)\).

Lemma 14. Algorithm \[3\] runs in \(O(|V|)\) time.

Proof. We show the lemma using induction on the number \(q\) of blocks of \(\phi\). 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 \(B_\omega^{\max}\) of \(\psi\) 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 \(\tau\) and one returns the block of \(\psi\) containing \(v_{\max}\), which can be done in \(O(|C_1|)\). Then let \(B\) be the union of the blocks preceding \(B_\omega^{\max}\). In order to check whether \(B \subseteq C_1\) or, equivalently, whether \(C_1 \setminus B_\omega^{\max} = B\), we proceed as in the previous lemma: we first check whether \(C_1 \setminus B_\omega^{\max} \subseteq B\) and then whether \(|C_1 \setminus B_\omega^{\max}| = |B|\), 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 \textit{Refine} to the restrictions of \(\psi, \phi, \tau\) to the set \(V \setminus C_1\), gives an overall running time \(O(|V|)\).

We can now complete the complexity analysis of our algorithm.

Theorem 15. Let \(A\) be a nonnegative \(n \times n\) symmetric matrix given as a weighted adjacency list and let \(m\) be the number of (upper diagonal) nonzero entries of \(A\). Algorithm \[3\] recognizes whether \(A\) is a Robinsonian matrix in time \(O(d(|V| + |E|))\), where \(d\) is the depth of the recursion tree created by Algorithm \[3\]. 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 \[3\] we are given a matrix \(A\) and its support graph \(G\), and we set \(\psi = (V)\) and \(\tau = (1, 2, \ldots, n)\). First we run the routine \textit{CO-Lex-BFS}(\(G, \psi, \tau\)) in \(O(|V| + |E|)\) time, in order to find and order the components of \(G\). For each component \(V_\omega\), the following tasks are performed. We compute
a straight enumeration $\phi_\omega$ of $G[V_\omega]$, in time $O(|V_\omega| + m_\omega)$ where $m_\omega$ is the number of edges of $G[V_\omega]$ and we apply the routine $\text{Refine}$ to $\psi[V_\omega]$ and $\phi_\omega$, in time $O(|V_\omega|)$. 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 $\text{Robinson}(A'[V_\omega], \Phi_\omega, \tau_\omega)$ is carried out in time $O(d_\omega(|V_\omega| + m_\omega))$, where $d_\omega$ denotes the depth of the corresponding recursion tree. As $d_\omega \leq d - 1$ for each $\omega$, 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.

Note that, if the smallest entry $\alpha_0$ of some intermediary submatrix $A'[V_\omega]$ is nonzero, then before computing straight enumerations we substract $\alpha_0$ from all the entries of $A'[V_\omega]$ to get a matrix with smaller support, which turns out in a speed up of the algorithm. As we will see in the example in the appendix A, the depth $d$ of the recursion tree can be smaller than the number $L$ of distinct nonzero entries. Indeed in this example we have $d = 6$ while $L = 10$.

4.3 Finding all Robinsonian 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 [3].

A PQ-tree $T$ is a special rooted ordered tree. The leaves are in one-to-one correspondence with the elements of the groundset $V$ and their order gives a linear order of $V$. The nodes of $T$ 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. Given a node $\alpha$ of $T$, $T_\alpha$ denotes the subtree of $T$ with root $\alpha$.

A straight enumeration $\psi = (B_1, \ldots, B_p)$ of a graph $G = (V, E)$ corresponds in a unique way to a PQ-tree $T$ as follows. If $G$ is connected, then the root of $T$ is a Q-node, denoted $\gamma$, and it has children $\beta_1, \ldots, \beta_p$ (in that order). For $i \in [p]$, the node $\beta_i$ 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 $T_{\beta_i}$. If a block $B_i$ is a singleton then no node $\beta_i$ appears and the element of $B_i$ is directly a child of the root $\gamma$ (see the example in Figure 1).

$$A_G = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

Figure 1: A connected graph $G$ and the PQ-tree corresponding to its straight enumeration

If $G$ is not connected, let $V_1, \ldots, V_c$ be its connected components. For each connected component $G[V_\omega]$, $T_\omega$ is its PQ-tree (with root $\gamma_\omega$) as indicated above. Then, the full PQ-tree $T$ is obtained by inserting a P-node $\alpha$ as ancestor, whose children are the subtrees $T_{1}, \ldots, T_c$ (see Figure 2).
We now indicate how to modify Algorithms 4 and 5 in order to return a PQ-tree $T$ encoding all the permutations ordering $A$ as a Robinson matrix.

We modify Algorithm 4 by taking as input, beside the matrix $A$, the weak linear order $\psi$ and the linear order $\tau$ compatible with $\psi$, also a node $\alpha$. Then, the output is a PQ-tree $T_\alpha$ rooted in $\alpha$, representing all the possible weak linear orders $\Phi$ compatible with $\psi$ 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 $T_\omega$ for each connected component $V_\omega$ of $G$ and then to merge these trees according to the order of the components found by the routine $CO$-Lex-$BFS(G, \psi, \tau)$. To carry out this merging step we classify the components into the following three groups:

1. $\Theta$, which consists of all $\omega \in [c]$ for which the connected component $V_\omega$ meets at least two blocks of $\psi$.

2. $\Lambda$, which consists of all $\omega \in [c]$ for which the component $V_\omega$ is contained in some block $B_i$, which contains no other component.

3. $\Omega = \bigcup_{i=1}^p \Omega_i$, where $\Omega_i$ consists of all $\omega \in [c]$ for which the component $V_\omega$ 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 4, we create a Q-node $\gamma_\omega$. After the common refinement $\Phi_\omega$ (of $\psi|_{V_\omega}$ and the straight enumeration $\phi_\omega$ of $G|_{V_\omega}$) has been computed, we have two possibilities. If $A'|_{V_\omega}$ is diagonal, then we build the tree $T_\omega$ rooted in $\gamma_\omega$ and whose children are P-nodes corresponding to the blocks of $\Phi_\omega$ (and prune the recursion tree at this node). Otherwise, we build the tree $T_\omega$ recursively as output of $Robinson(A'|_{V_\omega}, \Phi_\omega, \tau_\omega, \gamma_\omega)$.

After all the connected components have been analyzed, we insert the trees $T_\omega$ in the final tree $T_\alpha$ in the order they appear according to the routine $CO$-Lex-$BFS(G, \psi, \tau)$. The root node is $\alpha$ and is given as input. For each component $V_\omega$, we do the following operation to insert $T_\omega$ in $T_\alpha$, depending on the type of the component $V_\omega$:

1. If $\omega \in \Theta$, then $\phi_\omega$ is the only straight enumeration compatible with $\psi|_{V_\omega}$. Then we delete the node $\gamma_\omega$ and the children of $\gamma_\omega$ become children of $\alpha$ (in the same order).

2. If $\omega \in \Lambda$, then both $\phi_\omega$ and its reversal $\bar{\phi}_\omega$ are compatible with $\psi|_{V_\omega}$. Then $\gamma_\omega$ becomes a child of $\alpha$.

3. If $\omega \in \Omega_i$ for some $i \in [p]$, then both $\phi_\omega$ and $\bar{\phi}_\omega$ 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 $\beta_i$ which is a P-node and becomes a child of $\alpha$ and, for each $\omega' \in \Omega_i$, $\gamma_{\omega'}$ becomes a child of $\beta_i$. 

Figure 2: The PQ-tree corresponding to the straight enumeration of a disconnected graph
Algorithm 6: Robinson($A, \psi, \tau, \alpha$)

**input:** a weak linear order $\psi$ of $V = [n]$, a linear order $\tau$ of $V$ compatible with $\psi$, a nonnegative matrix $A \in S^n$ and a node $\alpha$

**output:** A PQ-tree $T_\alpha$ 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. $\text{CO-Lex-BFS}(G, \psi, \tau)$ 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 $V_\omega$ ($\omega \in [c]$) of $G$ as follows:
   - $\Phi$ : all $\omega$ for which the connected component $V_\omega$ meets at least two blocks of $\psi$
   - $\Lambda$ : all $\omega$ for which $V_\omega$ is contained in a block $B_i$ containing no other component
   - for $i \in [p], \Omega_i$: all $\omega$ for which $V_\omega \subseteq B_i$ and $B_i$ contains at least two components
4. $\Phi = \emptyset$
5. for $\omega = 1, \ldots, c$ do
   - create a Q-node $\gamma_\omega$
   - if $\alpha_{\min} > 0$ then
     - $A[V_\omega] := A[V_\omega] - \alpha_{\min} J$ and $G[V_\omega]$ is modified accordingly
   end
   - $\phi_\omega = \text{Straightenumeration}(G[V_\omega], \sigma[V_\omega])$ (if $G[V_\omega]$ is a unit interval graph)
   - $\Phi_\omega = \text{Refine}(\psi[V_\omega], \phi_\omega)$ (if $\psi[V_\omega]$ and $\phi_\omega$ are compatible)
   - $\alpha'_{\min}$ is the smallest nonzero entry of $A[V_\omega]$
   - $A'[V_\omega]$ is obtained from $A[V_\omega]$ by setting its entries with value $\alpha'_{\min}$ to zero
   - if $A'[V_\omega]$ is diagonal then
     - create a PQ-tree $T_\omega$ rooted in $\gamma_\omega$ and whose children are P-nodes corresponding to the blocks of $\Phi_\omega$
   else
     - $\tau_\omega$ is a linear order of $V_\omega$ compatible with $\Phi_\omega$
     - $T_\omega = \text{Robinson}(A'[V_\omega], \Phi_\omega, \tau_\omega, \gamma_\omega)$
   end
end
6. $T_\alpha$ is the PQ-tree rooted in $\alpha$, build as follows:
7. $\omega = 1$
8. while $\omega \leq c$ do
   - if $\omega \in \Theta$ then
     - the children of $\gamma_\omega$ become children of $\alpha$ and remove $\gamma_\omega; \omega = \omega + 1$
   else
     - if $\omega \in \Lambda$ then
       - set $T_\omega$ as child of $\alpha$ (if $\alpha = \emptyset$, then set $\alpha = \gamma_\omega); \omega = \omega + 1$
     else
       - 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 $\alpha = \beta_i$)
       - foreach $\omega' \in \Omega_i$ do
         - set $\gamma_{\omega'}$ as children of $\beta_i$
       end
     end
   end
end
9. return: $T_\alpha$ or STOP
Finally, we modify Algorithm 5 by just giving the node $\alpha = \emptyset$ (i.e. undefined) as input to the first recursive call. The overall complexity of the algorithm after the above mentioned modifications is the same as for Algorithm 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 1. Furthermore, the operations on the PQ-tree are basic operations that do not increase the overall complexity of the algorithm.

Algorithm 7: Robinsonian($A$)

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

1. $\psi = (V)$
2. $\tau = (1, 2, \ldots, n)$
3. $\alpha = \emptyset$
4. $G$ is the support of $A$
5. $T = \text{Robinson}(A, \psi, \tau, \alpha)$
6. return: $T$ or “$A$ not Robinsonian”

5 Conclusions

We introduced a new combinatorial algorithm to recognize Robinsonian matrices, based on a divide-and-conquer strategy and 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. An obvious bound on $d$ is the number $L$ of distinct entries in the matrix. A first natural question is to find other better bounds on the depth $d$. Is $d$ in the order $O(n)$, where $n$ is the size of the matrix? 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 a 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 [21], 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 in the noisy case, when $A$ is not Robinsonian, for example by using (linear) certifying algorithms as in [20] to detect the edges and the nodes of the level graphs which create obstructions to being a unit interval graph.
Acknowledgements

This work was supported by the Marie Curie Initial Training Network “Mixed Integer Nonlinear Optimization” (MINO) grant no. 316647.

References

[1] 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.

[2] H.L. Bodlaender, T. Kloks, and R. Niedermeier. SIMPLE MAX-CUT for unit interval graphs and graphs with few P4s. *Electronic Notes in Discrete Mathematics*, 3:19–26, 1999.

[3] 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.

[4] A. Brandstädt, F.F. Dragan, and F. Nicolai. Lexbfs-orderings and powers of chordal graphs. *Discrete Mathematics*, 171(13):27–42, 1997.

[5] V. Chepoi and B. Fichet. Recognition of Robinsonian dissimilarities. *Journal of Classification*, 14(2):311–325, 1997.

[6] V. Chepoi and M. Seston. Seriation in the presence of errors: A factor 16 approximation algorithm for $l_\infty$-fitting Robinson structures to distances. *Algorithmica*, 59(4):521–568, 2011.

[7] J. Cohen, F. Fomin, P. Heggernes, D. Kratsch, and G. Kucherov. Optimal linear arrangement of interval graphs. In *Mathematical Foundations of Computer Science 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 267–279. Springer Berlin Heidelberg, 2006.

[8] D.G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004.

[9] 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.

[10] 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. SIAM, 1998.

[11] 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](http://math.sjtu.edu.cn/conference/Bannai/2013/data/20130629B/slides1.pdf).

[12] C.M.H de Figueiredo, J.Meidanis, and C.P de Mello. A linear-time algorithm for proper interval graph recognition. *Information Processing Letters*, 56(3):179–184, 1995.

[13] 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.
[14] M. Dom. Algorithmic aspects of the consecutive-ones property. *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59, 2009.

[15] F. Fogel, R. Jenatton, F. Bach, and A. d’Aspremont. Convex relaxations for permutation problems. In *Advances in Neural Information Processing Systems*, pages 1016–1024, 2013.

[16] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[17] P.C. Gilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

[18] M.C. Golumbic. *Algorithmic graph theory and perfect graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co., Amsterdam, The Netherlands, 2004.

[19] 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.

[20] 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.

[21] 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.

[22] J.Bang-Jensen, J. Huang, and L. Ibarra. Recognizing and representing proper interval graphs in parallel using merging and sorting. *Discrete Applied Mathematics*, 155(4):442–456, 2007.

[23] 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.

[24] 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.

[25] I. Liiv. Seriation and matrix reordering methods: An historical overview. *Statistical Analysis and Data Mining*, 3(2):70–91, 2010.

[26] P.J. Looges and S. Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993.

[27] R. Mahesh, C.P. Rangan, and A. Srinivasan. On finding the minimum bandwidth of interval graphs. *Information and Computation*, 95(2):218–224, 1991.

[28] B.G. Mirkin and S.N. Rodin. *Graphs and genes*. Biomathematics (Springer-Verlag). Springer, 1984.

[29] S. Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21–25, 1991.

[30] P. Préa and D. Fortin. An optimal algorithm to recognize Robinsonian dissimilarities. *Journal of Classification*, 31:1–35, 2014.
[31] F.S. Roberts. Indifference graphs. pages 139–146. Academic Press, 1969.

[32] F.S. Roberts. On the compatibility between a graph and a simple order. Journal of Combinatorial Theory, Series B, 11(1):28–38, 1971.

[33] F.S. Roberts. Graph theory and its applications to problems of society. Society for Industrial and Applied Mathematics, 1978.

[34] W.S. Robinson. A method for chronologically ordering archaeological deposits. American Antiquity, 16(4):293–301, 1951.

[35] 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.

[36] M. Seston. Dissimilarités de Robinson: algorithmes de reconnaissance et d’approximation. PhD thesis, Université de la Méditerranée, 2008.

[37] K. Simon. A new simple linear algorithm to recognize interval graphs. In H. Bieri and H. Noltemeier, editors, Computational geometry-methods, algorithms and applications, volume 553 of Lecture Notes in Computer Science, pages 289–308. Springer Berlin Heidelberg, 1991.
A  Example

We give a complete example of the algorithm. We consider the same matrix $A$ as the one used in the example in Section 5 of [30]. However, since [30] handles Robinsonian dissimilarities, we first transform it into a similarity matrix and thus we use instead the matrix $a_{\text{max}}J - A$, where $a_{\text{max}}$ denotes the largest entry in the matrix $A$. If we rename such a new matrix as $A$, it looks as follows:

$$A = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 \\
11 & 2 & 9 & 0 & 5 & 0 & 5 & 5 & 2 & 0 & 5 & 0 & 5 & 6 & 0 & 0 & 2 & 0 & 5 \\
11 & 2 & 0 & 9 & 0 & 8 & 5 & 10 & 0 & 5 & 0 & 5 & 2 & 0 & 0 & 10 & 0 & 8 \\
11 & 0 & 5 & 0 & 5 & 5 & 2 & 0 & 5 & 0 & 5 & 10 & 0 & 0 & 2 & 0 & 5 \\
11 & 0 & 3 & 0 & 0 & 0 & 3 & 0 & 3 & 0 & 0 & 10 & 3 & 0 & 9 & 0 \\
11 & 0 & 8 & 7 & 9 & 0 & 7 & 0 & 7 & 5 & 0 & 0 & 9 & 0 & 10 \\
11 & 0 & 0 & 0 & 10 & 0 & 6 & 0 & 0 & 5 & 8 & 0 & 5 & 0 \\
11 & 7 & 8 & 0 & 7 & 0 & 7 & 5 & 0 & 0 & 8 & 0 & 9 \\
11 & 6 & 0 & 10 & 0 & 8 & 8 & 5 & 0 & 0 & 6 & 0 & 7 \\
11 & 0 & 6 & 0 & 5 & 2 & 0 & 0 & 10 & 0 & 8 \\
11 & 0 & 6 & 0 & 0 & 4 & 9 & 0 & 5 & 0 \\
11 & 0 & 9 & 7 & 0 & 0 & 6 & 0 & 7 \\
11 & 0 & 0 & 9 & 6 & 0 & 10 & 0 \\
11 & 7 & 0 & 0 & 5 & 0 & 7 \\
11 & 0 & 0 & 2 & 0 & 5 \\
11 & 4 & 0 & 10 & 0 \\
11 & 0 & 4 & 0 \\
11 & 0 & 8 \\
11 & 0 \\
11
\end{pmatrix}$$

Here the red labels denote the original numbering of the elements. Throughout we will use the fact that adding any multiple of the all-ones matrix $J$ to the matrix $A$ does not change the Robinson(ian) property. The recursion tree computed by Algorithm 5 is shown in Figure 3 at page 29. The weak linear order at each node represents the weak linear order $\psi$ 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 4. Then, Algorithm 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 $A[V_1]$ induced by $V_1$ is shown below (after shifting the matrix by $a_{\text{min}} = 2$, i.e., substracting $2J$ from it).
If we invoke Algorithm 2 on the support $G[V_1]$, we get the following straight enumeration:

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

Note that since $\psi[V_1]$ has only one block, we do not need to compute the partition refinement in Algorithm 3 and then the common refinement is simply $\Phi = \phi$. The smallest nonzero value of the matrix in (4) is $a'_{\min} = 3$. Hence, we compute $A'[V_1]$ by setting to zero the entries of the matrix in (4) with value equal to $a'_{\min}$. Since $A'[V_1]$ is not diagonal, we make a recursion call, and we set $\psi = \Phi$. 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 input matrix $A[V_1]$ is obtained by setting to zero the entries of the matrix in (4) with value at most 3. 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 3 it is easy to see that the common refinement $\Phi$ of $\psi$ and $\phi$ is exactly $\phi$. The smallest nonzero value of $A[V_1]$ is now $a'_{\min} = 4$. Hence, we compute $A'[V_1]$, which is not diagonal and thus we make a recursion call, setting $\psi = \Phi$ and renaming $A'[V_1]$ as $A[V_1]$.

### 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 (4) with value at most 4. 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 $\psi$ 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 $a'_{\min} = 5$. Hence, we compute $A'[V_1]$, which is not diagonal and thus 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 (4) with value at most 5. The support of this matrix is still connected, and its straight enumeration is:

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

The common refinement with $\psi$ 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 $a'_{\min} = 6$. Hence, we compute $A'[V_1]$, which is not diagonal and thus we make a recursion call, setting $\psi = \Phi$ and renaming $A'[V_1]$ as $A[V_1]$.
at most 5. The support of this matrix is not connected, and thus Algorithm 1 will detect the following connected components:

\[ V_{11} = \{1, 3, 14\}, \]
\[ V_{12} = \{13, 8, 11\}, \]
\[ V_{13} = \{5, 7, 19, 9, 17, 2\}. \]

We analyze each connected component independently.

### 1.3.1 Connected component \( V_{11} \), level 4

The common refinement is \( \Phi = (\{1\}, \{3\}, \{14\}) \) and \( a'_{\text{min}} = 7 \). We make a recursion call, finding two connected components \( \{1\} \) and \( \{3, 14\} \), and then we stop, because the first one has only one vertex, while the submatrix corresponding to the second one is diagonal (after shifting).

### 1.3.2 Connected component \( V_{12} \), level 4

Since \( a_{\text{min}} = 6 \), we first “shift” the input submatrix. Then, the common refinement is \( \Phi = (\{13\}, \{11\}, \{8\}) \) and \( a'_{\text{min}} = 7 \) (i.e. \( =1 \) after shifting). We make a recursion call, finding two connected components \( \{13\} \) and \( \{11, 8\} \), and then we stop, because the first one has only one vertex, while the submatrix corresponding to the second one is diagonal (after shifting).

### 1.3.3 Connected component \( V_{13} \), level 4

Since \( a_{\text{min}} = 6 \), we first “shift” the input submatrix. Then, the common refinement is \( \Phi = (\{7\}, \{19\}, \{5\}, \{9, 17\}, \{2\}) \) and \( a'_{\text{min}} = 7 \) (i.e. \( =1 \) after shifting). We update \( A'[V_1] \), and we make a recursive call because it is not diagonal. The new input matrix is then given by the submatrix in (4) restricted to \( V_{13} \) by setting to zero the entries with value at most 7. The support of this matrix is not connected, and thus Algorithm 1 will detect the following connected components:

\[ V_{131} = \{7\}, \]
\[ V_{132} = \{19, 5\}, \]
\[ V_{133} = \{9, 17, 2\}. \]

We then split the problem over the connected components. The first one has only one vertex while the second and the third one are diagonal (after shifting). This was the last recursion node open.

Therefore, we get that the final common refinement of the level graphs of the matrix in (4) is:

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

and the PQ-tree \( T_1 \) computed by the algorithm is reported in Figure 4 at page 30.

### 2.0 Connected component \( V_2 \), level 0

The submatrix \( A[V_2] \) induced by \( V_2 \) is reported below (after shifting the matrix by \( a_{\text{min}} = 3 \)).

\[
A[V_2] = \begin{pmatrix}
4 & 6 & 10 & 12 & 15 & 16 & 18 \\
4 & 8 & 0 & 0 & 7 & 0 & 6 \\
6 & 8 & 7 & 3 & 2 & 5 & 2 \\
10 & 8 & 3 & 1 & 6 & 2 \\
12 & 8 & 6 & 3 & 7 \\
15 & 8 & 1 & 7 \\
16 & 8 & 1 \\
18 & 8
\end{pmatrix}
\]
If we invoke Algorithm 2 on the support $G[V_2]$, we get the following straight enumeration:

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

Note that since $\psi[V_2]$ has only one block, we do not have to compute the partition refinement in Algorithm 3 and then the common refinement is simply $\Phi = \phi$. The smallest nonzero value of the matrix in (5) is $a_{\min}' = 1$. Hence, we compute $A'[V_2]$ by setting to zero the entries of the matrix in (5) with value equal to $a_{\min}'$. Since $A'[V_2]$ is not diagonal, we make a recursion call, and we set $\psi = \Phi$. To simplify notation, we shall again rename $A'[V_2]$ as $A[V_2]$ after every iteration.

### 2.1 Connected component $V_2$, level 1

The input matrix $A[V_2]$ is obtained by setting to zero the entries of the matrix in (5) with value at most 1. The support of this matrix is still connected, and its straight enumeration is:

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

If we invoke Algorithm 3 it is easy to see that the common refinement is $\Phi = \phi$. The smallest nonzero value of $A[V_2]$ is now $a_{\min}' = 2$. Hence, we compute $A'[V_2]$, which is not diagonal and thus we make a recursion call, setting $\psi = \Phi$ and renaming $A'[V_2]$ as $A[V_2]$ for the next iteration.

### 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) with value at most 2. 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:

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

The smallest nonzero value of $A[V_2]$ is now $a_{\min}' = 3$. Hence, we compute $A'[V_2]$, which is not diagonal and thus we make a recursion call, setting $\psi = \Phi$ and renaming $A'[V_2]$ as $A[V_2]$ for the next iteration.

### 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) with value at most 3. The support of this matrix is still connected, and its straight enumeration is:

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

which is equal to the given $\psi$ (and thus will be the common refinement $\Phi$). The smallest nonzero value of $A[V_2]$ is now $a_{\min}' = 5$. Hence, we compute $A'[V_2]$, which is not diagonal and thus we make a recursion call, setting $\psi = \Phi$ and renaming $A'[V_2]$ as $A[V_2]$ for the next iteration.

### 2.4 Connected component $V_2$, level 4

The input matrix $A[V_2]$ is obtained by setting to zero the entries of the matrix in (5) with value at most 5. The support of this matrix is not connected, and thus Algorithm 1 will detect two connected components.

$$V_{21} = \{4, 15, 18, 12\},$$
$$V_{22} = \{6, 10, 16\}.$$
We then split the problem over the connected components.

2.4.1 Connected component $V_{21}$, level 5
The common refinement is $\Phi = (\{4\}, \{15\}, \{18\}, \{12\})$ and $a'_\text{min} = 6$. We then make a recursion call and we find again a connected graph. Again the common refinement does not change (in fact the blocks are singletons) and now $a'_\text{min} = 7$. Finally, the submatrix $A'[V_{21}]$ is diagonal, so we prune the node.

2.4.2 Connected component $V_{22}$, level 5
The common refinement is $\Phi = (\{6\}, \{10\}, \{16\})$ (again the blocks are singletons) and $a'_\text{min} = 6$. But now $A'[V_{22}]$ is diagonal, so we prune the node. This was the last recursion node open.

Therefore, we get that 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 $T_2$ computed by the algorithm is shown in Figure 5 at page 30. 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 $\psi$ (which at the beginning is $\psi = ([n])$), then we create a P-node (named $\alpha$ since it is the ancestor) whose children are the subtrees $T_1$ and $T_2$. The final PQ-tree is shown in Figure 6 at page 30 and is equivalent to the one returned by [30].

Note that, using the fact that a common refinement which is a linear order cannot be refined anymore, the depth could be lowered to $d = 4$ (since the right branch would have depth $d = 3$). Understanding this and other possible speed up will be the subject of future work.
Figure 3: Recursion tree. The weak linear order at each node represent the weak linear order $\psi$ given as input to the recursion node, while the number on the edge between two nodes denotes the minimum value in $A$ set to zero before making a new recursion call. The red weak linear orders are the pruned nodes.
Figure 4: The PQ-tree corresponding to the common refinement of level graphs of $A[V_1]$

Figure 5: The PQ-tree corresponding to the common refinement of level graphs of $A[V_2]$

Figure 6: The PQ-tree corresponding to the permutations reordering $A$ as a Robinson matrix