A Lex-BFS-based recognition algorithm for Robinsonian matrices

Monique Laurent[1,2] and Matteo Seminaroti[1]

[1] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
{M.Laurent, M.Seminaroti}@cwi.nl
[2] Tilburg University, 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 nonnegative 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 [25] for chronological dating, asks to order objects in such a way that similar objects are ordered close to each other, and it has applications in different fields (see [18]).

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 π 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. Robinsonian matrices play an important role in several hard combinatorial optimization problems and recognition algorithms are important

in designing heuristic and approximation algorithms when the Robinsonian property is desired but the data is affected by noise (see e.g. [5],[12],[17]). In the last decades, different characterizations of Robinsonian matrices have appeared in the literature, leading to different polynomial time recognition algorithms. Most characterizations are in terms of interval (hyper)graphs.

A graph $G = (V, E)$ is an *interval graph* if its nodes can be labeled by intervals of the real line so that adjacent nodes correspond to intersecting intervals. Interval graphs arise frequently in applications and have been studied extensively in relation to hard optimization problems (see e.g. [2],[6],[20]). A binary 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. 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 [13].

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 entry 1 when the corresponding hyperedge contains the corresponding vertex. Then, H is an *interval hypergraph* if its incidence matrix has C1P, i.e., its vertices can be ordered so that hyperedges are intervals.

Given a dissimilarity matrix $A \in \mathcal{S}^n$ and a scalar α, 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_α. Let $\mathcal{B}$ denote the collection of all the balls of A and $H_\mathcal{B}$ denote the corresponding *ball hypergraph*, with vertex set $V = [n]$ and with $\mathcal{B}$ as set of hyperedges. 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. Most of the existing algorithms are then based on the fact that a matrix A is Robinsonian if and only if the ball hypergraph $H_\mathcal{B}$ is an interval hypergraph or, equivalently, if the intersection graph $G_\mathcal{B}$ is an interval graph (see [21,22]).

Mirkin and Rodin [21] 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 Booth and Leuker [3] to check whether the incidence matrix has C1P. Later, Chepoi and Fichet [4] 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 Seston [27], who improved the algorithm to $O(n^2 \log n)$ by constructing paths in the threshold graphs of A. Very recently, Préa and Fortin [22] presented a more sophisticated $O(n^2)$ algorithm, which uses the fact that the maximal cliques of the graph $G_\mathcal{B}$ 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.

In this paper we introduce a new combinatorial algorithm to recognize Robinsonian matrices. Our approach differs from the existing ones in the sense that it is not directly related to interval (hyper)graphs, but it is based on a new characterization of Robinsonian matrices in terms of *straight enumerations* of unit interval graphs (Section 3). Unit interval graphs are a subclass of interval graphs, where the intervals labeling the vertices are required to have unit length. Several linear time recognition algorithms exist, based in particular on characterizations of unit interval graphs in terms of straight enumerations, that are special orderings of the vertices [8,7].

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. The most difficult task carried out is instead 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 [26,14]). Following [7], we in fact use the variant Lex-BFS+ introduced by [28] 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 6). 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 4.2).

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 [9]. Alternatively, one could check whether the incidence matrix M of the ball hypergraph of A has C1P, using the Lex-BFS based algorithm of [14], 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.

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 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.

2 Preliminaries

Throughout $\mathcal{S}^n$ denotes the set of symmetric $n \times n$ matrices. Given a permutation π of $[n]$ and a matrix $A \in \mathcal{S}_n$, $A_\pi := (A_{\pi(i)\pi(j)})_{i,j=1}^n \in \mathcal{S}^n$ is the matrix obtained by permuting both the rows and columns of A simultaneously according to π. 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* ψ 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 $B_1 <_\psi \ldots <_\psi B_p$. When all classes B_i are singletons then ψ is a linear order (i.e., total order) of V.

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

The following notions of compatibility and refinement will play an important role in our treatment. Two weak linear orders ψ_1 and ψ_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* 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.

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 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 [16]). 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*.

The following characterization of unit interval graphs in terms of straight enumerations will play a central role in our paper.

Theorem 1 (Unit interval graphs and straight enumerations). *[10] 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 connected component induces a straight enumeration of G. Several alternative characterizations for unit interval graphs are known (see [7] and references therein), including the following ones.

Theorem 2. *A graph $G = (V, E)$ is a unit interval graph if and only if it satisfies any of the following equivalent conditions:*

(i) **(3-vertex condition)** *[19] There is a linear ordering π of V such that, for all $x, y, z \in V$, $x <_\pi y <_\pi z$ and $\{x, z\} \in E$ implies $\{x, y\}, \{y, z\} \in E$.*

(ii) **(Neighborhood condition)** *[23] There is a linear ordering π of V such that for any $x \in V$ the vertices in $N[x]$ are consecutive with respect to π.*

3 Robinsonian matrices and unit interval graphs

In this section we characterize Robinsonian matrices in terms of straight enumerations of unit interval graphs. 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 [23]. Furthermore, as observed, e.g., by Corneil et al. [8], 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 2(ii).

Lemma 1. *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 3. *Let $G = (V, E)$ be a graph. A linear order π 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 ψ is compatible with π, i.e., satisfies:*

$$\forall x, y \in V \quad with \quad x \neq_\psi y \qquad x <_\pi y \quad \Longleftrightarrow \quad x <_\psi y. \tag{1}$$

Hence, in order to find the permutations reordering a given binary 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 [8,10], 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 [11] and references therein).

We now consider a general (nonbinary) matrix A. We first introduce its 'level graphs', the analogues for similarity matrices of the threshold graphs for dissimilarities. Let $\alpha_0 < \alpha_1 < \cdots < \alpha_L$ 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. Let J be the all ones matrix. Clearly, $\pm J$ is a Robinson matrix. Hence, we may and will assume, without loss of generality, that $\alpha_0 = 0$. Then, A is nonnegative and $G^{(1)}$ is its support graph.

As already observed by Roberts [24], Robinson matrices can be decomposed as conic combinations of binary Robinson matrices (up to a translation by the all-ones matrix). We omit the proof, which is easy.

Lemma 2. *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, $A = \alpha_0 J + \sum_{\ell=1}^{L} (\alpha_\ell - \alpha_{\ell-1}) A_{G^{(\ell)}}$.*

Combining the links between binary Robinsonian matrices and unit interval graphs (Lemma 1) and between reorderings of binary Robinsonian matrices and straight enumerations of unit interval graphs (Theorem 3) together with the decomposition result of Lemma 2, we obtain the following characterization of Robinsonian matrices.

Theorem 4. *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 $G^{(1)}$, ..., $G^{(L)}$ 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 straight enumerations of $G^{(1)}, \ldots, G^{(L)}$, whose corresponding common refinement is compatible with π.

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 4. 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. 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 $G^{(1)}, \ldots, G^{(\ell-1)}$.

Our algorithm consists of three main subroutines: *CO-Lex-BFS* (Algorithm 1), a variation of Lex-BFS, which finds and orders the connected components of the level graphs; *Straight_enumeration*, which computes the straight enumeration of a connected graph as in [7]; *Refine* (Algorithm 2), a variation of partition refinement, which finds the common refinement of two weak linear orders. These subroutines are used in the recursive algorithm *Robinson* (Algorithm 3).

Component ordering Our first subroutine is *CO-Lex-BFS* (where CO stands for 'Component Ordering') in Algorithm 1. 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 (one can show that this is possible if G admit a straight enumeration compatible with ψ).

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* appplied to $(G[V_\omega], \sigma_\omega)$, where $\sigma_\omega = \sigma[V_\omega]$ and σ is the vertex order returned by *CO-Lex-BFS*(G, ψ, τ).

This routine is essentially the 3-sweep unit interval graph recognition algorithm of Corneil [7] which, briefly, computes three times a Lex-BFS (each is named a *sweep*) and use the vertex ordering coming from the previous sweep to break ties in the search for the next sweep. The only difference with respect to Corneil's algorithm is that we save the first sweep, because we use the order σ_ω given by *CO-Lex-BFS*.

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). If G is connected, its straight enumeration ϕ is unique up to reversal and one can show that the 3-sweep Lex-BFS algorithm implicitly returns it correctly orientated with respect to ψ. 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 (see Section 4.2). However, for now we are interested in finding *one* common refinement, and the arbitrary choice made does not affect the correctness of the algorithm.

Refinement of weak linear orders Given two weak linear orders ψ and ϕ on V, our second subroutine *Refine* in Algorithm 2 computes their common refinement $\Phi = \psi \wedge \phi$ (if it exists).

Main algorithm We can now describe our main algorithm *Robinson*(A, ψ, τ). Given a nonnegative matrix $A \in \mathcal{S}_n$, a weak linear order ψ and an order τ of $V = [n]$ that are compatible, it either returns a weak linear order Φ of V compatible with ψ and with straight enumerations of the level graphs of A, or

Algorithm 1: *CO-Lex-BFS*(G, ψ, τ)

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

mark all the vertices as unvisited
u is the first vertex appearing in τ
$label(u) = |V|$
$\omega = 1$
$V_\omega, B_\omega^{\min}, B_\omega^{\max} = \emptyset$
foreach $v \in V \setminus u$ **do**
    $label(v) = \emptyset$
for $i = |V|, \ldots, 1$ **do**
    let S be the set of unvisited vertices with the lexicographically largest label
    pick the vertex v in S appearing first in τ and mark it as visited
    $\sigma(v) = |V| + 1 - i$
    if $label(v) = \emptyset$ **then**
        if $V_\omega \subseteq B_{\omega-1}^{\min}$ **then**
            swap V_ω and $V_{\omega-1}$ and modify σ accordingly
        else
            if $B_\omega^{\min} <_\psi B_{\omega-1}^{\max}$ *or if there exists a block* B *of* ψ *such that* $B \not\subseteq V_\omega$ *and* $B_\omega^{\min} <_\psi B <_\psi B_\omega^{\max}$ **then**
                stop (no ordering of the components compatible with ψ exists)
        $\omega = \omega + 1$
        $V_\omega = \emptyset$
    $V_\omega = V_\omega \cup \{v\}$
    $B_\omega^{\min}$ is the first block in ψ which meets V_ω
    $B_\omega^{\max}$ is the last block in ψ which meets V_ω
    foreach *unvisited vertex* w *in* $N(v)$ **do**
        append i to $label(w)$
return $(V_1, \ldots, V_c)$ and σ, or STOP

it indicates that such Φ does not exist. The idea behind our algorithm is to 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 to 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 3.

Algorithm 2: *Refine*(ψ, ϕ, τ)

input: two weak linear orders $\psi = (B_1, \ldots, B_p)$ and $\phi = (C_1, \ldots, C_q)$ of V, and a linear order τ of V compatible with ψ
output: their common refinement $\Phi = \psi \wedge \phi$, or STOP (ψ and ϕ are not compatible)

$B^{\max}$ is the last block of ψ meeting C_1
if *there exists a block* B *of* ψ *such that* $B <_\psi B^{\max}$ *and* $B \not\subseteq C_1$ **then**
  stop (ψ and ϕ are not compatible)
else
  $W = V \setminus C_1$
  $\Phi = (\psi[C_1], \mathit{Refine}(\psi[W], \phi[W], \tau[W]))$
return Φ or STOP

Algorithm 3: *Robinson*(A, ψ, τ)

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

G is the support of A
CO-Lex-BFS(G, ψ, τ) returns a linear order $(V_1, \ldots, V_c)$ of the connected components of G compatible with ψ (if it exists) and a vertex order σ
$\Phi = \emptyset$
for $\omega = 1, \ldots, c$ **do**
  $a_{\min}$ is the smallest entry of $A[V_\omega]$
  if $a_{\min} > 0$ **then**
    $A[V_\omega] := A[V_\omega] - a_{\min} J$ and $G[V_\omega]$ is its updated support
  $\phi_\omega = \mathit{Straight\_enumeration}(G[V_\omega], \sigma[V_\omega])$ (if $G[V_\omega]$ is a unit interval graph)
  $\Phi_\omega = \mathit{Refine}(\psi[V_\omega], \phi_\omega)$ (if $\psi[V_\omega]$ and ϕ_ω are compatible)
  $a'_{\min}$ is the smallest nonzero entry of $A[V_\omega]$
  $A'[V_\omega]$ is obtained from $A[V_\omega]$ by setting its entries with value $a'_{\min}$ to zero
  if $A'[V_\omega]$ *is diagonal* **then**
    $\Phi = (\Phi, \Phi_\omega)$
  else
    τ_ω is a linear order of V_ω compatible with Φ_ω
    $\Phi = (\Phi, \mathit{Robinson}(A'[V_\omega], \Phi_\omega, \tau_\omega))$
return: Φ or STOP

The final algorithm is Algorithm 4. 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 4, and whose leaves (i.e. the pruned nodes) are the subproblems whose corresponding submatrices are diagonal.

Algorithm 4: *Robinsonian*(A)

input: a nonnegative matrix $A \in \mathcal{S}^n$
output: a permutation π such that A_π is Robinson or stating that A is not Robinsonian

1 $\psi = (V)$
2 $\tau = (1, 2, \ldots, n)$
3 Φ=*Robinson*(A, ψ, τ)
4 π is a linear order of V compatible with Φ
5 **return**: π or "A not Robinsonian"

Correctness and complexity The correctness of Algorithm 4 follows directly from the correctness of Algorithm 3, which is shown by the next theorem.

Theorem 5. *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 3 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 3 stops then there do not exist straight enumerations of the level graphs of A that are pairwise compatible and compatible with ψ.

For the complexity analysis, we assume that $A \in \mathcal{S}^n$ is nonnegative and is given as weighted adjacency list. One can show that the three subroutines run in linear time in the size of the input. Hence, we have the following result.

Theorem 6. *Let $A \in \mathcal{S}^n$ be a nonnegative matrix and let m be the number of (upper diagonal) nonzero entries of A. Algorithm 4 recognizes whether A is a Robinsonian matrix in time $O(d(m + n))$, where d is the depth of the recursion tree created by Algorithm 4. Moreover, $d \leq L$, where L is the number of distinct nonzero entries of A.*

4.2 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 $\mathcal{T}$ 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 $\mathcal{T}$ 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. Given a node α of $\mathcal{T}$, $\mathcal{T}_\alpha$ denotes the subtree of $\mathcal{T}$ with root α.

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. If G is not connected, let $V_1, \ldots, V_c$ 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 P-node α as ancestor, whose children are the subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_c$.

We now indicate how to modify Algorithms 3 and 4 in order to return a PQ-tree $\mathcal{T}$ encoding all the permutations ordering A as a Robinson matrix. We modify Algorithm 3 by taking as input, beside the matrix A, the weak linear order ψ and the linear order τ compatible with ψ, also a node α. Then, the output is a PQ-tree $\mathcal{T}_\alpha$ rooted in α, representing all the possible weak linear orders Φ 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, τ). 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 3, we create a Q-node γ_ω. After the common refinement Φ_ω (of $\psi[V_\omega]$ and the straight enumeration ϕ_ω of $G[V_\omega]$) have 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*$(A'[V_\omega], \Phi_\omega, \tau_\omega, \gamma_\omega)$.

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, τ). 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 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 4 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 4. 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.

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 [16], 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 [15] 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 $P4$s. *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. V. Chepoi and B. Fichet. Recognition of Robinsonian dissimilarities. *Journal of Classification*, 14(2):311–325, 1997.
5. 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.
6. J. Cohen, F. Fomin, P. Heggernes, D. Kratsch, and G. Kucherov. Optimal linear arrangement of interval graphs. In Rastislav Krlovi and Pawe Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 267–279. Springer Berlin Heidelberg, 2006.
7. D.G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004.
8. 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.
9. 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`.
10. X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25(2):390–403, February 1996.
11. M. Dom. Algorithmic aspects of the consecutive-ones property. *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59, 2009.
12. 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.
13. D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.
14. 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.
15. P. Hell and J. Huang. Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs. *SIAM J. Discret. Math.*, 18(3):554–570, 2005.
16. P. Hell, R. Shamir, and R. Sharan. A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM J. Comput.*, 31(1):289–305, January 2002.
17. 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.
18. I. Liiv. Seriation and matrix reordering methods: An historical overview. *Statistical Analysis and Data Mining*, 3(2):70–91, 2010.

19. P.J. Looges and S. Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993.
20. R. Mahesh, C.P. Rangan, and A. Srinivasan. On finding the minimum bandwidth of interval graphs. *Information and Computation*, 95(2):218–224, 1991.
21. B.G. Mirkin and S.N. Rodin. *Graphs and genes*. Biomathematics (Springer-Verlag). Springer, 1984.
22. P. Préa and D. Fortin. An optimal algorithm to recognize Robinsonian dissimilarities. *Journal of Classification*, 31:1–35, 2014.
23. F.S. Roberts. Indifference graphs. *Proof Techniques in Graph Theory: Proceedings of the Second Ann Arbor Graph Theory Conference (F. Harary, Ed.), Academic Press, New York*, pages 139–146, 1969.
24. F.S. Roberts. *Graph theory and its applications to problems of society*. Society for Industrial and Applied Mathematics, 1978.
25. W.S. Robinson. A method for chronologically ordering archaeological deposits. *American Antiquity*, 16(4):293–301, 1951.
26. D.J Rose and R.E. Tarjan. Algorithmic aspects of vertex elimination. In *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, pages 245–254, New York, NY, USA, 1975. ACM.
27. M. Seston. *Dissimilarités de Robinson: algorithmes de reconnaissance et d'approximation*. PhD thesis, Université de la Méditerranée, 2008.
28. 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.