Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

LOCATING EIGENVALUES OF SYMMETRIC MATRICES - A SURVEY[*]

CARLOS HOPPEN[†], DAVID JACOBS [‡], AND VILMAR TREVISAN[†]

Abstract. We survey algorithms for locating eigenvalues of symmetric matrices taking advantage of the underlying graph. We present applications in spectral graph theory.

Key words. Eigenvalue, Symmetric matrix, Locating eigenvalues, Spectral graph theory, Clique-width, Treewidth

AMS subject classifications. 05C50, 15A20, 15A18.

1. Introduction.

Real symmetric matrices and, more generally, complex Hermitian matrices are fundamental objects in mathematics in light of their rich theory and of their natural suitability for many applications. Among their many useful features, we consider spectral properties. Given an $n \times n$ complex matrix A, recall that a complex number λ is an *eigenvalue* of A associated with an *eigenvector* v if there is a nonzero vector $v \in \mathbb{C}^n$ such that $Av = \lambda v$; equivalently, λ is an eigenvalue of A if it satisfies the polynomial equation $\det(A - \lambda I) = 0$. The multiset of solutions of this polynomial equation is called the *spectrum* $\operatorname{Spec}(A)$ of A, and the *multiplicity* of λ as an eigenvalue of A is the number of occurrences of λ in $\operatorname{Spec}(A)$. As it turns out, if A is a real symmetric matrix, its spectrum consists of real eigenvalues and there is an orthogonal basis of $\mathbb{R}^n$ whose elements are eigenvectors of A.

In this paper, given a real symmetric matrix A and a real interval I, we are concerned with the problem of determining the number of eigenvalues of A lying in I (where each eigenvalue is counted according to its multiplicity). We call this *eigenvalue location*[*]. The eigenvalue location algorithms we survey here are based on the notion of *matrix congruence* and on a classical linear algebra result known as Sylvester's Law of Inertia. We recall that two matrices A and B are congruent if there exists an invertible matrix P such that

$$A = P^T B P. \tag{1.1}$$

The *inertia* of the matrix A is the triple $(n_+(A), n_-(A), n_0(A))$ that records the number of positive eigenvalues, the number of negative eigenvalues, and the multiplicity of the eigenvalue zero, respectively. We state Sylvester's Law of Inertia for easy reference.

THEOREM 1.1. *Two $n \times n$ real symmetric matrices are congruent if and only if they have the same inertia.*

Suppose that, for any real number $x \in \mathbb{R}$, we were able to find a diagonal matrix D congruent to $B_x = A - xI$. By Theorem 1.1, the number of positive, negative, and zero entries in the diagonal of D is

[*]Received by the editors on June 3, 2023. Accepted for publication on January 5, 2024. Handling Editor: Froilán Dopico. Corresponding Author: Vilmar Trevisan.

[†]Instituto de Matemática e Estatística, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil (choppen@ufrgs.br) (trevisan@mat.ufrgs.br)

[‡]School of Computing, Clemson University, Clemson, USA (dpj@clemson.edu)

[*]Even though we state the results in terms of real symmetric matrices, some of our algorithms can be easily adapted to perform eigenvalue location in Hermitian matrices. This will be pointed out in the text.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

the number of eigenvalues greater than, less than and equal to x in A. This would immediately allow us to determine the number of eigenvalues of A in any given interval. We state this as a corollary for future use.

Corollary 1.2. *Let A be a real symmetric matrix. For $x \in \mathbb{R}$, define $B_x = A + xI$. Let $n_+(B_x)$, $n_-(B_x)$ and $n_0(B_x)$ be the number of positive values, the number of negative values, and the number of zeros in the diagonal of a diagonal matrix congruent to B_x, respectively. The following hold:*

(a) *The number of eigenvalues of A (with multiplicity) that are greater than $-x$ is equal to $n_+(B_x)$.*
(b) *The number of eigenvalues of A (with multiplicity) that are less than $-x$ is equal to $n_-(B_x)$.*
(c) *The multiplicity of $-x$ as an eigenvalue of A is equal to $n_0(B_x)$.*

In particular, the number of eigenvalues (with multiplicity) of A in a real interval $(\alpha, \beta]$ is $n_+(B_{-\alpha}) - n_+(B_{-\beta})$.

Similarly, the number of eigenvalues of A in the intervals (α, β) and $[\alpha, \beta]$ is given by $n_+(B_{-\alpha}) - n_+(B_{-\beta}) - n_0(B_{-\beta})$ and by $n_+(B_{-\alpha}) + n_0(B_{-\alpha}) - n_+(B_{-\beta})$, respectively.

Of course, it is well known that symmetric matrices are diagonalizable, that is, that any symmetric matrix A may be written as

$$A = Q^{-1} D Q, \tag{1.2}$$

where D and Q are real matrices such that D is diagonal and $Q^{-1} = Q^T$. Note that this is a special case of (1.1). In particular, A and D are *similar* and therefore share the same spectrum. This implies that, rather than simply being helpful for locating eigenvalues, the diagonal entries of D are precisely the eigenvalues of A. However, computing[†] the decomposition (1.2) is expensive, requiring $\Omega(n^3)$ operations in the worst case using classical algorithms, where n is the order of the symmetric matrix. As an aside, we observe that reducing this problem to matrix multiplication allows the use of faster modern algorithms, such as the seminal algorithms of Strassen [56] and Coppersmith-Winograd [20], for instance. This is beyond the scope of this paper, but it is well known that the complexity of matrix multiplication cannot fall below $O(n^{2+o(1)})$ in the worst case.

In a purely computational perspective, we may view the algorithms surveyed in this paper as algorithms that sacrifice the precision provided by the decomposition in (1.2) for gain in complexity. Indeed, we consider eigenvalue location algorithms for which the input is a symmetric matrix A of order n in a particular domain and a real number x, while the output is a diagonal matrix D congruent to $B_x = A + xI$. Each algorithm runs in linear time by exploiting structural properties of the matrices in its domain to perform a sequence of elementary operations on the rows R_i and columns C_i of B_x:

$$R_i \longleftarrow R_i - \alpha R_j, \qquad C_i \longleftarrow C_i - \alpha C_j. \tag{1.3}$$

Performing the row operation in (1.3) is equivalent to the product $E_{ij}(\alpha) B_x$, where $E_{ij}(\alpha)$ is the elementary matrix whose entries coincide with the identity matrix, except for the entry ij, which is equal to $-\alpha$. Similarly, performing the column operation in (1.3) is equivalent to the product $B_x \tilde{E}_{ij}(\alpha)$, where $\tilde{E}_{ij}(\alpha)$ is the elementary matrix whose entries coincide with the identity matrix, except for the entry ji, which is equal to $-\alpha$. Since $E_{ij}(\alpha) = \tilde{E}_{ij}(\alpha)^T$ is invertible, we conclude that the matrix produced after performing the operations (1.3) is congruent to the original matrix. As a consequence, if we perform a sequence

[†]We observe that there is an abuse of terminology, where by *computing* we mean *approximating to a given precision.*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

of elementary row and column operations in a way that every row operation is immediately followed by the corresponding column operation, congruence is preserved. We call such pairs of elementary operations *congruence operations*. In the remainder of the paper, we call the diagonal matrix D a *diagonalization* of B_x and we say that the algorithm *diagonalizes* B_x.

However, more than being an efficient tool of approximating eigenvalues, eigenvalue location algorithms have been quite useful in the solution of problems related to the distribution of eigenvalues in a given graph class. This will be the focus of our survey. To fulfill this objective, we shall describe the algorithms and show how they can be a valuable tool in solving theoretical problems in spectral graph theory.

We encode the structure of a symmetric matrix $A = [a_{ij}]$ of order n by considering its *underlying graph*. This is the graph G with vertex set $V = \{v_1, \ldots, v_n\}$ such that v_i is adjacent to v_j if and only if $a_{ij} \neq 0$. In other words, the vertices of G correspond to the rows of A and there is an edge $\{v_i, v_j\}$ if and only if the element of A in row i and column j (and hence in row j and column i) is nonzero. Clearly, the matrix itself may be viewed as a weighted graph, where the support is given by its underlying graph, the diagonal elements are vertex weights and the nonzero off-diagonal elements are edge weights.

As we shall see, two structural properties will be particularly useful in the design of our algorithms. One of them is the *sparsity* of the matrix, by which we mean that the input matrix A has a relatively small number of nonzero off-diagonal elements. Suppose that it is possible to perform row and column operations, in an organized way, so as to eliminate all these elements without creating any new nonzero elements (or creating a very small number of new nonzero elements, which can also be eliminated). We would then have an algorithm that diagonalizes the original matrix very quickly. As it turns out, this is always possible when the underlying graph associated with the original matrix is a tree, which has led to the seminal algorithm in this line of research due to Jacobs and Trevisan [37]. Their algorithm was designed specifically as an eigenvalue location algorithm for the eigenvalues of the adjacency matrix of trees (and was based on an earlier algorithm to compute the characteristic polynomial of such matrices [35]), but, as new applications were considered, the approach was naturally extended to other contexts, see [27] for the Laplacian matrix and [16] for general symmetric matrices whose underlying graph is a tree.

An early successful attempt to exploit sparsity when the underlying graph is not a forest involved eigenvalue location for unicyclic graphs [18], that is, for connected graphs that contain a single cycle. A more general strategy for symmetric matrices whose underlying graph has a treelike structure has been proposed by Fürer, Hoppen, and Trevisan [30]. It exploits a structural decomposition of graphs known as a *tree decomposition* [50], which is associated with a parameter that measures how far a graph G is from a tree based on this decomposition, the *treewidth* of G. It so happens that G has treewidth equal to 1 if and only if it is a forest. The approach in [30] uses tree decompositions to diagonalize matrices whose underlying graph G has treewidth k with complexity $O(k^2n)$, under the assumption that a tree decomposition of G is given as part of the input. This is a linear-time algorithm for graphs with bounded treewidth, i.e., for families of graphs whose treewidth is bounded by an absolute constant. For instance, unicyclic graphs have treewidth 2. More generally, all *cacti* have treewidth at most 2, where a cactus is a graph for which any two cycles either are disjoint or have a single vertex in common.

The second structural property of the input matrix A that will be exploited by our algorithms is the occurrence of many rows and columns that are almost equal, in the sense that $R_i - R_j$ (and therefore $C_i - C_j$) have a small number of nonzero elements. To illustrate what we mean, assume that A is a (possibly dense) symmetric matrix such that all off-diagonal component lie in $\{0, \alpha\}$ for some real number α. Despite being a restrictive assumption, it is satisfied by several graph-based matrices, such as the adjacency matrix, the (combinatorial) Laplacian matrix, and the signless Laplacian matrix, where nonzero off-diagonal entries are always equal to 1, -1, and 1, respectively. Let $G = (V, E)$ be the underlying graph of A and assume that v_i and v_j have the same neighbors in $V \setminus \{v_i, v_j\}$. Clearly, if we perform the row operation $R_i - R_j$, all components in the resulting row vector will be 0 except possibly the components in columns i and j, so that many components can be eliminated with a single row operation; moreover, if we knew in advance that R_i and R_j satisfied the above property, there would be no need to actually perform any operation on the components of $R_i - R_j$ that are different from i and j, we could just replace them by 0.

The property described in the previous paragraph also appears naturally in graph families. Given a graph $G = (V, E)$ and a vertex $v \in V$, define the *(open) neighborhood* $N(v)$ of v as the set of vertices of G that are adjacent to v, that is, $w \in N(v)$ if and only if $\{v, w\} \in E$. The *closed neighborhood* of v is the set $N[v] = N(v) \cup \{v\}$. Two vertices u and v are said to be *duplicates* if $N(u) = N(v)$ and *coduplicates* if $N[u] = N[v]$. A well-known graph class consists of *complement reducible graphs* (or *cographs* for short), which have been studied in various contexts and can be characterized in many different ways, see [22] for more information. One such characterization is that a graph G is a cograph if and only if every induced subgraph of G on at least two vertices contains vertices u and v that are duplicates or coduplicates. This makes them an ideal graph family to apply the strategy described in the previous paragraph. A diagonalization algorithm based on this approach was introduced in 2018 by Jacobs, Trevisan, and Tura [41]. As was the case for trees, this approach can be extended to graphs that are described by means of a hierarchical decomposition known as *slick clique decomposition*, introduced in [29], which is closely related to the *clique decomposition* introduced by Courcelle and Olariu [24] more than two decades ago. As it turns out, a graph G has slick clique-width 1 if and only if it is a cograph, so that the slick clique-width may be viewed as a parameter that measures how far a graph G is from a cograph. Fürer and the current authors [29] have used this decomposition to diagonalize matrices whose underlying graph has slick clique-width k with complexity $O(k^2 n)$, which leads to very efficient algorithms for graph families with bounded slick clique-width, such as distance-hereditary graphs, which have slick clique-width at most 2. As before, complexity considerations are under the assumption that a slick clique decomposition of the underlying graph of the input matrix is part of the input.

This survey has been organized in three parts. The first part describes the algorithms that have been designed to take advantage of sparsity, starting with the algorithm for trees in Section 2 and extending it to general graphs through tree decompositions in Section 3. Even though our focus is on eigenvalue location, the algorithms find a diagonal matrix that is congruent to any given symmetric matrix whose underlying graph has the required structure, and this has other applications. Among other things, congruence may be used to compute the determinant and to classify Gram matrices associated with a quadratic form on a finite-dimensional vector space. The second part is concerned with algorithms that exploit the presence of similar rows and columns, namely the algorithm for cographs (Section 4) and its generalization to general graphs through clique decompositions (Section 5). The third part describes applications of these algorithms to the solution of problems in spectral graph theory.

Part I - Sparse graphs

Let A be a symmetric matrix of order n. To find a diagonal matrix D that is congruent to A, we may use a procedure that resembles the Gaussian elimination introduced in a first course in linear algebra, by which we eliminate the off-diagonal elements one row at a time using elementary operations. The idea is to

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

simply choose a nonzero diagonal element a_{jj} and use it to eliminate all the other nonzero elements on its column and row. For instance, the elements a_{ij} and a_{ji} are eliminated if we perform the following congruence operations:

$$R_i \leftarrow R_i - \frac{a_{ij}}{a_{jj}} R_j \quad C_i \leftarrow C_i - -\frac{a_{ij}}{a_{jj}} C_j,$$

leading to

$$\begin{array}{c} i \\ j \end{array}\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix} \longrightarrow \begin{array}{c} i \\ j \end{array}\begin{pmatrix} a_{ii} - a_{ij}^2/a_{jj} & 0 \\ 0 & a_{jj} \end{pmatrix}. \tag{1.4}$$

Of course, to complete the diagonalization, we would need to explain what to do when the diagonal element is zero, but there are nonzero elements on its row and column. Note that, in Gaussian elimination, we are allowed to interchange rows to get a nonzero element to act as a pivot, but here any row operation is immediately followed by the corresponding column operation, so that diagonal elements can only be replaced by other diagonal elements when rows (and therefore columns) are interchanged. At this point, we simply mention that this can be dealt with.

If A is sparse, we would also like to implement this process in an efficient way, that is, in a way that requires a small number of field operations. This can be done if we are able to avoid *fill-in*, by which we mean the set of matrix positions that were initially 0, but became nonzero at some point during the computation. The idea is to define a convenient ordering of the rows (and therefore of the columns) so that the first row in the order is the first to be diagonalized, and so on, while at the same time minimizing fill-in. In the context of Gaussian elimination, such an ordering is known as a *pivoting scheme* (or an *elimination ordering*), and we also use this terminology here. A possible problem with using a pre-defined pivoting scheme is that accidental cancelations may occur in the diagonal (an *accidental cancelation* happens when a nonzero element becomes zero because of an operation aimed at canceling a different element in the same row or column). Even though they may seem to be helpful at first sight, accidental cancelations can mess up the pivoting scheme, as the unexpected zero could be the element needed to eliminate nonzero elements in its row and column, as mentioned in the previous paragraph. In this part of the survey, we will present instances for which it is possible to design fast algorithms that exploit a convenient pivoting scheme, while successfully dealing with accidental cancelations. The first section is concerned with an algorithm for matrices whose underlying graph is a tree. The next section extends this to matrices with arbitrary underlying graph, but which is efficient when the underlying graph has a tree decomposition of small width.

2. Locating eigenvalues on trees.

Let A be an arbitrary symmetric matrix of order n whose underlying graph is a *tree*, that is, a connected and acyclic graph. We shall describe a linear-time algorithm that finds a diagonal matrix D that is congruent to A using the strategy described above. It uses a pivoting scheme based on the following structural characterization of trees, whose simple proof is included for completeness.

Lemma 2.1. *A graph $G = (V, E)$ is a tree if and only if there is a vertex-ordering $v_1, \ldots, v_n$ such that, for any $i < n$, v_i has a unique neighbor in $\{v_{i+1}, \ldots, v_n\}$. Moreover, if G is a tree and $v \in V$, there exists such a vertex-ordering for which $v_n = v$.*

Proof. First assume that T is a tree on $n \geq 1$ vertices. We proceed by induction on n. If $n \in \{1, 2\}$, any ordering of the vertex set satisfies the desired property. Assuming that, for some fixed $n \geq 2$, the result holds for any tree on n vertices, let T be a tree with $n + 1$ vertices. It is well known that T has a leaf w (actually, we may further choose $w \neq v$, where v is any fixed vertex, as any tree with at least two vertices contains at

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

least two leaves). Consider the tree $T' = T - w$. By induction, it contains an ordering $v'_1, \ldots, v'_n = v$ as in the statement of the theorem. By setting $v_1 = w$ and $v_i = v'_{i-1}$ for $i \in \{2, \ldots, n+1\}$, we obtain the desired ordering for T.

Conversely, assume that $G = (V, E)$ has an ordering $v_1, \ldots, v_n$ of the elements of V such that, for any $i < n$, v_i has a unique neighbor in $\{v_{i+1}, \ldots, v_n\}$. By induction, we show that $T_i = G[\{v_{n-i+1}, \ldots, v_n\}]$ is a tree for every $i \in \{1, \ldots, n\}$, and hence, $G = T_n$ is a tree. This is obvious for $i = 1$ and assume that it holds for some i such that $1 \leq i < n$. By hypothesis, T_{i+1} is obtained from the tree T_i by the addition of a vertex v_{n-i} with exactly one neighbor in T_i, so that T_{i+1} is clearly acyclic and connected. □

A concept that will be particularly useful is that of a *rooted tree*, that is a tree $T = (V, E)$ for which one of the vertices r is distinguished as the *root*. Each neighbor of r is regarded as a *child* of r, and r is called its *parent*. For each child c of r, all of its neighbors, except the parent, become its children. This process continues until all vertices except r have parents. A vertex that does not have children is called a *leaf* of the rooted tree. The *descendants* of a vertex in a rooted tree are defined as follows. If v is a leaf, then it has no descendants. Otherwise, the descendants of v are its children, together with their descendants. With this, given a rooted tree T and vertex v, we define the *subtree* T_v rooted at v as the subtree of T induced by v and all of its descendants. Moreover, the *depth* of a vertex in a rooted tree is its distance to the root, while the *depth of a rooted tree* is the maximum depth of a vertex.

We come back to an arbitrary symmetric matrix A of order n whose underlying graph is a tree T. Assume that T is rooted at any vertex v and consider a vertex-ordering $v_1, \ldots, v_n = v$ as in Lemma 2.1. It is easy to see that any vertex v_i must appear after all of its descendants, which is equivalent to saying that all vertices appear after all their children. This is called a *bottom-up ordering* of the rooted tree. An example is depicted in Fig. 1.

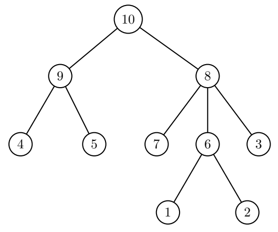


FIGURE 1. *A tree T whose vertices are ordered bottom-up.*

To diagonalize A, the aim is to perform congruence operations in a way that eliminates the off-diagonal elements of A without any fill-in. The intuition behind it is straightforward, and we further simplify the discussion by assuming first that A does not have any zeros in the diagonal and that no accidental cancelations occur. We start the process with a leaf v_j of T. The only nonzero off-diagonal element on its row and column is the element $a_{ij} = a_{ji}$ associated with the parent v_i of v_j. These elements are eliminated when we perform

$$R_i \leftarrow R_i - \frac{a_{ij}}{a_{jj}} R_j \qquad C_i \leftarrow C_i - \frac{a_{ij}}{a_{jj}} C_j,$$

which affects the matrix as in (1.4). Note that the only additional position modified by these operations is the position ii, which was already nonzero by assumption. At this point, we may view row j as diagonalized and we repeat the process for the remaining rows and columns, until we get a diagonal matrix. The assumption that the diagonal of A is fully nonzero implies that there was no fill-in. If we only assume that the diagonal

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

elements are nonzero when we use them to diagonalize their row/column (but could be initially 0), the fill-in might not be empty, but would have cardinality less than n.

2.1. The algorithm. To formalize this idea, and deal with the case when zeros appear in the diagonal, we say a vertex v has been *processed* if the submatrix corresponding to v and its descendants is diagonal. Before any operations are performed, set $A_0 = A$ and observe that the leaves of T are the only vertices of T that count as processed, as they have no descendants. At some point in the diagonalization process, let k be the least index such that v_k has not been processed and let A_{k-1} be the matrix obtained from A while processing vertices $v_1, \ldots, v_{k-1}$. We wish to process v_k. The submatrix of A_{k-1} induced by v_k and its descendants $j_1, \ldots, j_\ell$ is as in

$$\begin{pmatrix} d_{j_1} & 0 & 0 & \cdots & 0 & \alpha_{j_1k} \\ 0 & d_{j_2} & 0 & \cdots & 0 & \alpha_{j_2k} \\ 0 & 0 & d_{j_3} & \cdots & 0 & \alpha_{j_3k} \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \cdots & d_{j_\ell} & \alpha_{j_\ell k} \\ \alpha_{j_1k} & \alpha_{j_2k} & \alpha_{j_3k} & \cdots & \alpha_{j_\ell k} & a_{kk} \end{pmatrix}.$$

First assume that all descendants v_j of v_k for which $\alpha_{kj} \neq 0$ have diagonal value $d_j \neq 0$. By performing the congruence operations

$$R_k \leftarrow R_k - \frac{\alpha_{jk}}{d_j} R_j \qquad C_k \leftarrow C_k - \frac{\alpha_{jk}}{d_j} C_j,$$

the off-diagonal jk and kj entries of A_{k-1} are eliminated as follows

$$\begin{array}{c} j \\ k \end{array} \begin{pmatrix} d_j & \alpha_{jk} \\ \alpha_{jk} & d_k \end{pmatrix} \longrightarrow \begin{array}{c} j \\ k \end{array} \begin{pmatrix} d_j & 0 \\ 0 & d_k - \alpha_{jk}^2/d_j \end{pmatrix}.$$

The *net effect* for all descendants is

$$d_k = d_k - \sum_c \frac{\alpha_{ck}^2}{d_c}, \tag{2.5}$$

where c ranges over the descendants of v_k[‡]. Let A_k be the matrix obtained from A_{k-1} after these operations are performed. Note that no new nonzero entries are created because the children of v_k had already been processed in earlier steps. Moreover, the only entries of A_{k-1} that were modified while processing v_k are the entries jk and kj, where v_j is a descendant of v_k, which became 0, and the entry kk.

On the other hand, suppose that $d_j = 0$ for some descendant v_j of v_k such that $\alpha_{jk} \neq 0$, so that (2.5) is ill-defined. Select one such vertex v_j. We examine the submatrix corresponding to v_j and any sibling v_i, their parent v_k, and their grandparent v_ℓ, if present. As v_i and v_j have already been processed, the operations

$$R_i \leftarrow R_i - \frac{\alpha_{ik}}{\alpha_{jk}} R_j \qquad C_i \leftarrow C_i - \frac{\alpha_{ik}}{\alpha_{jk}} C_j,$$

annihilate the two off-diagonal entries of v_i as follows

$$\begin{array}{c} i \\ j \\ k \\ \ell \end{array} \begin{pmatrix} d_i & & \alpha_{ik} & \\ & 0 & \alpha_{jk} & \\ \alpha_{ik} & \alpha_{jk} & d_k & \alpha_{\ell k} \\ & & \alpha_{\ell k} & d_\ell \end{pmatrix} \longrightarrow \begin{array}{c} i \\ j \\ k \\ \ell \end{array} \begin{pmatrix} d_i & & 0 & \\ & 0 & \alpha_{jk} & \\ 0 & \alpha_{jk} & d_k & \alpha_{\ell k} \\ & & \alpha_{\ell k} & d_\ell \end{pmatrix}.$$

[‡]Actually, this sum only ranges over the children of v_k, but at this point we cannot justify this.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

We repeat this for all siblings of v_j. If v_k is the root, we ignore the next step. Otherwise, we can remove the two entries representing the edge between v_k and its parent v_ℓ with these operations:

$$R_\ell \leftarrow R_\ell - \frac{\alpha_{\ell k}}{\alpha_{jk}} R_j \qquad C_\ell \leftarrow C_\ell - \frac{\alpha_{\ell k}}{\alpha_{jk}} C_j.$$

This transforms the submatrix as follows

$$\begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 0 & \alpha_{jk} & \\ 0 & \alpha_{jk} & d_k & \alpha_{\ell k} \\ & & \alpha_{\ell k} & d_\ell \end{pmatrix} \longrightarrow \begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 0 & \alpha_{jk} & \\ 0 & \alpha_{jk} & d_k & 0 \\ & & 0 & d_\ell \end{pmatrix}.$$

Next we deal with the 0 in the diagonal. The congruence operations

$$R_k \leftarrow R_k - \frac{d_k}{2\alpha_{jk}} R_j \qquad C_k \leftarrow C_k - \frac{d_k}{2\alpha_{jk}} C_j$$

lead to

$$\begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 0 & \alpha_{jk} & \\ 0 & \alpha_{jk} & d_k & 0 \\ & & 0 & d_\ell \end{pmatrix} \longrightarrow \begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 0 & \alpha_{jk} & \\ 0 & \alpha_{jk} & 0 & 0 \\ & & 0 & d_\ell \end{pmatrix}.$$

Then, the operations

$$R_j \leftarrow R_j + \frac{1}{\alpha_{jk}} R_k \qquad C_j \leftarrow C_j + \frac{1}{\alpha_{jk}} C_k$$

produce

$$\begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 0 & \alpha_{jk} & \\ 0 & \alpha_{jk} & 0 & 0 \\ & & 0 & d_\ell \end{pmatrix} \longrightarrow \begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 2 & \alpha_{jk} & \\ 0 & \alpha_{jk} & 0 & 0 \\ & & 0 & d_\ell \end{pmatrix}.$$

Finally, the operations

$$R_k \leftarrow R_k - \frac{\alpha_{jk}}{2} R_j \qquad C_k \leftarrow C_k - \frac{\alpha_{jk}}{2} C_j$$

produce the diagonalized form

$$\begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 2 & \alpha_{jk} & \\ 0 & \alpha_{jk} & 0 & 0 \\ & & 0 & d_\ell \end{pmatrix} \longrightarrow \begin{array}{c} i \\ j \\ k \\ \ell \end{array}\begin{pmatrix} d_i & & 0 & \\ & 2 & 0 & \\ 0 & 0 & -\alpha_{jk}^2/2 & 0 \\ & & 0 & d_\ell \end{pmatrix}.$$

The elements that changed in the diagonal are

$$d_j = 2 \quad \text{and } d_k = -\frac{\alpha_{jk}^2}{2}. \tag{2.6}$$

Furthermore, if v_k is not the root, the fact that the entries ℓk and $k\ell$ have been eliminated may be interpreted as deleting the edge from v_k to its parent in the tree. We note that there cannot be any fill-in outside the diagonal, which justifies why no entry of the type α_{jk} can become nonzero for a descendant v_j of v_k that is not one of its children. It also shows why entries of type α_{jk}, where v_j is a child of v_k, can become zero.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

Indeed, while processing v_j, the diagonal element corresponding to one of its children could be 0, leading to the deletion of the edge $\{v_j, v_k\}$ as above. Finally, it shows that any nonzero entry of type α_{jk} associated with the unprocessed vertex v_k must be equal to the original entry a_{jk} in A, as it can only be modified if it becomes 0.

This discussion also justifies why we may perform this diagonalization process without operating on the matrix directly, but instead by operating on its underlying tree T if we assume access to a weighted tree with vertex and edge weights that record the entries of the original matrix. To keep track of the process, we need to update diagonal values, which we may interpret as weights associated with the vertices of T. As it turns out, off-diagonal values are not changed unless they become 0, so that nonzero off-diagonal entries may be viewed as weights on the edges of T that need not be updated. It is convenient to interpret the weight becoming 0 as deleting the corresponding edge from the tree, so that the algorithm proceeds on the remaining forest. Figure 2 shows the pseudocode for this algorithm. We note that, in our description, we omit any reference to the data structure used to record the entries of the sparse input matrix and instead assume that we have oracle access to these entries, i.e., we assume that there is an oracle that, given any pair i, j, outputs the value a_{ij}.

```
Input:  Real symmetric matrix $A = (a_{ij})$
        Underlying tree $T$ of $A$ whose vertices $v_1, \ldots, v_n$ are ordered bottom-up
Output: Diagonal matrix $D = \text{diag}(d_1, \ldots, d_n)$ congruent to $A$

Algorithm Diagonalize Tree($A$)
   initialize $d_i \longleftarrow a_{ii}$, for all $i$
   for  $k = 1$ to $n$
      if $v_k$ is a leaf
         then continue
         else if $d_c \neq 0$ for all children $c$ of $v_k$
            then $d_k \longleftarrow d_k - \sum \frac{(a_{ck})^2}{d_c}$, summing over all children of $v_k$
            else
               select one child $v_j$ of $v_k$ for which $d_j = 0$
               $d_k \longleftarrow -\frac{(a_{jk})^2}{2}$
               $d_j \longleftarrow 2$
               if $v_k$ has a parent $v_\ell$, then delete the edge $\{v_k, v_\ell\}$.
   end loop
```

FIGURE 2. *Diagonalizing A for a symmetric matrix A with an underlying tree.*

As is clear from the pseudocode in Fig. 2, the algorithm `Diagonalize Tree` does not perform the congruence operations described in this section, but instead only records changes on the diagonal values d_i of A. Because of this, only $O(n)$ space is necessary. It is not hard to see that the algorithm takes $O(n)$ operations, but we refer the reader to the original paper [37] for details.

In the following, we illustrate how the algorithm works with an example. Before doing this, we remark that it may be easily adapted to diagonalize Hermitian matrices.

REMARK 2.2. *A complex matrix H is Hermitian if is satisfies $H^* = H$, where H^* is the conjugate*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

transpose of H. The notion of congruence may be extended to Hermitian matrices by saying that H and F are $*$-congruent if there exists an invertible complex matrix U such that

$$H = U^*FU.$$

Sylvester's Law of Inertia may also be generalized to this context, and two Hermitian matrices are congruent if and only if they have the same inertia.

It is easy to modify the algorithm in Fig. 2 so that it computes a diagonal matrix congruent to a Hermitian matrix H whose underlying graph is a tree. Indeed, the only difference in the pseudocode is that the terms $(a_{ck})^2$ and $(a_{jk})^2$ are replaced by the products $a_{ck}\overline{a_{ck}}$ and $a_{jk}\overline{a_{jk}}$, respectively. To see why this is, first note that the diagonal of any Hermitian matrix consists of real numbers and that, to preserve congruence, any row operation of type $R_k \leftarrow R_k - \beta R_j$ for $\beta \in \mathbb{C}$ is followed by the operation $C_k \leftarrow C_k - \overline{\beta} C_j$. Indeed, the row operation is equivalent to the product $E_{kj}(\beta)H$, where $E_{kj}(\beta)$ is the elementary matrix whose entries coincide with the identity matrix, except for the entry kj, which is equal to $-\beta$. Similarly, performing the column operation is equivalent to the product $H\tilde{E}_{kj}(\overline{\beta})$, where $\tilde{E}_{kj}(\beta)$ is the elementary matrix whose entries coincide with the identity matrix, except for the entry jk, which is equal to $-\overline{\beta}$. Since $E_{ij}(\beta) = \tilde{E}_{ij}(\beta)^*$ is invertible, we conclude that the matrix produced after the row and column operations is $*$-congruent to the original matrix.

Going back to the operations that define the algorithm, note that row and column operations

$$R_k \leftarrow R_k - \frac{\alpha_{jk}}{d_j} R_j \qquad C_k \leftarrow C_k - \frac{\overline{\alpha_{jk}}}{d_j} C_j,$$

lead to the elimination of off-diagonal jk and kj entries of A_{k-1} as follows

$$\begin{array}{c} j \\ k \end{array}\begin{pmatrix} d_j & \overline{\alpha_{jk}} \\ \alpha_{jk} & d_k \end{pmatrix} \longrightarrow \begin{array}{c} j \\ k \end{array}\begin{pmatrix} d_j & 0 \\ 0 & d_k - {}^{\alpha_{jk}\overline{\alpha_{jk}}}/_{d_j} \end{pmatrix}.$$

Similarly, in (2.6), the term α_{jk}^2 is replaced by $\alpha_{jk}\overline{\alpha_{jk}}$ if the multiplicative constants used in row operations are replaced by their conjugates in column operations, which are again $*$-congruence-preserving operations.

$$L = \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ -1 & -1 & 0 & 3 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 2 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

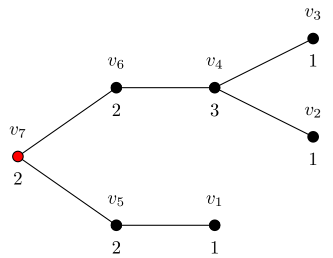


FIGURE 3. *Initial tree and its Laplacian matrix.*

EXAMPLE 2.3. *Consider L the Laplacian matrix of the tree T given in Fig. 3, that is, the matrix for which each diagonal entry ℓ_{ii} is the degree of vertex i and for which, given $i \neq j$, $\ell_{ij} = -1$ if $\{i, j\} \in E(T)$ and $\ell_{ij} = 0$ otherwise. Vertices are ordered from right to left, and from bottom to top. In particular, the first*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

three vertices in this ordering are the leaves of the tree and the root is the leftmost vertex, depicted in red. The diagonal values d_i at each step are represented below the corresponding vertex. Let L be the Laplacian matrix of this tree.

Initialization:
$d_i \leftarrow d_i - 1$
leaves are processed

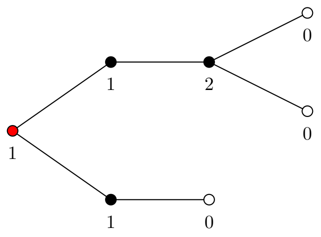


Figure 4. *Initialization.*

Suppose that we wish to determine the number of Laplacian eigenvalues greater than/smaller than/equal to 1. For this, we apply the algorithm Diagonalize(A) to the matrix $A = L - I$. On the tree of Fig. 4, we have the initial diagonal values, given by vertex degrees minus 1. If we represent processed vertices in white, the tree in Fig. 4 depicts the tree after all leaves are processed, as nothing happens when a leaf is processed.

Processing vertices
v_4 and v_5 that have
children with zero value

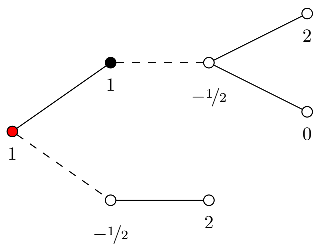


Figure 5. *Processing vertices having children with 0 value.*

After this, vertex v_6 became a leaf, and nothing happens when it is processed. The final step processes the root v_7, which has a single child v_6, whose value is $d_6 = 1 \neq 0$. Hence, $d_7 = 1 - \frac{(-1)^2}{1} = 0$. The diagonal values at the end of the algorithm are represented on the right of Fig. 6. Since there are two negative values, two positive values and two zeros, Corollary 1.2 implies that L has 2 eigenvalues smaller than 1, 3 eigenvalues greater than 1, while 1 is an eigenvalue of multiplicity 2. As it turns out, the approximate spectrum is given by $\{0, 0.22538, 1, 1, 2.18589, 3.36041, 4.22833\}$.

Processing v_6 (as a leaf)
Processing the root v_7:
$d_7 \leftarrow d_7 - \frac{(-1)^2}{d_6} = 1 - 1 = 0$

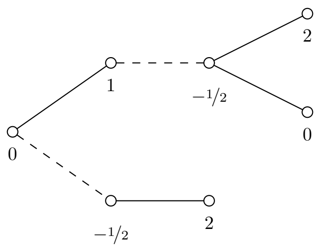


Figure 6. *Final values of the diagonalization.*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

3. Locating eigenvalues using tree decompositions.

3. Locating eigenvalues using tree decompositions. The aim here is to describe an algorithm that extends the approach of the previous section, which applies only to a specific class of matrices, to a general setting. It relies on the notion of *tree decomposition*, a well-known structural decomposition of graphs that has proved to be very useful in the design of algorithms.

3.1. Tree decomposition. The definition of tree decomposition that is popular today is due to Robertson and Seymour [50, 51], but notions that are similar or even equivalent have been independently studied in other contexts, and we refer to [6, 32] for earlier work. The study of tree decompositions and their applications has become a thriving research area, and our discussion will be restricted to the properties that are needed for our purposes. Readers who would like to know more are encouraged to read influential survey papers by Bodlaender [8, 10, 11], and the references therein.

A *tree decomposition* of a graph $G = (V, E)$ is a tree $\mathcal{T}$ with nodes $1, \ldots, m$, where each node i is associated with a set $B_i \subseteq V$, called a *bag*, satisfying the following properties:

(1) $\bigcup_{i=1}^{m} B_i = V$;
(2) For every edge $\{v, w\} \in E$, there exists B_i containing v and w;
(3) For any $v \in V$, the subgraph of $\mathcal{T}$ induced by the nodes that contain v is connected.

The *width* of the tree decomposition $\mathcal{T}$ is defined as $\max\{|B_i| - 1 \colon i \in V\}$ and the *treewidth* $tw(G)$ of graph G is the smallest k such that G has a tree decomposition of width k. Figure 7 depicts a tree decomposition of a graph G. Note that the distinction between the nodes of the tree and the bags associated with them is important, as different nodes may be associated with equal bags.

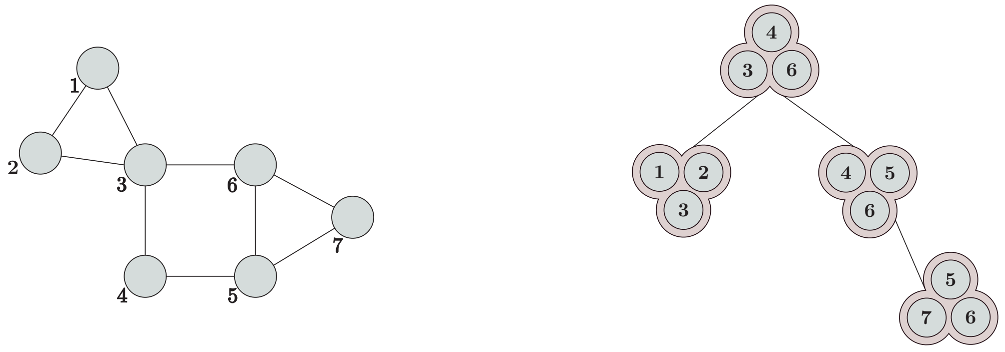


FIGURE 7. *A graph G and a tree decomposition $\mathcal{T}$ of G with four nodes and width 2. The elements of each bag are depicted within the corresponding node.*

It is known that if G is an n-vertex graph where $n \geq 2$, then $tw(G) = 1$ if and only if G is a forest, and $tw(G) = n - 1$ if and only if G is a complete graph. Regarding sparsity, one may easily show that graphs with small treewidth must be sparse. Precisely, any graph G with $n \geq k+1$ vertices and treewidth at most k has at most $kn - \binom{k+1}{2}$ edges. In general, the tree $\mathcal{T}$ with a single node 1 associated with the bag $B_1 = V(G)$ is a tree decomposition of G, so that $tw(G) \leq n - 1$. This illustrates an important difference between a tree decomposition of a graph and other decompositions, such as cotrees of cographs and k-expressions of graphs (which will also be introduced in this survey): tree decompositions do not determine the graph, i.e., different graphs may have the same tree decomposition. Indeed, a tree decomposition does not tell us that any edge actually lies in a graph G, but instead that a pair $\{v, w\}$ is *not* an edge of G if there is no bag containing the pair $\{v, w\}$. In other words, if we had an oracle with the ability to answer whether any given pair $\{v, w\}$ is

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

an edge of a graph G, a tree decomposition would restrict the set of pairs that need to be queried in order to expose the entire graph G. Under this point of view, the tree with a single node and bag $B_1 = V(G)$ does not give any information about the graph.

The above discussion shows that, for any subgraph H of G, we have $tw(H) \leq tw(G)$. More generally, we have $tw(H) \leq tw(G)$ for any *minor* H of G, that is, for any graph H that may be obtained from G by a sequence of vertex deletions, edge deletions and edge contractions, in any order. The *contraction* of an edge $\{v, w\}$ consists in replacing vertices v and w by a single vertex x adjacent with all vertices initially adjacent to v or w. Two useful lower bounds on $tw(G)$ are

$$tw(G) \geq \delta(G) \text{ and } tw(G) \geq \omega(G) - 1, \tag{3.7}$$

where $\delta(G)$ and $\omega(G)$ denote the minimum degree of G and the size of a maximum clique in G, respectively. From this, we deduce that $tw(G) \geq k - 1$ if G contains a complete graph K_k as a minor. Note that both bounds in (3.7) imply that the tree decomposition of G in Fig. 7 has minimum width, that is, $tw(G) = 2$.

The treewidth is a parameter of major interest in algorithm design, particularly in *parameterized complexity*, given that some NP-hard or even harder problems may often be solved in time $O(f(k)n^c)$, where c is a constant and f is a computable function that depends on the treewidth k of the n-vertex input graph. Even if f grows fast, for small values of k, such algorithms are often very practical. Readers are referred to Niedermeier [45] for a general introduction to parameterized complexity. Bodlaender and Koster [12] are particularly concerned with algorithms that take advantage of bounded treewidth.

Unfortunately, computing the treewidth of a graph G is NP-complete in general [2]; however, Bodlaender [9] introduced an algorithm that, for any fixed constant k, decides in time $f(k)n$ whether an n-vertex graph has treewidth at most k, and outputs a tree decomposition if the answer is positive (where $f(k) = k^{O(k^3)}$). Moreover, for many graph classes, there are polynomial-time algorithms that produce decompositions. We refer to [13, 14] for information about upper and lower bounds on the treewidth and for algorithms that compute tree decompositions. In our discussion, we shall always assume that a tree decomposition of the graph is given as part of the input.

The algorithm that we will present here uses tree decompositions with a particular structure, which have been introduced by Kloks [42]. For a graph G, a *nice tree decomposition* is a rooted tree decomposition $\mathcal{T}$ for which the root has an empty bag and each node i is of one of the following types:

(a) **(Leaf)** The node i is a *leaf* of $\mathcal{T}$;
(b) **(Introduce)** The node i *introduces* vertex v, that is, it has a single child j, $v \notin B_j$ and $B_i = B_j \cup \{v\}$.
(c) **(Forget)** The node i *forgets* vertex v, that is, i has a single child j, $v \notin B_i$ and $B_j = B_i \cup \{v\}$;
(d) **(Join)** The node i is a *join*, that is, it has two children j and ℓ, where $B_i = B_j = B_\ell$.

Combining the fact that the root of a nice tree decomposition has an empty bag with property (3) in the definition of tree decomposition, we see that every vertex of G must be forgotten exactly once in a nice tree decomposition. Figure 8 depicts a nice tree decomposition of the graph G of Fig. 7.

The next result states that, despite the additional structure, any graph G that admits a tree decomposition of width k also admits a nice tree decomposition of width k. We include part of the proof originally given in [30, Lemma 1] because it tells us how one such tree may be obtained from an arbitrary tree decomposition, which is quite useful for applications.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

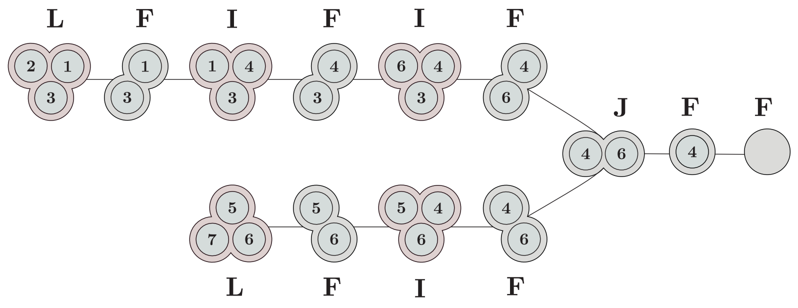


FIGURE 8. *A nice tree decomposition of the graph G of Fig. 7. The root is on the right, and each node is labeled L (leaf), I (introduce), J (join), or F (forget).*

LEMMA 3.1. *Let G be a graph of order n with a tree decomposition $\mathcal{T}$ of width k with m nodes. Then, G has a nice tree decomposition of the same width k with fewer than $4n-1$ nodes which can be computed from $\mathcal{T}$ in time $O((k \log k)m + kn)$.*

Proof. Let $\mathcal{T}$ be a tree decomposition of G with width k with m nodes, and fix an arbitrary node i as root. As a pre-processing step, we sort the vertices in each bag so that they are in increasing order (according to some arbitrary pre-determined order). This may be done in time $O((k \log k)m)$ using a standard sorting algorithm such as MergeSort. We modify the tree decomposition in a sequence of depth-first traversals. In the first traversal, every node whose bag is contained in the bag of its parent is deleted and its children are connected directly to the parent.

In the second traversal, whenever the bag of a node has fewer elements than the bag of its parent, we add elements from the parent's bag until both bags have the same size. At the end of the second traversal, the bags of all children are at least as large as the bags of their parents, but they are not contained in the bag of their parent. In the third traversal, each node i with $c \geq 2$ children and bag B_i is replaced by a binary tree with exactly c leaves whose nodes are all assigned the bag B_i. Each child of i in the original tree becomes the single child of one of the leaves of this binary tree. At this point, all nodes have at most two children and those with two children are Join nodes. In the fourth traversal, for any node i with a single child j, if necessary, replace the edge $\{i,j\}$ by a path such that the bags of the nodes along the path differ by exactly one vertex in each step. This is done from j to i by a sequence of nodes, starting with a Forget node, then alternating between Introduce and Forget nodes, and possibly ending with a sequence of Forget nodes in case the child's bag is larger than its parent's. To ensure that property (3) of a tree decomposition is satisfied, each vertex in the symmetric difference of the original bags B_i and B_j produces a single Forget or Introduce node.

To finish the construction, if the root i has a nonempty bag B_i, we append a path to the root where each node is a forget node, until we get an empty bag, which becomes the new root. Note that at most $k+1$ nodes are appended to the path, as $|B_i| \leq k+1$. We call this nice tree decomposition $\mathcal{T}'$. We claim that $\mathcal{T}'$ is a nice tree decomposition of G with the same width as $\mathcal{T}$. It is easy to see that properties (1) and (3) in the definition of tree decomposition are not violated after each traversal, and that all nodes in $\mathcal{T}$ have one of the types in the definition of nice tree decomposition. Moreover, bag sizes have only been increased in the second traversal, but the size of a new bag was always bounded by the size of a bag already in the tree. Finally, we see that $\mathcal{T}'$ is a decomposition of G, in the sense that property (2) in the definition of tree decomposition is satisfied. To this end, if two vertices $u, v \in V(G)$ lie a bag of $\mathcal{T}$, note that one of

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

the bags of the new tree must contain the largest bag originally containing u and v, and therefore it is a tree decomposition of G.

Next, we count the number m' of nodes of $\mathcal{T}'$. The Leaf nodes have degree 1, the Join nodes have degree 3 (unless one of the Join nodes is the root and has degree 2) and Forget and Introduce nodes have degree 2 (unless one of the Forget nodes is the root and has degree 1). Let m'_F, m'_I, m'_J and m'_L denote the number of Forget, Introduce, Join and Leaf nodes in $\mathcal{T}'$, respectively. Since the sum of degrees is $2m'-2$, we have

$$2(m'_F+m'_I+m'_J+m'_L)-2=3m'_J+2(m'_I+m'_F)+m'_L-1,$$

which leads to $m'_J=m'_L-1$. Recall that every vertex is forgotten exactly once, so $m'_F=n$. To see that $m'_L\leq n$, first note that $m'_L=1$ if there are no join nodes. Otherwise, the first and the fourth traversals ensure that there is at least one Forget node on the path between each Leaf node and the first Join node on its path to the root, so that $m'_L\leq m'_F\leq n$. It follows that $m'_J\leq n-1$. The fourth traversal ensures that the single child of every Introduce node is a Forget node, so that we also have $m'_I\leq m'_F\leq n$. Therefore $\mathcal{T}'$ has at most $4n-1$ nodes.

We omit the part of the proof that deals with the number of operations required for this transformation. It may be found in [30, Lemma 1]. □

3.2. The algorithm. We now describe the algorithm based on a tree decomposition of the input graph. Let A be a symmetric matrix of order n and let $G=(V,E)$ be the underlying graph with vertex set $V=[n]$ associated with it. Let $\mathcal{T}$ be a nice tree decomposition of G with node set $[m]$ and width k. As was the case for trees, the algorithm works bottom-up in the rooted tree $\mathcal{T}$, performing congruence operations at each of them. The result at each node i other than the root is a data structure known as a *box*, which is transmitted to the node's parent. A box is a symmetric matrix N_i of order at most $2(k+1)$ that consists of a pair of matrices $(N_i^{(1)},N_i^{(2)})$ whose rows and columns are labeled by elements of $[n]$. These labels make the connection between the rows and columns of the boxes with rows and columns of the original matrix. We may think of the box as a small part of the matrix where the algorithm is operating[§]. While producing the box, the algorithm may also produce diagonal elements of a matrix congruent to A. These diagonal elements are not part of the box transmitted to the node's parent, but are appended to a global array. At the end of the algorithm, this global array consists of the n diagonal elements of a diagonal matrix D that is congruent to A. Figure 9 shows a high-level description of the algorithm, emphasizing that it proceeds in steps marked by the nodes of the tree decomposition, and that the operations performed at each node depend on its type.

As was the case for trees, the algorithm performs row and column operations according to a pivoting scheme. Here, the pivoting scheme is based on the bottom-up ordering of the tree decomposition $\mathcal{T}$ as follows. The first row in the pivoting scheme is the row associated with the vertex forgotten in the first Forget node in this ordering, the second row is the one associated with the vertex forgotten in the second Forget node, and so on. The intuition is that the algorithm attempts to diagonalize row v at the node where v is forgotten. Consider, for instance, the first Forget node i and let v be the vertex of G that is forgotten at i. Let j be the child of i in the tree. By the structure of a nice tree decomposition, we know that $B_i\subset B_j$ and that $\{v\}=B_j\setminus B_i$. Also, our assumption that no vertices have been forgotten up to this point ensures that any neighbor u of v in G must be an element of B_j, as u and v must both lie in a common bag, and v

[§]However, as we shall see, the entries of the box are not necessarily a submatrix of the matrix that would be obtained if the same row and column operations were to be performed in the entire matrix.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

is forgotten before u. In the algorithm, node i receives information about the entries in the neighborhood of v through the box that was transmitted from j to its parent i[¶]. The box is then updated with information about v. If the diagonal entry d_v corresponding to v is nonzero, the algorithm uses it to annihilate nonzero elements in this row/column, and the diagonal element d_v is appended to the global array mentioned above[‖]. The reason why the algorithm works in this case is that if the same row and column operations had been performed in the entire matrix, then the row and column associated with v in the resulting matrix would be diagonalized with diagonal entry d_v. On the other hand, if $d_v = 0$, instead of dealing with it immediately, the row and column associated with v is added to a "temporary buffer," and information about it remains in the box transmitted to i's parent. It will be diagonalized later in the process of producing new boxes.

```
Input:  a nice tree decomposition $\mathcal{T}$ of width $k$
of the underlying graph $G$ associated with $A$ and the nonzero entries of $A$
Output:  diagonal entries in $D \cong A$

Diagonalize Treewidth(A)
Order the nodes of $\mathcal{T}$ as $1, 2, \ldots, m$ in post order
for $i$ from 1 to $m$ do
    if is-Leaf($i$) then $(N_i^{(1)}, N_i^{(2)})$=LeafBox($B_i$)
    if is-Introduce($i$) then $(N_i^{(1)}, N_i^{(2)})$=IntroBox($N_j$)
    if is-Join($B_i$) then $(N_i^{(1)}, N_i^{(2)})$=JoinBox($B_i$)
    if is-Forget($B_i$) then $(N_i^{(1)}, N_i^{(2)})$=ForgetBox($B_i$)
```

FIGURE 9. *High level description of the algorithm* `Diagonalize Treewidth`*.*

Our objective is to describe each operation in detail. This requires us to give a more detailed description of the boxes produced at each node of the tree. An important ingredient to limit fill-in for vertices that have been added to the temporary buffer because of accidental cancelations is to keep matrices in *row echelon form.* A (not necessarily square) matrix $A = (a_{ij})$ is in row echelon form if

(i) All rows having only zero entries are at the bottom;
(ii) The *pivot* of row i is the element $a_{ij} \neq 0$ with least j, if it exists. If rows $i_1 < i_2$ have pivots in columns j_1 and j_2, respectively, then $j_1 < j_2$.

The following are two examples of matrices in row echelon form:

$$A = \begin{pmatrix} 1 & 6 & 2 & -3 \\ 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & -1 \end{pmatrix},\ B = \begin{pmatrix} 3 & 1 \\ 0 & 4 \\ 0 & 0 \end{pmatrix}.$$

Matrix A has pivots 1, 2, and -1 in the first, second, and third rows, respectively. Matrix B has pivots 3 and 4 in the first and second rows, but there is no pivot in the third row.

[¶]In fact, before the first row is diagonalized, the box contains no actual information about previous operations, but this recorded information will be crucial in later steps.

[‖]If we are interested in the diagonal matrix rather than simply on the diagonal elements, we keep track of the pair (v, d_v).

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

Each box $N_i = (N_i^{(1)}, N_i^{(2)})$ produced by a node i represents a single matrix as follows:

$$N_i = \begin{array}{|c|c|} \hline N_i^{(0)} & N_i^{(1)} \\ \hline N_i^{(1)T} & N_i^{(2)} \\ \hline \end{array}, \tag{3.8}$$

where $N_i^{(0)}$ is a zero matrix of dimension $k_i' \times k_i'$, $N_i^{(2)}$ is a symmetric matrix of dimension $k_i'' \times k_i''$ and $N_i^{(1)}$ is a $k_i' \times k_i''$ matrix in row echelon form such that every row has a pivot. Moreover, $0 \leq k_i' \leq k_i'' = |B_i| \leq k+1$. We allow k_i' or k_i'' to be zero, in which case the corresponding matrices $N_i^{(0)}$, $N_i^{(1)}$ or $N_i^{(2)}$ are empty. Recall that the rows of N_i (and therefore the columns) are labeled by vertices of G. Let $V(N_i)$ denote the set of vertices of G associated with the rows of N_i. Using the terminology of the original papers, we say that the k_i' rows in $N_i^{(0)}$ have *type-i* and the k_i'' rows of $N_i^{(2)}$ have *type-ii*. This means that the rows of $N_i^{(1)}$ have type-i and the columns of $N_i^{(1)}$ have type-ii. This is denoted by the partition $V(N_i) = V_1(N_i) \cup V_2(N_i)$, where $V_1(N_i)$ and $V_2(N_i)$ are the vertices of type-i and type-ii, respectively. It is helpful to be aware that the vertices of type-ii are precisely the vertices in B_i and that the vertices in $V_1(N_i)$ correspond to the rows that have been added to the "temporary buffer" at Forget nodes in the branch of the tree decomposition rooted at i and have yet to be fully diagonalized. We also observe that the rows of type-ii are sorted according to a pre-defined order so that they can be merged quickly.

To conclude this section, we describe each step of the algorithm in detail.

Leaf box. When the node is a leaf corresponding to a bag B_i of size b_i, then we apply procedure `Leaf Box` of Fig. 10. This procedure simply initializes a box N_i where $k' = 0$ and $k'' = b_i$, where $N_i^{(2)}$ is the zero matrix of dimension $k'' \times k''$ whose rows and columns are labeled by the elements of B_i.

```
Input:  leaf node $i$ with bag $B_i$ of size $b_i$
Output:  a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$

Leaf Box($i$)
   Set $N_i^{(1)} = \emptyset$
   $N_i^{(2)}$ is a zero matrix of dimension $b_i \times b_i$ with rows and columns
      labeled by the elements of $B_i$ in increasing order.
```

FIGURE 10. *Procedure* `Leaf Box`.

We remark that it is tempting to immediately put the entry a_{uv} at position u, v of $N_i^{(2)}$. However, it is better to incorporate information about a_{uv} in the step associated with forgetting u or v to keep the Join operation simple.

Introduce box. Assume that node i introduces vertex v and let j be its child, so that we have $B_i = B_j \cup \{v\}$ ($v \notin B_j$). The input of `Introduce Box` is the vertex v that has been introduced and the box N_j transmitted

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

by its child. At step i, the box N_i is produced simply by the addition of a new type-ii row/column labeled by v whose elements are all zero, taking care to insert it in the correct order (in term of the pre-defined order for type-ii vertices). This is described in Fig. 11.

```
Input:  bag $B_i$, vertex $v$ and bag $N_j = (N_j^{(1)}, N_j^{(2)})$ produced by its child
Output:  a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$

Introduce Box($B_i, v, N_j$)
   $N_i^{(1)} = N_j^{(1)}$, $N_i^{(2)} = N_j^{(2)}$
   add zero row and zero column associated with $v$ to $N_i^{(2)}$
      making sure that type-ii rows remain in increasing order
   if $N_i^{(1)}$ is nonempty, then add zero column to $N_i^{(1)}$
      preserving the column order of $N_i^{(2)}$
```

FIGURE 11. *Procedure* `Introduce Box`*.*

Join box. Let i be a node of type join and let N_j and N_ℓ be the boxes induced by the pairs transmitted by its children, where $j < \ell < i$. By the definition of the join operation, we have $V_2(N_j) = V_2(N_\ell) = B_i$. Moreover, a step for proving the correctness of the algorithm is to show that $V_1(N_j) \cap V_1(N_\ell) = \emptyset$. The `Join Box` operation first creates a matrix N_i^* whose rows and columns are labeled by $V_1(N_j) \cup V_1(N_\ell) \cup V_2(N_j)$ with the structure below. Assume that $|V_1(N_j)| = r$, $|V_1(N_\ell)| = s$ and $|B_i| = t$, and define

$$N_i^* = \begin{array}{|c|c|c|}\hline \mathbf{0}_{r\times r} & \mathbf{0}_{r\times s} & N_j^{(1)} \\ \hline \mathbf{0}_{s\times r} & \mathbf{0}_{s\times s} & N_\ell^{(1)} \\ \hline N_j^{(1)T} & N_\ell^{(1)T} & N_i^{*(2)} \\ \hline \end{array}, \tag{3.9}$$

where $N_i^{*(2)} = N_j^{(2)} + N_\ell^{(2)}$. The matrices $N_j^{(1)}$ and $N_\ell^{(1)}$ simply appear on top of each other because we made sure that its columns have the same labeling, which is why we require the labels of type-ii rows to be in a pre-defined order. Let $N_i^{*(0)}$ denote the zero matrix of dimension $(r+s)\times(r+s)$ on the top left corner. Note that the matrix

$$N_i^{*(1)} = \begin{array}{|c|}\hline N_j^{(1)} \\ \hline N_\ell^{(1)} \\ \hline \end{array}$$

is an $(r+s)\times t$ matrix consisting of two matrices in row echelon form on top of each other.

To obtain N_i from N_i^*, we perform row operations on $N_i^{*(1)}$. To preserve congruence, each row operation must be followed by the corresponding column operation in $N_i^{*(1)T}$, but of course we need not actually perform the operation. The goal is to merge the rows labeled by $V_1(N_\ell)$ (the *right rows*) into the matrix $N_j^{(1)}$ labeled by the *left rows* to produce a single matrix in row echelon form. We do this as follows. While there is a pivot α_c of a right row w that lies in the same column c as the pivot β_c of v, where v is a left row or a different right row, we use v to eliminate the pivot of w by performing the operation

$$R_w \leftarrow R_w - \frac{\alpha_c}{\beta_c} R_v. \tag{3.10}$$

After this loop has ended, the matrix $N_i^{(1)}$ is obtained from $N_j^{(1)}$ by simply inserting any right rows that still have a pivot in the correct position, in the sense that the final matrix $N_i^{(1)}$ is in row echelon form. Each

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

row and column keeps its label throughout. If Z_i denotes the set of labels of the right rows that became zero vectors while performing the above calculations, the algorithm appends diagonal elements $(0, v)$ for all $v \in Z_i$ to the global array and does not add their rows to $N_i^{(1)}$. This produces the box

$$N_i = \begin{array}{|c|c|}\hline \mathbf{0}_{k' \times k'} & N_i^{(1)} \\ \hline N_i^{(1)T} & N_i^{(2)} \\ \hline \end{array}, \tag{3.11}$$

where $k' = r+s-|Z_i|$, $k'' = t$ and $N_i^{(1)}$ is a matrix of dimension $k' \times k''$ in row echelon form and $N_i^{(2)} = N_i^{*(2)}$. Since every row of $N_i^{(1)}$ has a pivot, we have $k' \leq k''$. This procedure is described in Fig. 12.

```
Input:  bag $B_i$ and boxes $N_j, N_\ell$ produced by the two children of $i$
Output:  a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$ and a set of diagonal elements

Join Box($B_i, N_j, N_\ell$)
   define $N_i^*$ as in (3.9)
   do row operations on $N_i^{*(1)}$ as in (3.10) to achieve row echelon form
   for each zero row of $N_i^{*(1)}$ (labeled by a vertex $u$), output $(0, u)$
   $N_i^{(1)}$ is obtained from $N_i^{*(1)}$ by removing zero rows and exchanging rows as (3.11)
```

FIGURE 12. *Procedure JoinBox.*

Forget box. The pseudocode for this procedure is depicted in Fig. 13, after we explain how it works. Assume that node i forgets vertex v and let j be its child, so that $B_i = B_j \setminus \{v\}$. This procedure starts with N_j and produces a new matrix N_i where the row associated with v becomes of type-i or is diagonalized. Initially, the matrix is updated to include the entries of the original matrix that involve v and the other vertices in B_j. This is done by defining a new matrix N_i^* from the box N_j, where, for all $u \in B_j$ (including $u = v$), the entries uv and vu are replaced by $N_j^{(2)}[u, v] + a_{uv}$, while the other entries remain unchanged.

We observe that the row corresponding to v is type-ii in the box N_j. Thus, after introducing the entries of M, v is implicitly associated with a row in $N_i^{*(2)}$. For convenience, we exchange rows and columns to look at N_i^* in the following way[**]:

$$N_i^* = \begin{array}{|c|c|c|}\hline d_v & \mathbf{x}_v & \mathbf{y}_v \\ \hline \mathbf{x}_v^T & \mathbf{0}_{k' \times k'} & N_i^{*(1)} \\ \hline \mathbf{y}_v^T & N_i^{*(1)T} & N_i^{*(2)} \\ \hline \end{array}. \tag{3.12}$$

Here, the first row and column represent the row and column in N_i^* associated with v, while $N_i^{*(1)}$ and $N_i^{*(2)}$ determine the entries uw such that u has type-i and $w \in B_i$, and such that $u, w \in B_i$, respectively. In particular $\mathbf{x}_v$ and $\mathbf{y}_v$ are row vectors of size k'_j and $k''_j - 1$, respectively.

Depending on d_v and on the vectors $\mathbf{x}_v$ and $\mathbf{y}_v$, we proceed in different ways.

Case 1: $\mathbf{x}_v$ *is empty or* $\mathbf{x}_v = [0 \cdots 0]$.

[**]This is helpful for visualizing the operations, but this step is not crucial in an implementation of this procedure.

If $\mathbf{y}_v = [0 \cdots 0]$ (or $\mathbf{y}_v$ is empty), the row of v is already diagonalized, we simply add (v, d_v) to D_i and remove the row and column associated with v from N_i^* to produce N_i. We refer to this as Subcase 1(a).

If $\mathbf{y}_v \neq [0 \cdots 0]$, there are again two options. In Subcase 1(b), we assume that $d_v \neq 0$ and we use d_v to eliminate the nonzero entries in y_v and diagonalize the row corresponding to v. For each element $u \in B_i$ such that the entry α_v of y_v associated with u is nonzero, we perform

$$R_u \leftarrow R_u - \frac{\alpha_v}{d_v} R_v, C_u \leftarrow C_u - \frac{\alpha_v}{d_v} C_v. \tag{3.13}$$

When all such entries have been eliminated, we append (v, d_v) to the global array and we let N_i be the remaining matrix. Observe that, in this case, $N_i^{(1)} = N_i^{*(1)}$, only the elements of $N_i^{*(2)}$ may be modified to generate $N_i^{(2)}$.

If $\mathbf{y}_v \neq [0 \cdots 0]$ and $d_v = 0$, we are in Subcase 1(c). The aim is to turn v into a row of type-i. To do this, we need to insert $\mathbf{y}_v$ into the matrix $N_i^{*(1)}$ in a way that the resulting matrix is in row echelon form. Note that this may be done by only adding multiples of rows of $V(N_i^{*(1)})$ to the row associated with v. At each step, if the pivot α_j of the (current) row associated with v is in the same position as the pivot β_j of R_u, the row associated with a vertex u already in $N_i^{*(1)}$, we use R_u to eliminate the pivot of R_v:

$$R_v \leftarrow R_v - \frac{\alpha_j}{\beta_j} R_u, C_v \leftarrow C_v - \frac{\alpha_j}{\beta_j} C_u. \tag{3.14}$$

This is done until the pivot of the row associated with v may not be canceled by pivots of other rows, in which case the row associated with v may be inserted in the matrix (to produce the matrix $N_i^{(1)}$), or until the row associated with v becomes a zero row, in which case $(v, 0)$ is added to D_i and the row and column associated with v are removed from N_i^* to produce N_i.

Case 2: $\mathbf{x}_v$ is nonempty and $\mathbf{x}_v \neq [0 \cdots 0]$.

Let u be the vertex associated with the rightmost nonzero entry of x_v. Let α_u be this entry. We use this element to eliminate all the other nonzero entries in x_v, from right to left. Let w be the vertex associated with an entry $\alpha_w \neq 0$. We perform

$$R_w \leftarrow R_w - \frac{\alpha_w}{\alpha_u} R_u, \quad C_w \leftarrow C_w - \frac{\alpha_w}{\alpha_u} C_u. \tag{3.15}$$

A crucial fact is that the choice of u ensures that, even though these operations modify $N_i^{*(1)}$, the new matrix is still in row echelon form and has the same pivots as $N_i^{*(1)}$. If $d_v \neq 0$, we still use R_u to eliminate this element:

$$R_v \leftarrow R_v - \frac{d_v}{2\alpha_u} R_u, \quad C_v \leftarrow C_v - \frac{d_v}{2\alpha_u} C_u. \tag{3.16}$$

At this point, the only nonzero entries in the $(k'+1) \times (k'+1)$ left upper corner of the matrix obtained after performing these operations are in positions uv and vu (and are equal to α_j). We perform the operations

$$R_u \leftarrow R_u + \frac{1}{2} R_v,\ C_u \leftarrow C_u + \frac{1}{2} C_v,\ R_v \leftarrow R_v - R_u,\ C_v \leftarrow C_v - C_u. \tag{3.17}$$

The relevant entries of the matrix are modified as follows:

$$\begin{pmatrix} 0 & \alpha_u \\ \alpha_u & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \alpha_u \\ \alpha_u & \alpha_u \end{pmatrix} \rightarrow \begin{pmatrix} -\alpha_u & 0 \\ 0 & \alpha_u \end{pmatrix}. \tag{3.18}$$

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

We are now in the position to use the diagonal elements to diagonalize the rows associated with v and u, as was done in Case 1, when $x_v = [0,\ldots,0]$ and $d_v \neq 0$. At the end of the step, we add $(v, -\alpha_u)$ and (u, α_u) to D_i. No pivots in $N_i^{*(1)}$ are modified, except for the pivot of the row associated with u, which is diagonalized during the step.

```
Input:  bag $B_i$, a vertex $v$ and matrix $N_j$ transmitted by the child of $i$
Output:  a box $N_i = (N_i^{(1)}, N_i^{(2)})$ and a set of diagonal elements

Forget Box($B_i, v, N_j$)
    $N_i^{*(1)} = N_j^{(1)}$, $N_i^{*(2)}[u,w] = N_j^{(2)}[u,w]$, for all $u,w \in B_j$ with $v \notin \{u,w\}$
    $N_i^{*(2)}[u,v] = N_i^{*(2)}[v,u] = N_j^{(2)}[u,v] + m_{uv}$, for all $u \in B_j$
    Perform row/column exchange so that $N_i$ has the form of (3.12)
    if $\mathbf{x}_v$ is empty or 0
        then if $\mathbf{y}_v$ is empty or 0 // Subcase 1(a)
            then append $(v, d_v)$ to the global array and remove row $v$ from $N_i$
            else if $d_v \neq 0$// Subcase 1(b)
                then use $d_v$ to diagonalize row/column $v$ as in (3.13)
                    append $(v, d_v)$ to the global array and remove row $v$ from $N_i$
                else // Here $d_v = 0$ // Subcase 1(c)
                    do congruence operations as in (3.14)
                    if a zero row is obtained
                        then append $(v, 0)$ to the global array and remove row $v$ from $N_i$
                        else insert row $v$ into $N_i^{(1)}$
        else // Here $\mathbf{x}_v \neq 0$ // Case 2
            use congruence operations as in (3.15)-(3.17) to diagonalize rows/columns
            $u$ and $v$ append $(v, d_v), (u, d_u)$ to the global array and remove rows $v, u$ from $N_i$.
```

FIGURE 13. *Procedure* `Forget Box`.

Figure 13 summarizes the procedure. This concludes the description of the algorithm. In the next subsection, we illustrate its performance with an example. Before this, we observe that, as was the case for the algorithm for trees (see Remark 2.2), the algorithm of Fig. 9 may be modified to produce a diagonal matrix that is $*$-congruent to a Hermitian matrix whose underlying graph has a nice tree decomposition $\mathcal{T}$ of width k. This may be done by keeping all the row operations $R_i \leftarrow R_i - \alpha R_j$ as they currently are, but replacing the corresponding column operation by $C_i \leftarrow C_i - \overline{\alpha} C_j$. We omit the details.

To conclude this section, we state a result proved in [30], which establishes the correctness and the complexity of algorithm `Diagonalize Treewidth`.

THEOREM 3.2. *Given a symmetric matrix M of order n and a tree decomposition $\mathcal{T}$ of width k for the underlying graph of M, algorithm* `Diagonalize Treewidth` *produces a diagonal matrix D congruent to M in time $O(k|\mathcal{T}| + k^2 n)$.*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

3.3. Example. We finish this section with an example that illustrates how the algorithm `Diagonalize Treewidth` is applied. Consider the matrix

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 6 & 0 & -2 & 0 \\ 0 & 0 & 6 & -5 & -3 & 2 & 0 \\ 0 & 0 & 0 & -3 & -4 & -1 & 2 \\ 0 & 0 & -2 & 2 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 1 & -1 \end{pmatrix}. \tag{3.19}$$

The underlying graph of M is the graph G of Fig. 7. We wish to apply the algorithm of Fig. 9 to find a diagonal matrix D that is congruent to M. The nice tree decomposition $\mathcal{T}$ of G given in Fig. 8 is part of the input. For the reader's convenience, $\mathcal{T}$ is depicted in Fig. 14. Its caption explains a way in which the nodes may ordered to comply with the first step of the algorithm. Node $i = 1$ is a leaf with bag $B_1 = \{1, 2, 3\}$.

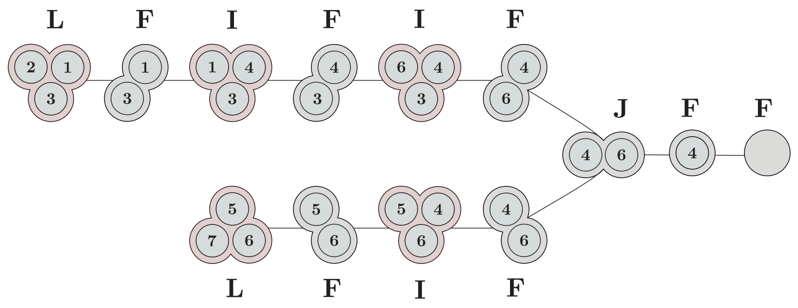
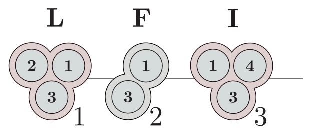


FIGURE 14. *A nice tree decomposition of the graph G of Fig. 7. The root is on the right, and each node is labeled according to its type. The nodes are ordered in post order as follows: nodes 1 to 6 are on the top branch, from left to right. Nodes 7 to 10 are at the bottom branch, also from left to right. Nodes 11 to 13 go from the join node to the root.*

It simply produces a zero matrix $\mathbf{0}_{3\times 3}$ whose rows are labeled by the vertices in B_1. Node $i = 2$ forgets vertex $v = 2$. To apply `ForgetBox`, the first step is to update the entries of the box transmitted by node 1 to produce

$$N_2^* = \begin{array}{c|c:cc} & 2 & 1 & 3 \\ \hline 2 & -1 & 1 & 1 \\ \hdashline 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 \end{array}$$



Note that the vertices corresponding to each row and column are indicated as labels. Moreover, rows and columns have been reordered so that the row corresponding to the vertex being forgotten at this step appears first for better visualization. We are in the case where $\mathbf{x}_v$ is empty and $\mathbf{y}_v = (1, 1)$ is nonempty. Since $d_v = -1 \neq 0$, we are in Subcase 1(b), and the algorithm performs the operations $R_1 \longleftarrow R_1 + R_2$[*], $C_1 \longleftarrow C_1 + C_2$, $R_3 \longleftarrow R_3 + R_2$, $C_3 \longleftarrow C_3 + C_2$ to diagonalize the row and column associated with $v = 2$. After these operations, the matrix N_2^* becomes

$$N_2^* = \begin{array}{c|c:cc} & 2 & 1 & 3 \\ \hline 2 & -1 & 0 & 0 \\ \hdashline 1 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \end{array}.$$

[*]As before, we use the notation R_v and C_v to refer to the row and column labeled by vertex v, respectively.

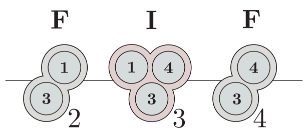
Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

This step ends with the pair $(v, d_v) = (2, -1)$ appended to the global array and a box N_2 such that $N_2^{(1)}$ is empty and $N_2^{(2)} = N_2$:

$$N_2 = \begin{array}{c|cc} & 1 & 3 \\ \hline 1 & 1 & 1 \\ 3 & 1 & 1 \end{array}.$$

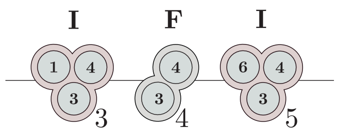
Node $i = 3$ introduces vertex $v = 4$ and produces the following box:

$$N_3 = \begin{array}{c|ccc} & 1 & 3 & 4 \\ \hline 1 & 1 & 1 & 0 \\ 3 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 \end{array}.$$



Node $i = 4$ forgets vertex $v = 1$, and starts with N_4^* given by

$$N_4^* = \begin{array}{c|ccc} & 1 & 3 & 4 \\ \hline 1 & 2 & 2 & 0 \\ 3 & 2 & 1 & 0 \\ 4 & 0 & 0 & 0 \end{array}$$

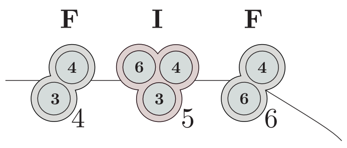


Since $\mathbf{x}_v$ is empty, $\mathbf{y}_v = (2, 0)$ and $d_v = 2 \neq 0$, the algorithm performs the operations $R_3 \longleftarrow R_3 - R_1$, $C_3 \longleftarrow C_3 - C_1$ to diagonalize the row and column associated with $v = 1$. The pair $(1, 2)$ is appended to the global array, and the following matrix N_4 is transmitted to its parent:

$$N_4 = \begin{array}{c|cc} & 3 & 4 \\ \hline 3 & -1 & 0 \\ 4 & 0 & 0 \end{array}.$$

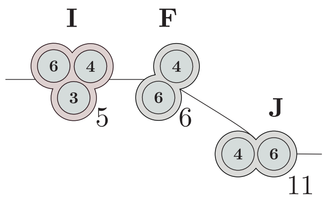
As before, all rows have type-ii. Node 5 introduces vertex $v = 6$, producing the box

$$N_5 = \begin{array}{c|ccc} & 3 & 4 & 6 \\ \hline 3 & -1 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 \end{array}.$$



Node $i = 6$ forgets vertex $v = 3$, and starts with the box

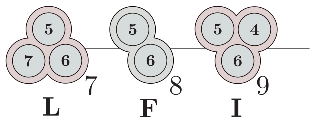
$$N_6^* = \begin{array}{c|ccc} & 3 & 4 & 6 \\ \hline 3 & 0 & 6 & -2 \\ 4 & 6 & 0 & 0 \\ 6 & -2 & 0 & 0 \end{array}.$$



Here $d_v = 0$, $\mathbf{x}_v$ is empty and $\mathbf{y}_v = (6, -2)$. The algorithm is in Subcase 1(c), and the row corresponding to $v = 3$ becomes a type-i row. Since there are no other type-i rows that could require row/column operations to maintain $N_6^{(1)}$ in row echelon form, the box transmitted by node $i = 6$ is $N_6 = N_6^*$, where $v = 3$ has type-i, the other rows have type-ii, $N_6^{(1)} = [6 \;\; -2]$ and $N_6^{(2)} = \mathbf{0}_{2\times 2}$.

Node $i = 7$ is a leaf node starting a new branch of the tree, and node $i = 8$ forgets vertex $v = 7$. After updating the matrix at the beginning of the step, we get

$$N_8^* = \begin{array}{c|ccc} & 7 & 5 & 6 \\ \hline 7 & -1 & 2 & 1 \\ 5 & 2 & 0 & 0 \\ 6 & 1 & 0 & 0 \end{array}.$$



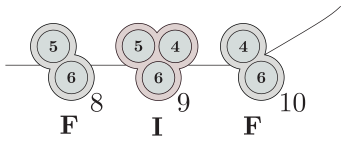
Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

Since $d_v = -1 \neq 0$ and $\mathbf{x}_v$ is empty, the algorithm diagonalizes the row and column associated with vertex $v = 7$. The operations $R_5 \longleftarrow R_5 + 2R_7$, $C_5 \longleftarrow C_5 + 2C_7$, $R_6 \longleftarrow R_6 + R_7$, and $C_6 \longleftarrow C_6 + C_7$ produce the pair $(7, -1)$, which is appended to the global array, and the following box:

$$N_8 = \begin{array}{c|cc} & 5 & 6 \\ \hline 5 & 4 & 1 \\ 6 & 1 & 2 \end{array}.$$

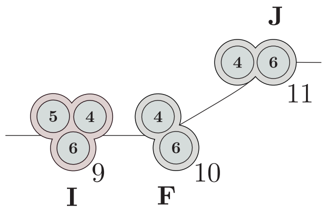
Node $i = 9$ introduces vertex $v = 4$ to produce the box

$$N_9 = \begin{array}{c|ccc} & 4 & 5 & 6 \\ \hline 4 & 0 & 0 & 0 \\ 5 & 0 & 4 & 2 \\ 6 & 0 & 2 & 1 \end{array}.$$



Node $i = 10$ forgets vertex $v = 5$. The step starts with

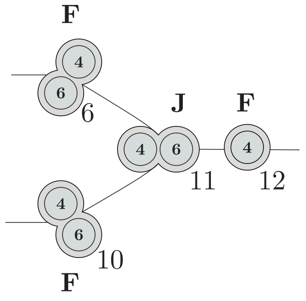
$$N_{10}^* = \begin{array}{c|c:cc} & 5 & 4 & 6 \\ \hline 5 & 0 & -3 & 1 \\ \hdashline 4 & -3 & 0 & 0 \\ 6 & 1 & 0 & 1 \end{array}.$$



We have $d_v = 0$, $\mathbf{x}_v$ empty and $\mathbf{y}_v = (-3, 1)$. The algorithm is in Subcase 1(c), and the row corresponding to $v = 5$ becomes a type-i row. Since there are no other type-i rows that could require row/column operations to maintain $N_{10}^{(1)}$ in row echelon form, the box transmitted by node $i = 10$ is $N_{10} = N_{10}^*$, where $v = 5$ has type-i, the other rows have type-ii, $N_{10}^{(1)} = [-3\ 1]$ and $N_{10}^{(2)} = \mathbf{0}_{2\times 2}$.

Node $i = 11$ is a join node. We start with a matrix N_{11}^* that merges the two boxes transmitted by its children, namely with

$$N_{11}^* = \begin{array}{c|c:c:cc} & 3 & 5 & 4 & 6 \\ \hline 3 & 0 & 0 & 6 & -2 \\ \hdashline 5 & 0 & 0 & -3 & 1 \\ \hdashline 4 & 6 & -3 & 0 & 0 \\ 6 & -2 & 1 & 0 & 1 \end{array}.$$



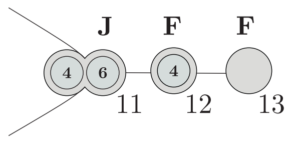
Note that vertices 3 and 5 are type-i vertices coming from the two boxes. To produce a matrix $N_{11}^{(1)}$ in row echelon form, we perform the congruence operations $R_5 \longleftarrow R_5 + {}^1\!/\!_2\ R_3$ and $C_5 \longleftarrow C_5 + {}^1\!/\!_2\ C_3$, so that N_{11}^* becomes

$$\begin{array}{c|c:c:cc} & 3 & 5 & 4 & 6 \\ \hline 3 & 0 & 0 & 6 & -2 \\ \hdashline 5 & 0 & 0 & 0 & 0 \\ \hdashline 4 & 6 & 0 & 0 & 0 \\ 6 & -2 & 0 & 0 & 1 \end{array}, \text{ so that } (5,0) \text{ is appended to the global array and } N_{11} = \begin{array}{c|c:cc} & 3 & 4 & 6 \\ \hline 3 & 0 & 6 & -2 \\ \hdashline 4 & 6 & 0 & 0 \\ 6 & -2 & 0 & 1 \end{array}.$$

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

Next the algorithm processes node $i = 12$, which forgets vertex $v = 6$. It first updates the matrix transmitted by its child to

$$N_{12}^* = \begin{array}{c|ccc} & 6 & 3 & 4 \\ \hline 6 & 2 & -2 & 2 \\ 3 & -2 & 0 & 6 \\ 4 & 2 & 6 & 0 \end{array}.$$



Now we have $d_v = 2 \neq 0$, $\mathbf{x}_v = [-2]$ and $\mathbf{y}_v = [2]$. ForgetBox is in Case 2. Since $\mathbf{x}_v$ has a single element, we use it to eliminate d_v by performing $R_6 \longleftarrow R_6 + {}^1\!/_2\, R_3$ and $C_6 \longleftarrow C_6 + {}^1\!/_2\, C_3$, leading to

$$\begin{array}{c|ccc} & 6 & 3 & 4 \\ \hline 6 & 0 & -2 & 5 \\ 3 & -2 & 0 & 6 \\ 4 & 5 & 6 & 0 \end{array}.$$

Next the algorithm performs operations to replace the nonzero entries in positions $(6, 3)$ and $(3, 6)$ by nonzero elements in the diagonal by performing $R_3 \longleftarrow R_3 + {}^1\!/_2\, R_6$ and $C_3 \longleftarrow C_3 + {}^1\!/_2\, C_6$, followed by $R_6 \longleftarrow R_6 - R_3$ and $C_6 \longleftarrow C_6 - C_3$. This first produces

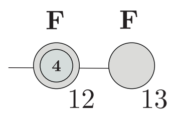
$$\begin{array}{c|ccc} & 6 & 3 & 4 \\ \hline 6 & 0 & -2 & 5 \\ 3 & -2 & -2 & 17/2 \\ 4 & 5 & 17/2 & 0 \end{array}, \text{ followed by } \begin{array}{c|ccc} & 6 & 3 & 4 \\ \hline 6 & 2 & 0 & -7/2 \\ 3 & 0 & -2 & 17/2 \\ 4 & -7/2 & 17/2 & 0 \end{array}.$$

Finally, the algorithm uses the new pivots to diagonalize the rows and columns associated with vertices 3 and 6 by performing $R_4 \longleftarrow R_4 + {}^7\!/_4\, R_6$ and $C_4 \longleftarrow C_4 + {}^7\!/_4\, C_6$, followed by $R_4 \longleftarrow R_4 + {}^{17}\!/_4\, R_3$ and $C_4 \longleftarrow C_4 + {}^{17}\!/_4\, C_3$. We get

$$\begin{array}{c|ccc} & 6 & 3 & 4 \\ \hline 6 & 2 & 0 & 0 \\ 3 & 0 & -2 & 0 \\ 4 & 0 & 0 & 30 \end{array}, \text{ so that } (6, 2) \text{ and } (3, -2) \text{ are appended to the global array and } N_{12} = \begin{array}{c|c} & 4 \\ \hline 4 & 30 \end{array}.$$

The final node is $i = 13$, which forgets vertex $v = 4$. Processing it starts with

$$N_{13}^* = \begin{array}{c|c} & 4 \\ \hline 4 & 25 \end{array}$$



The algorithm appends $(4, 25)$ to the global array and terminates. It has produced a diagonal matrix with diagonal entries $(2, -1, -2, 25, 0, 2, -1)$. In particular, the input matrix M has three positive eigenvalues, three negative eigenvalues, and one eigenvalue equal to 0.

Part II - Dense graphs

In the second part of our survey, we present algorithms that take advantage of a second structural property that may be exploited to compute a diagonal matrix D that is congruent to an input matrix A in linear time. The approach here applies to matrices such that the underlying graph of the input matrix is not necessarily sparse. In particular, the underlying graph could even be the complete graph. The trick is to find pairs of rows (R_i, R_j) that are very similar, in the sense that the vector $R_i - R_j$ has a very small

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

number of nonzero components, so that a single row/column operation eliminates a large number of nonzero off-diagonal elements. In fact, the algorithms will exploit the fact that we know how to choose rows with this property in a way that there is no need to actually operate on most of the components of $R_i - R_j$, it suffices to replace them by 0.

As was the case in the Part I, we shall consider matrices whose underlying graph belongs to a graph class. However, differently from sparse graphs, the strategy presented here cannot be extended to arbitrary symmetric matrices with such an underlying graph. It applies to matrices such that any off-diagonal component lie in $\{0, z\}$ for some real number z. This also means that this approach does not apply to general Hermitian matrices. It is worth noticing that this class of matrices include most matrices dealt in the spectral graph theory (such as the adjacency, the Laplacian, etc).

This part consists of two sections. Section 4 considers matrices whose underlying graphs are cographs, which are graphs with the property that we may always find rows R_i and R_j that are *siblings*, in the sense that they are identical except for the entries corresponding to the elements i and j. This makes them particularly suitable for the approach described in the first paragraph. Section 5 generalizes the algorithm of Section 4 using a hierarchical decomposition closely related to the *clique decomposition* of Courcelle and Olariu [24]. As was the case for the tree decompositions of Part I, this decomposition has a *width* associated with it. To get an intuition, we may think that if there is a decomposition of width k, we are always able to find rows R_i and R_j that are identical except for at most $k+1$ entries. This allows us to extend the approach used for cographs to a situation where computations happen on small boxes.

4. Locating eigenvalues in cographs.

We describe here an algorithm that locates eigenvalues of symmetric matrices whose underlying graph is a cograph, and whose nonzero off-diagonal entries are all equal to some real value $z \neq 0$. The original paper, due to Jacobs, Tura and Trevisan [41], is concerned with the adjacency matrix of a cograph.

4.1. Cographs and cotrees.

The class of complement reducible graphs, known as *cographs*, is the hereditary class $\mathcal{F}orb(P_4)$ of all P_4-free graphs, that is, the class of graphs that do not contain P_4 as an induced subgraph. Cographs have appeared in various contexts and have been redefined in many different ways. A more detailed account may be found in Corneil, Lerchs, and Burlingham [22]. One of the many equivalent definitions is constructive and uses two graph operations: disjoint union and join. The *disjoint union* of two graphs $G = (V, E)$ and $H = (W, F)$, where $V \cap W = \emptyset$, is the graph $G \cup H$ with vertex set $V \cup W$ an edge set $E \cup F$. The *join* $G \oplus H$ is the graph with vertex set $V \cup W$ and edge set $E \cup F \cup \{\{v, w\} : v \in V, w \in W\}$.

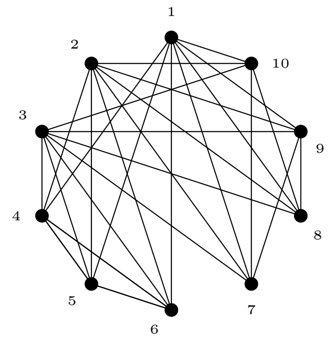
Consider the following graph class $\mathcal{C}$:

(i) $K_1 \in \mathcal{C}$.
(ii) If G_1 and G_2 are vertex-disjoint graphs in $\mathcal{C}$, then $G_1 \cup G_2 \in \mathcal{C}$.
(iii) If G_1 and G_2 lie in $\mathcal{C}$, then $G_1 \oplus G_2 \in \mathcal{C}$.

It is well known that $\mathcal{F}orb(P_4) = \mathcal{C}$.

Using the recursive construction that produces graphs in $\mathcal{C}$, we may represent every n-vertex cograph with vertex set V as a rooted tree, known as *cotree*, whose *nodes* consist of n leaves corresponding to the elements of V and of internal nodes that carry either the label "$\cup$" for union or "$\oplus$" for join. Given such a tree T, we can easily construct the corresponding cograph G_T. The construction can be made unique as

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.



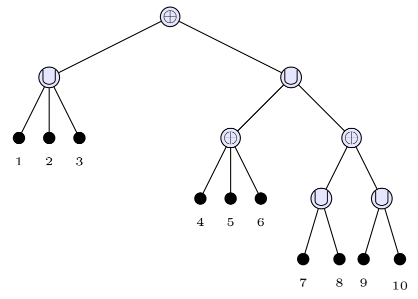


Figure 15. *A cograph with 10 vertices and its minimal cotree.*

follows. We say that a cotree is in *normalized form*[†] if every internal node has at least two children and has a label that differs from the label of its parent. In other words, the children of nodes labeled $\cup$ are leaves or nodes labeled $\oplus$, while the children of nodes labeled $\oplus$ are leaves or nodes labeled $\cup$. As an example, the cograph on the left of Fig. 15 is represented by the minimal cotree on the right. In this note, we assume that the cotree representation of a cograph is always in normalized form. We refer to [34, Theorem 4.3] for a proof that every cograph has a unique minimal cotree. Unlike the tree decomposition of Section 3, a cotree (and, particularly, the minimal cotree) of a cograph may be computed efficiently, we refer to Corneil, Perl and Stewart [23] for a linear-time algorithm with this purpose.

Two vertices u and v in a graph G are called *siblings* if they are either *duplicates* or *coduplicates*, that is if they share the same (open) neighborhood $N(u) = N(v)$ or if they share the same closed neighborhood $N[u] = N[v]$, respectively. The following is well known, a proof may be found in [34, Lemma 4.2].

Lemma 4.1. *Two vertices v and u in a cograph are siblings if and only if they share the same parent node x in the normalized cotree. Moreover, if $x = \cup$, they are duplicates. If $x = \oplus$, they are coduplicates.*

One of the consequences of this lemma is that any cograph on $n \geq 2$ vertices contains siblings.

Since the class of cographs is hereditary, if $G = (V, E)$ is a cograph and $v \in V$, then $G - v$ is also a cograph. Depending on the way v is chosen, the normalized cotree associated with $G - v$ can be easily derived from T_G.

Lemma 4.2. *Let T_G be the normalized cotree of a cograph $G = (V, E)$ with $|V| \geq 2$, and let $v, u \in V$ be siblings of greatest depth in T_G and let w be their parent. Assume that w has $k \geq 2$ children. The following hold for the normalized cotree T_{G-v} of $G - v$:*

(a) *If $k > 2$, T_{G-v} is obtained by deleting the leaf node corresponding to v from T_G.*
(b) *If $k = 2$ and w is not the root of T_G, T_{G-v} is obtained by deleting the leaf nodes corresponding to v and w from T_G and by adding an edge between the node corresponding to u and the parent of w in T_G.*
(c) *If $k = 2$ and w is the root, T_{G-v} whose root is the node corresponding to u.*

[†]A cotree that is in normalized form is often known as a *minimal cotree*.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

4.2. The algorithm. We assume that a real symmetric matrix $A = [a_{ij}]$ of order n is given and that its underlying graph is a cograph G. The nonzero off-diagonal elements, which may be viewed as edge weights, are all equal to some real value $z > 0$, but the diagonal elements, which may be viewed as vertex weights, are arbitrary. Further assume that the algorithm is given the normalized cotree representation T_G of G. As was the case for the algorithms of the previous sections, the algorithm runs in linear time and is based on congruence operations, namely elementary row operations, followed by the same elementary column operation. It works bottom-up on T_G, and at each step it identifies a pair of siblings and diagonalizes the rows and columns of at least one of them. To conclude the step, it updates the cotree for it to generate the subgraph obtained by deleting the diagonalized vertex (or vertices).

We start our description by a sample step of the algorithm. Let $\{v_k, v_\ell\}$ be a pair of siblings in G. Lemma 4.1 tells us they have the same parent w. We assume the diagonal values d_k and d_ℓ of rows R_k and R_ℓ, respectively, may have been modified in the previous computations, but that all off-diagonal entries in these rows and columns have the same value as in the input matrix. In particular, all off-diagonal entries are either 0 or z. The goal is to annihilate off-diagonals in the row and column corresponding to v_k, maintaining congruence to A. Let w be the parent of the siblings in T_G. There are two cases.

Case 1: $w = \oplus$**.** Since $\{v_k, v_\ell\}$ are coduplicates, rows (columns) ℓ and k have the same entries, except possibly in positions ℓ and k, where the diagonal elements d_k, d_ℓ appear. Moreover, $a_{k\ell} = a_{\ell k} = z$. The representation of the rows and columns in the matrix is given below:

$$\begin{pmatrix}
 & & & & a_1 & a_1 & & \\
 & & & & \vdots & \vdots & & \\
 & & & & a_i & a_i & & \\
 & & & & \vdots & \vdots & & \\
a_1 & \dots & a_i & \dots & d_\ell & z & \dots & a_n \\
a_1 & \dots & a_i & \dots & z & d_k & \dots & a_n \\
 & & & & \vdots & \vdots & & \\
 & & & & a_n & a_n & &
\end{pmatrix}.$$

Here, $a_i \in \mathbb{R}$. As the rows R_k and R_ℓ have many equal entries, we execute the following operations.

$$R_k \leftarrow R_k - R_\ell, \qquad C_k \leftarrow C_k - C_\ell.$$

The effect on rows k and ℓ is recorded below. Since the matrix is symmetric, there is no need to represent the columns.

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix}
a_1 & \dots & a_i & \dots & d_\ell & z - d_\ell & \dots & a_n \\
\cdot & & \cdot & & \cdot & & \cdot & \cdot \\
0 & \dots & 0 & \dots & z - d_\ell & d_k + d_\ell - 2z & \dots & 0
\end{pmatrix},$$

After this, most of the nonzero elements in row and column k have been removed. The next step is to diagonalize row k, and to do this, we must eliminate the two entries $z - d_\ell$. There are three subcases, depending on whether $d_\ell = z$ and on whether $d_k + d_\ell = 2z$.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

subcase 1a: $d_k + d_\ell \neq 2z$. Then, we may perform the operations

$$R_\ell \leftarrow R_\ell - \frac{z-d_\ell}{d_k+d_\ell-2z}R_k, \qquad C_\ell \leftarrow C_\ell - \frac{z-d_\ell}{d_k+d_\ell-2z}C_k$$

obtaining:

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} a_1 & \dots & a_i & \dots & \gamma & \dots & 0 & \dots & a_n \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & 0 & \dots & d_k+d_\ell-2z & \dots & 0 \end{pmatrix},$$

where

$$\gamma = d_\ell - \frac{(z-d_\ell)^2}{d_k+d_\ell-2z} = \frac{d_k d_\ell - z^2}{d_k+d_\ell-2z}.$$

The following assignments are made by the algorithm, reflecting the net result of these operations.

$$d_k \leftarrow d_k + d_\ell - 2z, \qquad d_\ell \leftarrow \frac{d_k d_\ell - z^2}{d_k+d_\ell-2z}. \tag{4.20}$$

As row k has been diagonalized, the value d_k becomes permanent and we may remove v_k from the cograph, and hence of the cotree (see Lemma 4.2):

$$T_G \leftarrow T_G - v_k. \tag{4.21}$$

subcase 1b: $d_k + d_\ell = 2z$ and $d_\ell = z$. In this case, rows k and ℓ of the matrix look like

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} a_1 & \dots & a_i & \dots & z & \dots & 0 & \dots & a_n \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & 0 & \dots & 0 & \dots & 0 \end{pmatrix},$$

and we are done. We make the assignments

$$d_k \leftarrow 0, \qquad d_\ell \leftarrow z, \qquad T_G \leftarrow T_G - v_k,$$

as d_k becomes permanent, and v_k is removed.

subcase 1c: $d_k + d_j = 2z$ and $d_\ell \neq z$. Then, our matrix looks like

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} a_1 & \dots & a_i & \dots & d_\ell & \dots & z-d_\ell & \dots & a_n \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & z-d_\ell & \dots & 0 & \dots & 0 \end{pmatrix}.$$

Since $z - d_\ell \neq 0$ is the only nonzero element on row R_k, we may use it to eliminate any element of the form $a_{i\ell}$ for $i \notin \{k, \ell\}$ such that $a_{i\ell} \neq 0$ (and hence $a_{i\ell} = z$) by performing

$$R_i \leftarrow R_i - \tfrac{z}{z-d_\ell}R_k, \quad C_i \leftarrow C_i - \tfrac{z}{z-d_\ell}C_k \ . \tag{4.22}$$

This annihilates most entries in row (and column) ℓ, without changing any other values, yielding

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} 0 & \dots & 0 & \dots & d_\ell & \dots & z-d_\ell & \dots & 0 \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & z-d_\ell & \dots & 0 & \dots & 0 \end{pmatrix}.$$

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

The operations

$$R_\ell \leftarrow R_\ell + \tfrac{1}{2}R_k, \quad C_\ell \leftarrow C_\ell + \tfrac{1}{2}C_k,$$

replace the diagonal d_ℓ with the value z, while the operations

$$R_k \leftarrow R_k - \tfrac{(z-d_\ell)}{z}R_\ell; \quad C_k \leftarrow C_k - \tfrac{(z-d_\ell)}{z}C_\ell,$$

eliminate the two off-diagonal elements, finally giving

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} 0 & \dots & 0 & \dots & z & \dots & 0 & \dots & 0 \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & 0 & \dots & -\frac{(z-d_\ell)^2}{z} & \dots & 0 \end{pmatrix}.$$

What is different about this subcase is that both rows k and ℓ have been diagonalized. We make the following assignments

$$d_k \leftarrow -\frac{(z-d_\ell)^2}{z}, \qquad d_\ell \leftarrow z, \qquad T_G \leftarrow T_G - v_k, \qquad T_G \leftarrow T_G - v_\ell,$$

removing both vertices from T_G and making both variables permanent.

Case 2: $w = \cup$**.** As v_ℓ and v_k are duplicates, by Lemma 4.1, rows R_k and R_ℓ of A have the same values except possibly in positions ℓ and k, where the diagonal elements d_k, d_ℓ appear. In this case, $a_{\ell k} = a_{k\ell} = 0$.

Similarly to Case 1, the row and column operations

$$R_k \leftarrow R_k - R_\ell, \quad C_k \leftarrow C_k - C_\ell,$$

yield the following transformation:

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} a_1 & \dots & a_i & \dots & d_\ell & 0 & \dots & a_n \\ \cdot & & \cdot & & \cdot & \cdot & & \cdot \\ a_1 & \dots & a_i & \dots & 0 & d_k & \dots & a_n \end{pmatrix} \longrightarrow \begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} a_1 & \dots & a_i & \dots & d_\ell & -d_\ell & \dots & a_n \\ \cdot & & \cdot & & \cdot & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & -d_\ell & d_k + d_\ell & \dots & 0 \end{pmatrix}.$$

subcase 2a: $d_k + d_\ell \neq 0$**.** The algorithm performs the matrix operations

$$R_\ell \leftarrow R_\ell + \frac{d_\ell}{d_k + d_\ell}R_k, \quad C_\ell \leftarrow C_\ell + \frac{d_\ell}{d_k + d_\ell}C_k,$$

to diagonalize row k. The net result of these operations is that the following assignments are made.

$$d_k \leftarrow d_k + d_\ell, \qquad d_\ell \leftarrow \tfrac{d_k d_\ell}{d_k + d_\ell}, \qquad T_G \leftarrow T_G - v_k.$$

subcase 2b: $d_k + d_\ell = 0$ **and** $d_\ell = 0$**.** As in the case **subcase 1b**, the row and column k of the matrix is in diagonal form and we assign

$$d_k \leftarrow 0, \qquad d_\ell \leftarrow 0, \qquad T_G \leftarrow T_G - v_k.$$

subcase 2c: $d_k + d_\ell = 0$ **and** $d_\ell \neq 0$**.** Since $d_\ell \neq 0$, we use row and column operations similar to (4.22) to annihilate most of entries in row ℓ and column ℓ:

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

$$\begin{array}{c} \ell \\ \cdot \\ k \end{array}\begin{pmatrix} 0 & \dots & 0 & \dots & d_\ell & \dots & -d_\ell & \dots & 0 \\ \cdot & & \cdot & & \cdot & & \cdot & & \cdot \\ 0 & \dots & 0 & \dots & -d_\ell & \dots & 0 & \dots & 0 \end{pmatrix}.$$

The congruence operations

$$R_k \leftarrow R_k + R_\ell, \quad C_k \leftarrow C_k + C_\ell,$$

complete the diagonalization of rows k and ℓ. The following assignments are made.

$$d_k \leftarrow -d_\ell, \qquad d_\ell \leftarrow d_\ell, \qquad T_G \leftarrow T_G - v_k, \qquad T_G \leftarrow T_G - v_\ell.$$

```
Input:  matrix $A$ and minimal cotree $T_G$ associated with its underlying graph $G$
$A$ may be given by its diagonal $(d'_1, d'_2, \dots, d'_n)$, and by its off-diagonal nonzero value $z$
Output:  diagonal matrix $D = \mathtt{diag}(d_1, d_2, \dots, d_n)$ congruent to $A$

 Algorithm Diagonalize Cograph($T_G, x$)
   initialize $d_i := d'_i$, for $1 \le i \le n$
   while $T_G$ has $\ge 2$ leaves
      select siblings $\{v_k, v_\ell\}$ of maximum depth with parent $w$
      $\alpha \leftarrow d_k \quad \beta \leftarrow d_\ell$
      if $w = \oplus$
         if $\alpha + \beta \ne 2z$              //subcase 1a
            $d_k \leftarrow \alpha + \beta - 2z$, $d_\ell \leftarrow \frac{\alpha\beta - z^2}{\alpha+\beta-2z}$, $T_G = T_G - v_k$
         else if $\beta = z$                 //subcase 1b
            $d_k \leftarrow 0$, $d_\ell \leftarrow z$, $T_G = T_G - v_k$
         else                              //subcase 1c
            $d_k \leftarrow -\frac{(z-\beta)^2}{z}$, $d_\ell \leftarrow z$, $T_G = T_G - v_k$, $T_G = T_G - v_\ell$
      else if $w = \cup$
         if $\alpha + \beta \ne 0$               //subcase 2a
            $d_k \leftarrow \alpha + \beta$, $d_\ell \leftarrow \frac{\alpha\beta}{\alpha+\beta}$, $T_G = T_G - v_k$
         else if $\beta = 0$                 //subcase 2b
            $d_k \leftarrow 0$, $d_\ell \leftarrow 0$, $T_G = T_G - v_k$
         else                              //subcase 2c
            $d_k \leftarrow -\beta$, $d_\ell \leftarrow \beta$, $T_G = T_G - v_k$, $T_G = T_G - v_\ell$
   end loop
```

FIGURE 16. *Diagonalizing A.*

Based on the sample step described above, we provide an algorithm that constructs a diagonal matrix D, congruent to a real symmetric matrix A such that all off-diagonal entries lie in a set $\{0, z\}$ for some $z \in \mathbb{R}$ and such that the underlying graph is a cograph G. It takes as input the minimal cotree T_G, initializing all entries of D with the respective diagonal element of A.

At the start of each iteration, we have the cotree of the subgraph H of G induced by the rows that have not yet been diagonalized. A pair of siblings $\{v_k, v_j\}$ of H with maximum depth is selected, and either one or both of the corresponding rows are diagonalized. In the process of doing this, some of the other diagonal values need to be updated. A crucial fact in establishing the correctness of the algorithm is that, after a row

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

is diagonalized, its entries do not participate in later congruence operations, so that we can assume that the corresponding vertex has been deleted from the graph.

Figure 16 is the pseudocode of the algorithm `Diagonalize Cograph`. Note that the cotree and the matrix can be represented with a data structure of size $O(n)$. Since the algorithm does not actually perform the congruence operations described in this section, but rather only records changes on the diagonal values d_i of A, only $O(n)$ space is necessary. The algorithm terminates when all rows and columns have been diagonalized. Each iteration of `Diagonalize Cograph` takes constant time, so its running time is $O(n)$.

EXAMPLE 4.3. *We illustrate the application of the algorithm. For small cographs, the algorithm may be performed by hand, using its cotree. In the remainder of this section, we apply Algorithm* `Diagonalize Cograph` *to the adjacency matrix of the cograph in Fig. 15, so that the diagonal is zero and $z = 1$. In our figures, diagonal values d_i appear under the vertex v_i in the cotree. Initially, all d_i will be $x = 0$. Figure 17 (left) shows this initialization. At the end of the algorithm, the d_i values correspond to the diagonal values of a matrix that is congruent to $A(G)$. Sibling pairs that are being processed appear in red. Dashed edges indicate vertices about to be removed from the cotree. In the first iteration of the algorithm, it chooses siblings $\{v_k, v_\ell\}$ of maximum depth, as shown in red in Fig. 17(center). Since $d_k = \alpha = 0$ and $d_\ell = \beta = 0$,* **subcase 2b** *occurs and the assignments*

$$d_k \leftarrow 0 \qquad d_\ell \leftarrow 0$$

are made. Figure 17(right) depicts the cotree after vertex v_k is removed and v_ℓ is relocated (in green) according to the rules in Lemma 4.2. Additionally, we prepare for the next step, marking siblings to be processed in red.

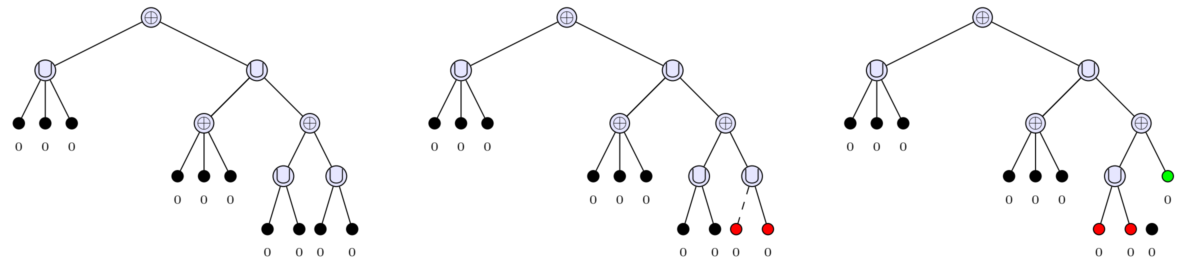


FIGURE 17. *Initialization (left), First iteration (center), Rearranging the cotree (right).*

The two red siblings are processed exactly as the first iteration, so we arrive at the cotree of Fig. 18 (left). We note that after these two steps, two diagonal values equal zero have been found.

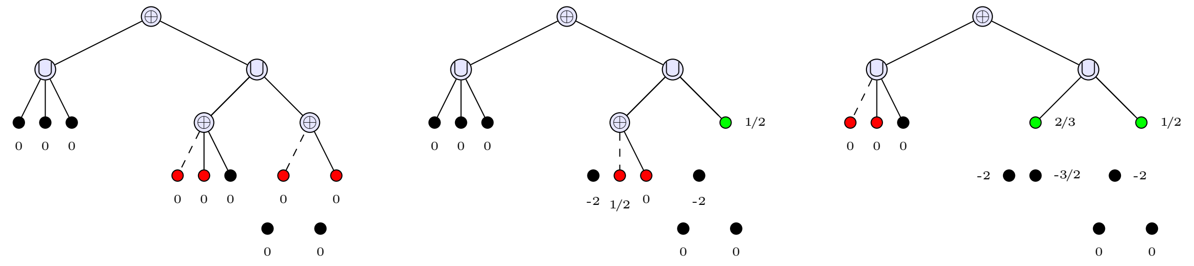


FIGURE 18. *Processing siblings of depth 3.*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

For the next steps, we can choose any of the two pairs of siblings of maximum depth (depth 3), which are shown in red in Fig. 18. Both are of type $\oplus$, so that to process $\{v_k, v_\ell\}$ with $d_k = \alpha = 0$ and $d_\ell = \beta = 0$, **subcase 1a** *is executed and the following assignments are made:*

$$d_k \leftarrow -2 \qquad d_\ell \leftarrow \frac{1}{2}.$$

After these two steps, we arrive at the cotree depicted in Fig. 18 (center), with the vertices v_k removed and one vertex v_ℓ relocated. We notice that one v_ℓ does not move according to the rules in Lemma 4.2. Next, the only remaining sibling pair $\{v_k, v_\ell\}$ of depth 3 is chosen, illustrated in red in Fig. 18 (center). Since $d_k = \alpha = \frac{1}{2}$ and $d_\ell = \beta = 0$, **subcase 1a** *is again executed and the assignments*

$$d_k \leftarrow -\frac{3}{2} \qquad d_\ell \leftarrow \frac{2}{3},$$

and the updated cotree is depicted in Fig. 18 (right). We now prepare for processing the next round of nodes that are of depth 2.

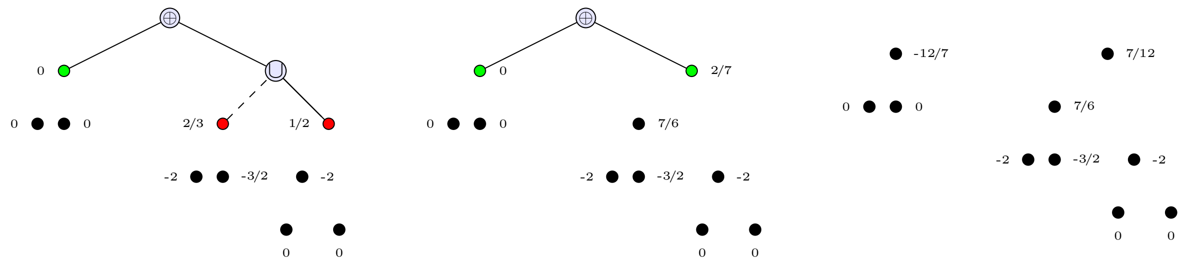


FIGURE 19. *Processing depth two and depth one vertices.*

The leftmost vertices of depth two on the cotree depicted in Fig. 18 (right) have $d_k = \alpha = 0$ and $d_\ell = \beta = 0$, **subcase 2b** *is executed and the assignments $d_k \leftarrow 0 \quad d_\ell \leftarrow 0$ are made, leaving a permanent value $d_k = 0$. We apply again the same step with the same values, leaving a cotree in the shape of Fig. 19 (left). We are then ready to process the red sibling pair with $d_k = \alpha = 2/3$ and $d_\ell = \beta = 1/2$. The assignments $d_k \leftarrow 7/6 \quad d_\ell \leftarrow 2/7$ are made, leaving a permanent value $7/6$. The updated cotree after these operations is in Fig. 19 (center). We finally process the two remaining vertices having $d_k = \alpha = 0$ and $d_\ell = \beta = 2/7$, this is* **subcase 1a**, *and the assignments $d_k \leftarrow -12/7 \quad d_\ell \leftarrow 7/12$.*

The diagonal values are represented in Fig. 19 (right). We note there are 4 negative values, 4 zero values, and 2 positive values, meaning the cograph has 4 negative eigenvalues, 2 positive eigenvalues, and 0 is an eigenvalue of multiplicity 4. For comparison, the actual spectrum of the cograph is the multiset

$$\{1 - \sqrt{22}, -2, -1^2, 0^4, 2, 1 + \sqrt{22}\}.$$

5. Locating eigenvalues using clique-width decomposition. In the previous Section 4, we presented a diagonalization algorithm for symmetric matrices whose underlying graph is a cograph G. Here, we generalize this approach to arbitrary graphs, using a parse tree representation of G in terms of its slick clique decomposition.

5.1. Slick clique-width decomposition. Motivated by the tree decompositions discussed in Section 3, Courcelle and Olariu [24] aimed to define a hierarchical decomposition that would apply to wider

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

classes of graphs, including dense graphs, while also being useful for the design of algorithms. This resulted in a logic-algebraic description of graphs, called a *clique decomposition*, where each graph is described by an algebraic expression that is allowed to use a set of basic labels $[k] = \{1, \dots, k\}$. The *width* of the decomposition is given by the size k of the set of labels used in the decomposition. In this section, we consider a variant of this decomposition, called a *slick clique decomposition*, which satisfies the additional property that subexpressions of the original expression produce induced subgraphs of the original graph. We should mention that a similar variant of the clique decomposition with this property is the NLC-width, introduced by Wanke [59].

Let k be a positive integer. A *slick expression* is an expression formed from *atoms* $i(v)$ and *binary operations* $\oplus_{S,L,R}$, where $L\colon [k] \to [k]$ and $R\colon [k] \to [k]$ are functions, and S is a binary relation on $[k]$. Expressions produce a graph according to the following rules:

(a) $i(v)$ creates a vertex v with label i, where $i \in [k]$[‡].
(b) Given two graphs G and H whose vertices have labels in $[k]$, the labeled graph $G \oplus_{S,L,R} H$ is obtained as follows. Starting with the disjoint union of G and H, add edges from every vertex labeled i in G to every vertex labeled j in H for all $(i,j) \in S$. Afterwards, every label i of the *left component* G is replaced by $L(i)$, and every label j of the *right component* H is replaced by $R(j)$.

The graph constructed by a slick expression is obtained by deleting the labels of the labeled graph produced by it. The *slick clique-width* of a graph G, denoted $scw(G)$, is the smallest k such that the graph can be constructed by a slick expression. Clearly, $scw(G) \leq |V(G)|$.

As an example, consider the graph G in Fig. 20 (left). It may be generated by the slick expression below, which has slick clique-width 2. Here, id denotes the identity map in [2], $f_1\colon 2 \to 1$ maps all elements to 1, $f_2\colon 1 \to 2$ maps all elements to 2, and $S = \{(1,1),(1,2),(2,1)\}$.

$$\Big(\big(1(a) \oplus_{\{(1,2)\},id,id} 2(b)\big) \oplus_{S,f_1,f_2} \big(1(d) \oplus_{\{(1,2)\},id,id} 2(c)\big)\Big) \oplus_{\{(1,1)\},id,2\to 1} 1(e). \tag{5.23}$$

An easy induction shows that a graph G satisfies $scw(G) = 1$ if and only if G is a cograph (see [34, Proposition 4.4], e.g.). Since $G[a,b,c,e]$ is isomorphic to P_4, the graph G in Fig. 20 is not a cograph, and therefore $scw(G) = 2$.

We should mention that the parameters clique-width and slick clique-width are linearly related.

THEOREM 5.1. *If G is a graph then $scw(G) \leq cw(G) \leq 2\, scw(G)$.*

In fact, the proof of Theorem 5.1 [29, Theorem 1] is constructive and may be viewed as two linear-time algorithms for translating a k-expression into an equivalent slick k-expression, and for translating a slick k-expression into an equivalent $2k$-expression. Because of this, any clique decomposition in the literature may be theoretically turned into a slick clique decomposition with the same width. This is convenient because computing the clique-width is NP-complete for arbitrary graphs [25], and we believe that the same must be true for the slick clique-width. Moreover, unlike what happens for the treewidth, there is still no known polynomial algorithm to recognize whether a graph has clique-width bounded by k for any fixed $k > 3$, but there are algorithms for $k \leq 3$ [21]. In terms of approximation algorithms, for fixed k, Oum and Seymour [48] introduced an algorithm that either gives a $(2^{3k+2}-1)$-expression for an input n-vertex graph G, or provides

[‡]In this section, the *label* of a vertex is not its name, but a value assigned to it. Different vertices may be assigned the same label.

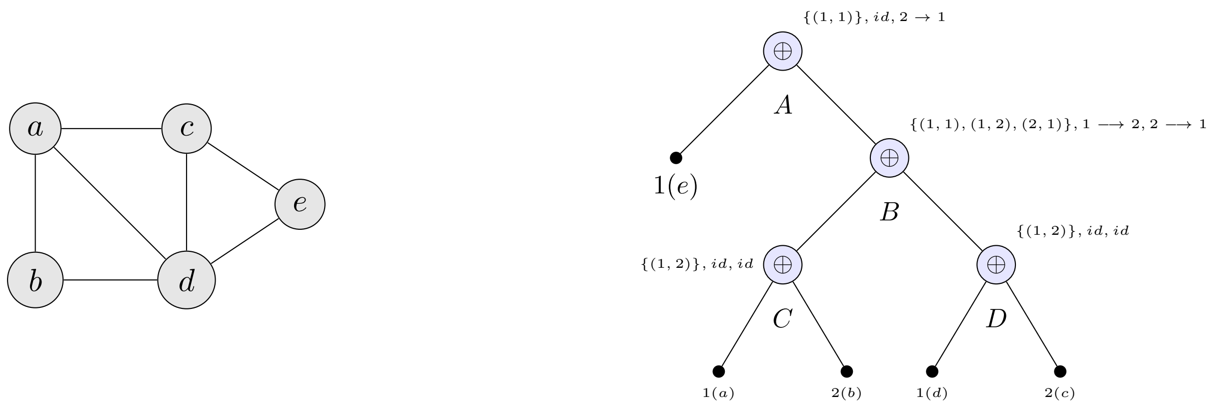


FIGURE 20. *A graph G with $scw(G) = 2$ and its parse tree for the slick expression (5.23).*

a witness that G does not have clique-width $k+1$ in time $O(n^9 \log n)$. Thus, as was the case for the tree decomposition, algorithms based on the (slick) clique-width typically assume that a (slick) decomposition of the graph is part of the input.

As illustrated in Fig. 20, a slick expression for a graph G of order n may be represented as a binary parse tree T having $2n-1$ nodes, where the n leaves are associated with the atoms $i(v)$ and the internal nodes are associated with the binary operations $\oplus_{S,L,R}$. Moreover, the left child corresponds to the root of the left component, while the right child corresponds to the root of the right component. Additional similarities with the cotree representation of cographs are as follows.

The algorithm `Diagonalize Cograph` of the previous section exploited the fact that in any cograph of order $n \geq 2$, there exist two vertices u and v for which either $N(u) = N(v)$ or $N[u] = N[v]$. This means that their corresponding rows and columns in the adjacency matrix can differ by at most two positions. By subtracting say, the row (column) of u from the row (column) of v, the off-diagonal entries of the row and column of v are annihilated except possibly for one such entry. A similar property of slick decompositions that is consistent with our goal of finding vertices whose neighborhoods are very close is that, once two vertices lie in the same component and have the same label, their adjacency relation to any vertex that is added to the graph in a later operation is equal. We formalize this below as a remark, pointing out that this property is crucial to the algorithm.

REMARK 5.2. *Let T_G be a parse tree for a graph $G = (V, E)$ with adjacency matrix A, and let Q be a node in T_G. If two vertices u and v have the same label at Q, then their rows (columns) are the same outside of the matrix for the subtree rooted at Q, that is, if $w \in V$ is not associated with a leaf of this subtree, then $a_{uw} = a_{vw} = a_{wu} = a_{wv}$.*

Another property that is crucial in the cograph algorithm is that subgraphs generated by subexpressions are induced subgraphs. This is exactly the purpose of using the slick clique decomposition, for which this property holds. As pointed out in Theorem 5.1, there are linear-time transformations to translate a k-expression into a slick k-expression, and a slick k-expression into a $2k$-expression. Hence, we may assume that we are given either a clique decomposition or a slick clique decomposition of width k for the graph G with n vertices.

To take advantage of the similarities between cotrees and the parse tree given by the slick clique-width decomposition, the algorithm requires several new developments. The new algorithm, for example, does not

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

diagonalize a given number of vertices at each node of the parse tree. Instead, it transmits information up the tree in a compact way, using a data structure we call *box*, similar to the algorithm of Section 3.

5.2. The algorithm. We present an $O(k^2n)$ time diagonalization algorithm for the adjacency matrix $A = (a_{ij})$, of a graph G having clique-width k. The version stated here is a slight modification of the algorithm in the original paper [29] (a preliminary version is in [28]), which had complexity $O(k^3n)$. As was the case for cographs, the algorithm is easily extendable to any symmetric matrix M whose underlying graph is G and whose nonzero off-diagonal entries are all equal to each other. This includes many widely used graph matrices, such as the adjacency, the Laplacian and the signless Laplacian matrices. We notice that such matrices can have $\Omega(n^2)$ nonzero entries, and that the clique-width may be a small constant even if other parameters, such as the treewidth, are linear in n. In other words, a nice feature of the algorithm of this section is that the underlying graph may be dense.

```
input:  the parse tree $T_G$ of a slick $k$-expression $Q_G$ for $G$, a scalar $x$
output:  diagonal matrix $D = \text{diag}(d_1, \ldots, d_n)$ congruent to $A(G) + xI$

Diagonalize Clique-width(G,x)
   Order the vertices of $T_G$ as $Q_1, Q_2, \ldots, Q_{2n-1} = Q_G$ in post order
   for $t$ from 1 to $2n-1$ do
      if is-leaf($Q_t$)
         then $b_{Q_t}$=LeafBox($Q_t, x$)
         else if $Q_t = Q_\ell \bigoplus_{S,L,R} Q_r$
            then $b_{Q_t}$ = Combine Boxes($b_{Q_\ell}, b_{Q_r}$)
   DiagonalizeBox($b_{Q_{2n-1}}$)
```

FIGURE 21. *High level description of the algorithm* `Diagonalize Clique-width`.

For a given graph $G = (V, E)$ with $n = |V|$ vertices and adjacency matrix A, let Q_G be a slick expression that generates G. Given a real number x, we find a diagonal matrix congruent to $B = A + xI_n$. This may be easily extended to matrices having arbitrary diagonal entries and off-diagonal elements in $\{0, z\}$ for some real number z.

The expression Q_G is associated with a parse tree T_G having $2n - 1$ nodes, as seen above. It is a rooted binary tree where the n leaves are labeled by the operators $i(v)$ and the internal nodes contain operations of type $\oplus_{S,R,L}$. Additionally, the left child corresponds to the root of the left subgraph, while the right child corresponds to the root of the right subgraph. The algorithm `Diagonalize Clique-width` works bottom-up in the parse tree T_G. At each node Q of the tree, the algorithm produces a data structure that we call a *k-box* b_Q, a 4-tuple $[k', k'', M, \Lambda]$. Here k' and k'' are non-negative integers bounded above by k, M is an $m \times m$ symmetric matrix, where $m \leq 2k$, and Λ is a vector whose m entries are labels in $\{1, \ldots, k\}$. If node Q is a leaf, the algorithm initializes a box. If Q is an internal node, it combines the boxes produced by its children into its own box, transmitting it to its parent. At each node, the algorithm operates on a small $O(k) \times O(k)$ matrix, performing congruence operations. These operations represent operations that would be performed on the large $n \times n$ matrix B in order to diagonalize it, and this small matrix represents a *partial view* of the actual $n \times n$ matrix. While processing the node, the algorithm may also produce diagonal elements of a matrix congruent to A. These diagonal elements are appended to a global array as they are produced.

Figure 21 gives a high-level description of the algorithm. We now provide the main ideas of each stage and refer to [29] and [34] for full details. The goal at each node Q_t of T_G is to construct, by means of

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

congruence operations, a matrix associated with the subgraph $G(Q_t)$, which is the subgraph given by the expression rooted at Q_t. This is a diagonal matrix except for at most $2k$ rows and columns, having the form of matrix (5.24):

$$\left(\begin{array}{ccc|c|c} d_1 & & 0 & & \\ & \ddots & & \multicolumn{2}{c}{0} \\ 0 & & d_\ell & & \\ \cline{4-5} & & & \multicolumn{1}{|c|}{M_t^{(0)}} & M_t^{(1)} \\ \cline{4-5} & 0 & & \multicolumn{1}{|c|}{M_t^{(1)T}} & M_t^{(2)} \\ \cline{4-5} \end{array}\right). \tag{5.24}$$

As was the case for the algorithm of Section 3, $M^{(0)} = \mathbf{0}_{k' \times k'}$, $M^{(2)}$ is a $k'' \times k''$ symmetric matrix, and $M^{(1)}$ is a $k' \times k''$ matrix with $0 \leq k' \leq k'' \leq k$ in row echelon form, so that the pair $(M_t^{(1)}, M_t^{(2)})$ defines the box produced at node t. Note that k' can be zero in which case we regard $M^{(0)}$ as empty. To see how the matrix M_{Q_t} produced by box b_{Q_t} fits into the matrix being diagonalized, consider

$$\left(\begin{array}{c|c|c|ccc} D & \multicolumn{2}{c}{0} & & 0 & \\ \cline{2-3} & \multicolumn{1}{|c|}{M^{(0)}} & M^{(1)} & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & \begin{matrix}\cdots\\ \\ \cdots\end{matrix} & \begin{matrix}0\\ \vdots\\ 0\end{matrix} \\ \cline{2-3} 0 & \multicolumn{1}{|c|}{M^{(1)T}} & M^{(2)} & \begin{matrix}\beta_1^1\\ \vdots\\ \beta_{k''}^1\end{matrix} & \begin{matrix}\cdots\\ \\ \cdots\end{matrix} & \begin{matrix}\beta_1^s\\ \vdots\\ \beta_{k''}^s\end{matrix} \\ \cline{2-3} 0 & \begin{matrix}0 & \cdots & 0\\ \vdots & & \vdots\\ 0 & \cdots & 0\end{matrix} & \begin{matrix}\beta_1^1 & \cdots & \beta_{k''}^1\\ \vdots & & \vdots\\ \beta_1^s & \cdots & \beta_{k''}^s\end{matrix} & & M' & \end{array}\right). \tag{5.25}$$

The matrix in (5.25) shows how the large matrix B has been transformed by the operations performed to produce the small submatrix M_{Q_t}. The diagonal matrix D represents *all* diagonalized elements produced up to step t in the algorithm. Also, the entries on the rows and columns corresponding to the k' rows and columns of $M^{(0)}$ are necessarily 0 outside M, which is precisely the distinction between $M^{(0)}$ and $M^{(2)}$. The β_i^j are zero-one entries in the partially diagonalized matrix, whose relation with the corresponding entries in the original matrix B will be explained below. It is important to keep in mind that after node Q_t has been processed, all vertices in the subgraph $G(Q_t)$ correspond to rows in D or M_{Q_t}. Some rows of D may correspond to vertices outside of $G(Q_t)$, which have been diagonalized in earlier steps. The submatrix M' in (5.25) contains all undiagonalized rows $w \notin G(Q_t)$, may include vertices in $M_{Q_{t'}}$ for $t' \neq t$, and becomes empty after the last iteration of the algorithm.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

We say that the k' rows in $M^{(0)}$ have *type-i*, and that the k'' rows of $M^{(2)}$ have *type-ii*. Recall that these rows are indexed by the rows of the original matrix, and each of them has a label in $[k]$ coming from the slick expression that defines $G(Q_t)$. Each row of the original matrix is associated with a single leaf of the parse tree, so that the sets of type-i and type-ii rows coming from different branches of the tree are always disjoint. The important information is $M = M_{Q_t}$, these labels, and the integers k' and k'', which are all stored in the box $b_{Q_t} = [k', k'', M, \Lambda]$ mentioned above, whose *size* is $k' + k''$. To ensure that $M^{(2)}$, whose rows have type-ii, has order $k'' \leq k$ at the end of step t, each label appears in at most one type-ii row. It is helpful to keep in mind that a row always begins as a type-ii row, then becomes a type-i row, and finally becomes diagonalized.

```
    input:  $k$-expression $Q = i(v)$, a scalar $x$
    output:  $[0, 1, [x], [i]]$

LeafBox($Q$,$x$)
    return:  $[0, 1, [x], [i]]$
```

FIGURE 22. *Procedure LeafBox.*

When the node Q_t is a leaf corresponding to a subexpression $i(v)$, $B_{Q_t} = [x]$. This means that the box contains a 1×1 matrix $M_{Q_t} = [x]$, whose row is labeled i, $k' = 0$, and $k'' = 1$. Procedure `LeafBox` of Fig. 22 describes the box corresponding to a leaf.

Now we will describe `Combine Boxes`, that is, we explain how an internal node produces its box from the boxes transmitted by its children. For simplicity, when referring to operations, we always mention the row operations, with the understanding that the corresponding column operations are also performed.

$$\left(\begin{array}{cc} \begin{array}{|c|c|}\hline 0 & \begin{matrix} * & * & * \\ & * & * \\ & & * \end{matrix} \\ \hline \begin{matrix} * & & \\ * & * & \\ * & * & * \end{matrix} & M_\ell^{(2)} \\ \hline \end{array} & \begin{matrix} 0 & 0 \\ 0 & F \end{matrix} \\ \begin{matrix} 0 & 0 \\ 0 & F^T \end{matrix} & \begin{array}{|c|c|}\hline 0 & \begin{matrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \end{matrix} \\ \hline \begin{matrix} * & & & \\ * & * & & \\ * & * & * & \\ * & * & * & * \end{matrix} & M_r^{(2)} \\ \hline \end{array} \end{array}\right) \tag{5.26}$$

When processing $Q_t = Q_\ell \oplus_{S,L,R} Q_r$, the matrix M_{Q_t} is constructed by first taking the disjoint union of the matrices transmitted by its children and then updating the entries vw where v and w are type-ii vertices of different sides. Precisely, if $(i, j) \in S$, v is a type-ii vertex in $G(Q_\ell)$ with label i, and w is a type-ii vertex in $G(Q_r)$ with label j, we place a one in the row (column) of v and column (row) of w. Observe that the unique label condition imposed on $M^{(2)}$ implies that at most one pair of entries will be modified for

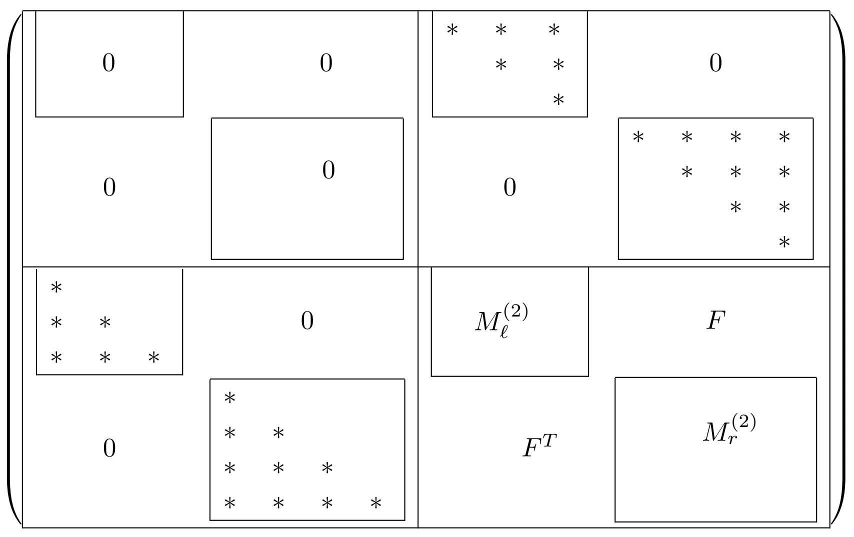
Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

any element of S. Let F be the block of ones defining these edges. Then node Q_t starts with a matrix as in (5.26).

Next, Q_t relabels the rows of M_{Q_ℓ} and M_{Q_r}, using the functions L and R, respectively. Using permutations of rows and columns, it combines the k'_ℓ type-i rows from M_{Q_ℓ} with the k'_r type-i rows of M_{Q_r}, and combines the k''_ℓ type-ii rows in M_{Q_ℓ} with the k''_r type-ii rows in M_{Q_r}, so that they are contiguous in M_{Q_t}.

We observe that the zero pattern outside the matrices M_{Q_ℓ} and M_{Q_r} implies that the type-i rows from M_{Q_ℓ} and M_{Q_r} are still type-i in (5.26). Additionally, from the fact that the type-i rows of M_{Q_ℓ} and M_{Q_r} are distinct, we see that the new $M^{(1)}$ of M_{Q_t}, which is formed by placing $M_\ell^{(1)}$ on top of $M_r^{(1)}$, is already in row echelon form. We illustrate in equation (5.27) the transformation of the matrix in (5.26) after we perform the permutations described above. Hence, these permutations of rows and columns lead to a matrix M_Q in the required form. In this matrix, $k' = k'_\ell + k'_r$ and $k'' = k''_\ell + k''_r$. By construction, we have $k' \le k''$, however we are not guaranteed that $k'' \le k$.

(5.27) $M_{Q_t} =$



Next, we explain how to use congruence operations to produce a box from M_{Q_t} in a way that $k'' \le k$ while keeping $k' \le k''$. As a byproduct, new permanent elements may be added to the diagonal submatrix D. The matrix M_{Q_t} is shrunk in two steps:

(i) If two type-ii rows j and j' have the same label, then congruence operations are performed to turn one of them, say $v = j'$, into type-i.
(ii) Congruence operations are performed with the aim of diagonalizing the row corresponding to v, or of inserting v into M_{Q_t} in a way that $M_{Q_t}^{(0)}$ is the zero matrix and $M_{Q_t}^{(1)}$ is in row echelon form and has a pivot on each row.

Note that, if steps (i) and (ii) are repeatedly applied to pairs of type-ii rows with the same label until there are no such pairs, we immediately obtain $k'' \le k$. By ensuring that $M_{Q_t}^{(1)}$ remains in row echelon form with pivots on each row (or becomes empty), we ensure that $k' \le k''$.

To achieve (i), fix two type-ii rows j and j' have the same label. By Remark 5.2, their rows and columns must agree outside of M_{Q_t}. Therefore in (5.25), $\beta^i_j = \beta^i_{j'}$ for all i. Performing the operations

$$R_{j'} \leftarrow R_{j'} - R_j,\ C_{j'} \leftarrow C_{j'} - C_j, \tag{5.28}$$

eliminates any nonzero entries of row j', except possibly for the columns within M_{Q_t}, and thus transforms the type-ii row j' into a type-i row.

Part (ii) is done in the same fashion as procedure `ForgetBox` of Section 3. We recall some of the reasoning here. Suppose that v is the row to be inserted. For convenience, after exchanging rows and columns, we look at M_{Q_t} in the following way:

$$M_{Q_t} = \begin{array}{|c|c|c|} \hline d_v & \mathbf{x}_v & \mathbf{y}_v \\ \hline \mathbf{x}_v^T & \mathbf{0}_{k' \times k'} & M_{Q_t}^{(1)} \\ \hline \mathbf{y}_v^T & M_{Q_t}^{(1)T} & M_{Q_t}^{(2)} \\ \hline \end{array}. \tag{5.29}$$

Here, the first row and column represent the row and column in M_{Q_t} associated with v. In particular, d_v is the diagonal element, while x_v is the vector whose elements correspond to the other type-i rows in $M_{Q_t}^{(0)}$. The vector y_v represents the correspondence with the type-ii rows in $M_{Q_t}^{(2)}$.

Depending on the value of d_v, on the vectors $\mathbf{x}_v$ and $\mathbf{y}_v$, we proceed in different ways. The preferred outcome is to diagonalize the row/column associated with v, and this is particularly easy when $d_v \neq 0$. Indeed, in this case we simply use d_v to annihilate all the elements in row/column v, performing the congruence operations.

In general, we deal with v exactly as in procedure `ForgetBox` of Fig. 13 in Section 3, and we refer the reader to the discussion following it for the full description of each case and its motivation. We recall that the row corresponding to v and an additional type-i row are diagonalized with diagonal entries of opposite signs if $\mathbf{x}_v \neq 0$ (case 2 of `Combine Boxes`), v is diagonalized with diagonal element d_v if $\mathbf{x}_v$ is empty or zero and $d_v \neq 0$ (subcases 1a and 1c), or if $d_v = 0$ and $\mathbf{x}_v$ and $\mathbf{y}_v$ are both empty or zero (subcase 1a). The only remaining case is when $d_v = 0$, $\mathbf{x}_v$ is empty or zero, and $\mathbf{y}_v \neq 0$. Then, the algorithm performs steps to insert $\mathbf{y}_v$ into the matrix in row echelon form (subcase 1b of `Combine Boxes`).

This concludes the description of the procedure Combine Boxes, which appears in Fig. 23. Note that all operations performed in the procedure are congruence operations. To conclude the description of the algorithm, we can assume that at the root

$$Q_G = Q_\ell \oplus_{L,R,S} Q_r,$$

L and R map all vertices to the same label, as labels are no longer needed. After applying `Combine Boxes`, we will obtain $k'' = 1$ and $M^{(0)}$ is either zero or empty. If it is empty then the 1×1 matrix $M^{(2)}$ contains the final diagonal element. Otherwise M_Q is a 2×2 matrix having form

$$\begin{pmatrix} 0 & a \\ a & b \end{pmatrix}.$$

If $b \neq 0$, it can be made fully diagonal using the congruence operations in (3.14) (where u and v denote the first and second rows, respectively). If $b = 0$, it can be made fully diagonal using the congruence operations in (3.16) and (3.17). This is what we call `DiagonalizeBox` in the algorithm of Fig. 21.

The correctness of the algorithm is proven in [29], by showing that a series of invariants hold (by induction) after the execution of each step. Moreover, a bound of $O(k^2 n)$ for the complexity of the algorithm is obtained by a clever accounting for the time spent by executing the elementary operations.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

```
Input:  two $k$-boxes $b_{Q_\ell}$ and $b_{Q_r}$
Output:  a $k$-box $b_Q$

Combine Boxes($b_{Q_\ell}$,$b_{Q_r}$)
   form matrix $M$ as in (5.26);
   relabel rows with functions $L$ and $R$;
   combine type-i rows (columns), combine type-ii rows (columns);
   ensure type-ii rows have distinct labels:
   for each pair $(v,w)$ of type-ii rows with equal labels
      execute operation as in (5.28);
         if $\mathbf{x}_v$ is empty or $\mathbf{0}$
            then if $\mathbf{y}_v$ is empty or $\mathbf{0}$ then //subcase 1a
               then add $(v,d_v)$ to $D$
               remove row $v$ from $M$
               else if $d_v = 0$ then          //subcase 1b
                     then do row/column operations as in (3.13)
                     if row $v$ gets a pivot
                        then insert the row into $M^{(1)}$
                        else add $0$ to $D$ and remove row from $M$
                     else                      //subcase 1c
                        use $d_v$ to diagonalize row/column $v$ as in (3.14)
                        add $d_v$ do $D$ and remove row $v$ form $M$
            else // Here $\mathbf{x}_v \neq 0$.  //case 2
            let $u$ be the vertex of the rightmost nonzero entry of $x_v$
            if $x_v$ has other nonzero entries
               then eliminate them with the operations (3.15)
            if $d_v \neq 0$
               then perform the operations (3.16)
            perform the operations (3.17)
            use $d_v$ and $d_u$ to diagonalize rows/columns $v$ and $u$ as in (3.13)
            add $d_v$ and $d_u$ to $D$ and eliminate rows $v$ and $u$ from $M$.
```

FIGURE 23. *Procedure* `Combine Boxes`*.*

THEOREM 5.3. *[29] Let G be a graph with adjacency matrix A, given by a slick expression Q_G with parse tree T, and let $x \in \mathbb{R}$. Algorithm* `Diagonalize Clique-width` *correctly outputs the diagonal elements of a diagonal matrix congruent to $B = A - xI$. Moreover, this is done in $O(k^2 n)$ operations.*

5.3. Example. To see how the algorithm acts on a concrete example, we refer to the graph on the left of Fig. 20, which may be constructed with the slick 2-expression given by equation (5.23) and whose parse tree is given in Fig. 20.

We apply the algorithm to the graph defined by this slick 2-expression for $x = \frac{1}{2}$. Since $k = 2$, the boxes created by the leaves may be of the following two types:

$$1(v) : [k', k'', M, \Lambda] = [0, 1, (1/2), (1)], \quad \text{or } 2(v) : [0, 1, (1/2), (2)],$$

This means that $k' = 0$ (thus $M^{(0)}$ is empty) and that $k'' = 1$ and $M^{(2)} = (1/2)$.

The nodes C and D of the parse tree lead to identical applications of `Combine Boxes`. This is depicted below, where the box is obtained from the box of the children as in (5.26). Since the relabeling functions L and R are equal to the identity and there are no vertices with the same label, the procedure ends and the nodes transmit the box

$$\left[0, 2, \begin{pmatrix} 1/2 & 1 \\ 1 & 1/2 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}\right],$$

to their parent. In the above terminology, this means that $M^{(0)}$ is empty ($k' = 0$), $k'' = 2$, and $M^{(2)} = \begin{pmatrix} 1/2 & 1 \\ 1 & 1/2 \end{pmatrix}$ and the labels of its rows are 1 and 2, respectively. Up to this point, we did not need to apply the congruence operations described above to insure type-ii rows have unique labels or to insert type-i rows into $M^{(0)}$.

Next, we process node B, which receives boxes from nodes C and D. Given that $S = \{(1,1),(1,2),(2,1)\}$, $L = 1 \to 2$ and $R = 2 \to 1$, the initial matrix given in (5.26) and the corresponding vector of labels are

$$M = \begin{pmatrix} 1/2 & 1 & 1 & 1 \\ 1 & 1/2 & 1 & 0 \\ 1 & 1 & 1/2 & 1 \\ 1 & 0 & 1 & 1/2 \end{pmatrix}, \quad \text{Lab} = \begin{pmatrix} 2 \\ 2 \\ 1 \\ 1 \end{pmatrix}.$$

We notice that rows 1 and 2 are type-ii with the same label. We then perform the operations $R_1 \leftarrow R_1 - R_2$, $C_1 \leftarrow C_1 - C_2$, leading to the matrix

$$M = \left(\begin{array}{c:ccc} -1 & 1/2 & 0 & 1 \\ \hdashline 1/2 & 1/2 & 1 & 0 \\ 0 & 1 & 1/2 & 1 \\ 1 & 0 & 1 & 1/2 \end{array}\right).$$

The algorithm then attempts to insert the type-i row into the empty matrix $M^{(0)}$. Here $\mathbf{x}_v$ is empty, $d_v = -1 \neq 0$, $\mathbf{y}_v = \left[\frac{1}{2}, 0, 1\right]$, leading to **Subcase 1c**, where d_v is used to diagonalize the first row and column, leading to

$$M = \left(\begin{array}{c:ccc} -1 & 0 & 0 & 0 \\ \hdashline 0 & 3/4 & 1 & 1/2 \\ 0 & 1 & 1/2 & 1 \\ 0 & 1/2 & 1 & 3/2 \end{array}\right).$$

Therefore, our first diagonal value is $d_1 = -1$. We remove this row/column resulting in the matrix

$$M = \begin{pmatrix} 3/4 & 1 & 1/2 \\ 1 & 1/2 & 1 \\ 1/2 & 1 & 3/2 \end{pmatrix}, \quad \text{Lab} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}.$$

Now, as rows 2 and 3 have equal labels, we perform $R_2 \leftarrow R_2 - R_3$, $C_2 \leftarrow C_2 - C_3$ to turn row 2 into type-i, so that the type-ii rows have unique labels. For better visualization, we perform $R_2 \longleftrightarrow R_1$ and

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

$C_2 \longleftrightarrow C_1$ and get to the matrix

$$M = \left(\begin{array}{c:cc} 0 & 1/2 & -1/2 \\ \hdashline 1/2 & 3/4 & 1/2 \\ -1/2 & 1/2 & 3/2 \end{array}\right), \quad \text{Lab} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}.$$

To insert the new type-i row into a proper box, the algorithm performs **Subcase 1b**, as $d_v = 0, x_v = \emptyset$ and $y_v = [1/2, -1/2]$. As the row vector y_v is itself a matrix in row echelon form, there is no need to perform any other operation. Hence, node B transmits to its parent A the box

$$\left[1, 2, \left(\begin{array}{c:cc} 0 & 1/2 & -1/2 \\ \hdashline 1/2 & 3/4 & 1/2 \\ -1/2 & 1/2 & 3/2 \end{array}\right), \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}\right].$$

Finally, node A receives the box produced by the leaf $1(v_2) : [0, 1, (1/2), (1)]$ and the box transmitted by node B. Procedure *Combine Boxes* first forms the matrix M using $S = \{(1, 1)\}$ and the relabeling functions $L = id$, $R : 2 \longrightarrow 1$:

$$M = \begin{pmatrix} 1/2 & 0 & 0 & 1 \\ 0 & 0 & 1/2 & -1/2 \\ 0 & 1/2 & 3/4 & 1/2 \\ 1 & -1/2 & 1/2 & 3/2 \end{pmatrix}, \quad \text{Lab} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

The next step is to regroup the type-i rows and the type typo-ii rows, performing $R_1 \longleftrightarrow R_2$ and $C_1 \longleftrightarrow C_2$, we arrive at

$$M = \left(\begin{array}{c:ccc} 0 & 0 & 1/2 & -1/2 \\ \hdashline 0 & 1/2 & 0 & 1 \\ 1/2 & 0 & 3/4 & 1/2 \\ -1/2 & 1 & 1/2 & 3/2 \end{array}\right).$$

As before, to ensure uniqueness of labels in type-ii rows, we perform $R_2 \longleftarrow R_2 - R_3$ and $C_2 \longleftarrow C_2 - C_3$. To better visualize the attempt of inserting the new type-i row into the matrix M, we perform $R_1 \longleftrightarrow R_2$ e $C_1 \longleftrightarrow C_2$. The two steps are represented below.

$$M = \left(\begin{array}{cc:cc} 0 & -1/2 & 1/2 & -1/2 \\ -1/2 & 5/4 & -3/4 & 1/2 \\ \hdashline 1/2 & -3/4 & 3/4 & 1/2 \\ -1/2 & 1/2 & 1/2 & 3/2 \end{array}\right), \text{ followed by } M = \left(\begin{array}{cc:cc} 5/4 & -1/2 & -3/4 & 1/2 \\ -1/2 & 0 & 1/2 & -1/2 \\ \hdashline -3/4 & 1/2 & 3/4 & 1/2 \\ 1/2 & -1/2 & 1/2 & 3/2 \end{array}\right).$$

Here, $d_v = 5/4, x_v = [-1/2]$ e $y_v = [-3/4, 1/2]$, implying that we are in Case 2 of the procedure *Combine Boxes*. Since x_v has no other nonzero entry, we need not eliminate them. As $d_v = 5/4 \neq 0$, we execute the operations described by *Combine Boxes* to diagonalize both type-i rows/columns simultaneously. First, we perform $R_1 \longleftarrow R_1 + \frac{5}{4}R_2$ and $C_1 \longleftarrow C_1 + \frac{5}{4}C_2$. Next, the operations $R_2 \longleftarrow R_2 + \frac{1}{2}R_1$, $C_2 \longleftarrow C_2 + \frac{1}{2}C_1$, $R_1 \longleftarrow R_1 - R_2$ and $C_1 \longleftarrow C_1 - C_2$. The two steps produce

$$M = \left(\begin{array}{cc:cc} 0 & -1/2 & -1/8 & -1/8 \\ -1/2 & 0 & 1/2 & -1/2 \\ \hdashline -1/8 & 1/2 & 3/4 & 1/2 \\ -1/8 & -1/2 & 1/2 & 3/2 \end{array}\right), \text{ followed by } M = \left(\begin{array}{cc:cc} 1/2 & 0 & -9/16 & 7/16 \\ 0 & -1/2 & 7/16 & -9/16 \\ \hdashline -9/16 & 7/16 & 3/4 & 1/2 \\ 7/16 & -9/16 & 1/2 & 3/2 \end{array}\right).$$

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

We then can use $d_2 = \dfrac{1}{2}$ and $d_3 = -\dfrac{1}{2}$ to eliminate the off-diagonal values. We obtain two additional diagonal values: d_2 and d_3. Removing these rows/columns, the resulting matrix is

$$M = \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 7/4 \end{pmatrix}, \qquad \text{Lab} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Performing the same procedure to guarantee the uniqueness of labels, we fall into **Subcase 1c** and, diagonalizing the matrix, we obtain the remaining diagonal values $d_4 = \dfrac{5}{4}$ and $d_5 = \dfrac{1}{2}$.

Hence, the diagonal matrix obtained by our algorithm id $D = \left(-1, \dfrac{1}{2}, -\dfrac{1}{2}, \dfrac{5}{4}, \dfrac{1}{2}\right)$. This means there are three eigenvalues greater than $-\dfrac{1}{2}$ and two eigenvalues less than $-\dfrac{1}{2}$. The actual spectrum of G is given (approximately) by the set

$$\text{Spec}(G) = \{2.94, 0.62, -0.46, -1.47, -1.62\}.$$

Finally, we discuss some implementation issues of `Diagonalize Clique-width` by itemizing a few features of our algorithm that we added to simplify the description, but are not necessary in an efficient implementation. It is worth noticing that the whole algorithm is very fast, as there are no large constants hidden in the O-notation.

(a) Since the matrix in equation (5.24) is not actually computed, one may easily write `Diagonalize Clique-width` as a recursive algorithm.
(b) It is not necessary to perform permutations of rows and columns to separate them according to type, it suffices to keep track of the vertices of each type in matrices M_{Q_t}.
(c) The requirement that all vertices are relabeled with the same label at the root node is not crucial. The root could just produce an arbitrary box from the boxes transmitted by its children, and the final step of the algorithm, DiagonalizeBox, could just diagonalize this box with congruence operations in any way.

Part III - Applications

6. Recurrence relations.

In this section, we explain how the tree algorithm of Fig. 2 behaves when applied to a pendant path of a tree T. We first notice that, by Corollary 1.2, the diagonalization of $M - xI$ (where M is a matrix having an underlying tree) tells us how many eigenvalues of M are less than, equal to and greater than x. Following the application of the algorithm in Fig. 2 from a leaf towards the root, we see that some recurrence relations appear. These recurrence relations may be studied in a unified way, and their analytical properties have been used to solve some long standing problems in spectral graph theory.

As a motivation, we observe the numerical values produced by the algorithm of Fig. 2 when applied to a path P_n in three distinct situations. In all three examples, the root v_n of P_n is chosen as one of its endpoints, and we run the algorithm starting from the other endpoint v_1.

(1) Consider an application to $M = A - \lambda I$, where A is the adjacency matrix of P_n and λ is a fixed real number. Note that the off-diagonal nonzero entries are equal to 1 and that $m_{ii} = -\lambda$. Applying the

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

algorithm gives $z_1 = -\lambda$, $z_2 = -\lambda - \frac{1^2}{z_1} = -\lambda - \frac{1}{z_1}$ and so on. Therefore, we have a recursion

$$\begin{cases} z_1 = -\lambda \\ z_{j+1} = -\lambda - \frac{1}{z_j},\ 1 \leq j \leq n-1. \end{cases} \tag{6.30}$$

Of course, this recursion is defined only up to the first index j such that $z_j = 0$. We shall also make this assumption for the sequences below. Moreover, by taking longer paths, we may produce arbitrarily long (albeit finite) sequences.

(2) Let $L(T)$ be the *Laplacian matrix* of a tree T. Recall that this is the matrix for which each diagonal entry ℓ_{ii} is the degree of vertex i and for which, given $i \neq j$, $\ell_{ij} = -1$ if $\{i,j\} \in E(T)$ and $\ell_{ij} = 0$ otherwise. Let $d = 2 - \frac{2}{n}$ be the average degree of T. We apply the diagonalization algorithm to $M = L(P_n) - dI$. Starting at the end vertex v_1, we have $a_1 = 1 - d = -1 + \frac{2}{n}$, $a_2 = 2 - d - \frac{(-1)^2}{a_1} = \frac{2}{n} - \frac{1}{a_1}$ and so on. Therefore, we have a recursion

$$\begin{cases} a_1 = -1 + \frac{2}{n} \\ a_{j+1} = \frac{2}{n} - \frac{1}{a_j},\ 1 \leq j \leq n-2. \end{cases} \tag{6.31}$$

Note that $a_n = \frac{2}{n} - 1 - \frac{1}{a_{n-1}}$ because v_n has degree 1, but we will again be interested on the structure of the sequence for arbitrarily long paths.

(3) Let $\mathcal{L}(T)$ be the *normalized Laplacian matrix* of a tree T, that is, the matrix for which all diagonal elements are equal to 1 and for which, given $i \neq j$, $t_{ij} = \frac{-1}{\sqrt{d_i d_j}}$ if $\{i,j\} \in E(T)$ and $t_{ij} = 0$ otherwise, where d_i and d_j denote the degrees of i and j, respectively. For a real value $\lambda \in [0,\ 2]$, we apply the algorithm to $M = \mathcal{L}(P_n) - \lambda I$, where $n \geq 3$. Here, the nonzero off-diagonal entries m_{ij} are equal to $\frac{-1}{\sqrt{2}}$ if $\{i,j\} \cap \{1,n\} \neq \emptyset$ and to $\frac{-1}{\sqrt{2\cdot 2}} = -\frac{1}{2}$ otherwise. Starting at the end vertex v_1, we obtain $x_1 = 1 - \lambda$, $x_2 = 1 - \lambda - \left(\frac{-1}{\sqrt{2}}\right)^2 \frac{1}{x_1} = 1 - \lambda - \frac{1}{2x_1}$, $x_3 = 1 - \lambda - \left(\frac{-1}{2}\right)^2 \frac{1}{x_2} = 1 - \lambda - \frac{1}{4x_2}$, $x_4 = 1 - \lambda - \left(\frac{-1}{2}\right)^2 \frac{1}{x_3} = 1 - \lambda - \frac{1}{4x_3}$ and so on. Therefore, we have a recursion

$$\begin{cases} x_1 = 1 - \lambda \\ x_2 = 1 - \lambda - \frac{1}{2(1-\lambda)} \\ x_{j+1} = 1 - \lambda - \frac{1}{4x_j},\ 2 \leq j \leq n-2. \end{cases} \tag{6.32}$$

The final step gives $x_n = 1 - \lambda - \frac{1}{2x_{n-1}}$.

As these examples show, applying the tree algorithm on a path produces certain numerical rational sequences. They all may be seen in a unified elementary form given by

$$x_{j+1} = \varphi(x_j), j \geq 1 \tag{6.33}$$

where $\varphi(t) = \alpha + \frac{\gamma}{t}$, for $t \neq 0$, $\alpha, \gamma \in \mathbb{R}$ are fixed numbers ($\gamma \neq 0$), and x_1 is a given initial condition. The explicit solution is a function $j \to f(j)$ such $x_j = f(j)$ for $j \geq 1$ and has been studied in [47]. The following result describes the structure of explicit solutions.

THEOREM 6.1. [47] *Consider the recurrence relation $x_{j+1} = \alpha + \frac{\gamma}{x_j}$, where $\alpha, \gamma \in \mathbb{R}$ are fixed numbers ($\gamma \neq 0$) and x_1 is a given initial condition. Then the general solution has one of three possible types according the sign of $\Delta = \alpha^2 + 4\gamma$:*

Type 1: *For $\Delta = 0$, the solution is*

$$x_j = \theta\left(1 + \frac{1}{(\beta + j)}\right),$$

where $\theta = \frac{\alpha}{2}$ and $\beta \in \mathbb{R}$ is defined in terms of the initial condition $x_1 \neq \theta$ by the formula $\beta = -1 + \frac{\theta}{x_1 - \theta}$. If $x_1 = \theta$ then $x_j = \theta$, $\forall j \geq 1$ is the solution.

Type 2: *For $\Delta > 0$, the solution is*

$$x_j = \theta + \frac{\theta' - \theta}{\beta\left(\frac{\theta}{\theta'}\right)^j + 1},$$

where $\theta = \frac{\alpha}{2} - \frac{1}{2}\sqrt{\alpha^2 + 4\gamma}$, $\theta' = \frac{\alpha}{2} + \frac{1}{2}\sqrt{\alpha^2 + 4\gamma}$ and $\beta \in \mathbb{R}$ is defined in terms of the initial condition $x_1 \neq \theta$ by the formula $\beta = \frac{\theta'}{\theta}\left(\frac{\theta' - \theta}{x_1 - \theta} - 1\right)$. If $x_1 = \theta$ then $x_j = \theta$, $\forall j \geq 1$ is the solution. Moreover, the following hold:

(a) If $x_1 < \theta$, then $x_j < \theta$ for all $j \geq 1$, x_j is an increasing sequence and $\lim_{j\to\infty} x_j = \theta$.

(b) If $\theta < x_1 < \theta'$, then $\theta < x_j < \theta'$ for all $j \geq 1$, x_j is a decreasing sequence and $\lim_{j\to\infty} x_j = \theta$.

Type 3: *For $\Delta < 0$, the solution is*

$$x_j = \rho\left(\cos(\phi) - \sin(\phi)\tan(j\phi + \omega)\right),$$

where $\rho = \sqrt{-\gamma}$, $\phi = \arctan\left(\frac{\sqrt{-\alpha^2 - 4\gamma}}{\alpha}\right)$ if $\alpha > 0$, $\phi = \arctan\left(\frac{\sqrt{-\alpha^2 - 4\gamma}}{\alpha}\right) + \pi$ if $\alpha < 0$ and $\omega \in [0, 2\pi)$ is defined in terms of the initial condition x_1 by the formula

$$\omega = -\phi + \arctan\left(\cot(\phi) - \frac{x_1}{\rho}\csc(\phi)\right).$$

If $\alpha = 0$, then $x_j = \begin{cases} \frac{\gamma}{x_1}, & j = 2k \\ x_1, & j = 2k+1 \end{cases}$, $j \geq 1$ is the solution of $x_{j+1} = \frac{\gamma}{x_j}$ for a given $x_1 \neq 0$.

Example 6.2. *An H-*shape tree*, or H-tree for short, is a tree obtained from a path P_{m+1}, $m \geq 1$, of length m by attaching two paths of length at least 1 to each endpoint of P_{m+1}. The endpoints of P_{m+1} are called the left and the right endpoints, respectively, and we denote an H-tree by $H(\ell_1, \ell_2, m, \ell_3, \ell_4)$, where $\ell_1 \leq \ell_2$ are the lengths of the paths attached to the left endpoint, and $\ell_3 \leq \ell_4$ are the lengths of the paths attached to the right endpoint.*

Consider applying the algorithm of Fig. 2 to the adjacency matrix A of $H = H(\ell_1, \ell_2, m, \ell_3, \ell_4)$ depicted in Figure 24. It is known (see for example [60]) that the maximum eigenvalue of A is greater than 2. We will show next that at most one eigenvalue of A is greater than $\frac{3}{\sqrt{2}} \approx 2.121320343$. In order to do that, we invoke Corollary 1.2 and show that all but possibly one diagonal value of a diagonal matrix congruent to $A - \lambda I$ are negative, for $\lambda = \frac{3}{\sqrt{2}} > 2$. We choose as the root v of H the right endpoint of the path P_{m+1}.

As the algorithm processes the vertices on the pendant paths, the values obtained satisfy the recurrence relations obtained for paths, that is,

$$\begin{cases} z_1 = -\lambda \\ z_i = -\lambda - \frac{1}{z_{i-1}}, \quad i = 2, \ldots, \ell_k. \end{cases} \tag{6.34}$$

For the first m vertices on the path P_{m+1}, starting from its left endpoint, we get

$$\begin{cases} b_1 = -\lambda - \frac{1}{z_{\ell_1}} - \frac{1}{z_{\ell_2}} \\ \\ b_i = -\lambda - \frac{1}{b_{i-1}}, \quad i = 2, \ldots, m. \end{cases} \tag{6.35}$$

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

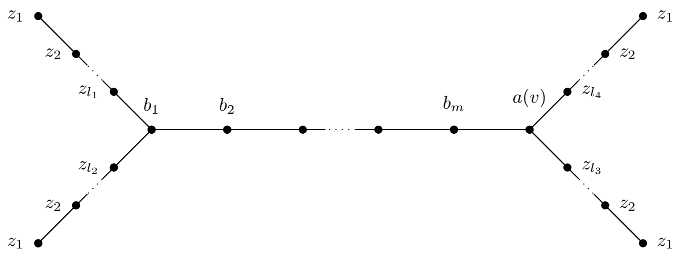


FIGURE 24. *Valued H-tree after application of the algorithm.*

The value at the root is

$$a(v) = -\lambda - \frac{1}{z_{\ell_3}} - \frac{1}{z_{\ell_4}} - \frac{1}{b_m}. \tag{6.36}$$

In the framework of Theorem 6.1, the recurrence relations (6.34) and (6.35) have $\alpha = -\lambda$ and $\gamma = -1$. By our choice of $\lambda > 2$, $\Delta = \alpha^2 + 4\gamma = \lambda^2 - 4 > 0$ which is a type 2 recurrence.

For Equation (6.34), we see that $z_i = \theta + \frac{\theta' - \theta}{\beta\left(\frac{\theta}{\theta'}\right)^i + 1}$, with $\theta = \frac{-\lambda}{2} - \frac{\sqrt{\lambda^2-4}}{2}$, $\theta' = -\frac{\gamma}{\theta} = \frac{1}{\theta}$. In particular, when $\lambda = 3/\sqrt{2}$, we have $z_1 = -\lambda = -3/\sqrt{2} < \theta = -\sqrt{2}$. Hence, by Theorem 6.1(a), z_i is strictly increasing and converges to $\theta = -\sqrt{2}$. So all z_i in Fig. 24 are negative. Moreover, the general solution may be written as $z_i = \theta + \frac{\theta' - \theta}{\beta\left(\frac{\theta}{\theta'}\right)^i + 1}$. As $\beta = \frac{\theta'}{\theta}\left(\frac{\theta' - \theta}{z_1 - \theta} - 1\right)$, substituting the given values, we arrive at $\beta = -1$, meaning that

$$z_i = -\frac{\sqrt{2}\left(2^{i+1} - 1\right)}{2^{i+1} - 2}. \tag{6.37}$$

For the recurrence given by (6.35), when the algorithm is applied with $\lambda = 3/\sqrt{2}$, the value for $b_1 = -\frac{3}{\sqrt{2}} - \frac{1}{z_{\ell_1}} - \frac{1}{z_{\ell_2}}$ satisfies

$$-\sqrt{2} = \theta < b_1 < \theta' = -\frac{\sqrt{2}}{2}, \tag{6.38}$$

for any positive values of ℓ_1 and ℓ_2. To see why this is true, observe that, as $z_i < \theta$, $b_1 = 3/\theta - 1/z_{\ell_1} - 1/z_{\ell_2} < 3/\theta - 2/\theta = \theta'$. For the other inequality, we see that $z_i \geq -3/\sqrt{2}$; hence ,$b_1 = -3/\sqrt{2} - 1/z_{\ell_1} - 1/z_{\ell_2} > -3/\sqrt{2} + 2\sqrt{2}/3 = -5\sqrt{2}/6 = 5\theta/6 > \theta$. By Theorem 6.1(b), this implies that $b_i \to -\sqrt{2}$, decreasingly and, consequently, all values of b_i are negative.

It follows that all but the last value, given by Equation (6.36), produced by the algorithm `Diagonalize`$(A - \lambda I)$ *of Fig. 2 are negative. Hence, we conclude that at most one eigenvalue of H is greater than or equal to $3/\sqrt{2}$.*

This example implies that, if the value at root is positive, then a single eigenvalue (necessarily the maximum eigenvalue, known as *index* or *spectral radius*) of an H-tree is larger than $3/\sqrt{2}$. Characterizing the graphs whose index is less than $3/\sqrt{2}$ is a challenging problem that has been studied by many authors but not yet completely solved (see [60],[19] for a structural characterization). It is related to the celebrated Hoffman-Smith (HS) limit points that will be discussed in the next section.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

6.1. Applications to limit points. A real number r is said to be a *limit point* of the spectral radii of graphs if there exists a sequence $\{G_k\}$ of graphs such that

$$\rho(G_i) \neq \rho(G_j), \quad \text{for all } i \neq j, \text{ and } \lim_{k\to\infty} \rho(G_k) = r,$$

where $\rho(G)$ is the spectral radius (or index) of the adjacency matrix of G.

A landmark paper in this area is due to J. Shearer [54], who proved that any real number $\lambda \geq \sqrt{2+\sqrt{5}}$ is a limit point of the spectral radii of graphs. Precisely, there exists an infinite sequence of graphs G_k, $k = 1, 2, \ldots$, whose spectral radii $\rho(G_1) < \cdots < \rho(G_k) < \lambda$ is an increasing sequence and $\lim_{k\to\infty} \rho(G_k) = \lambda$.

In general, the techniques used to prove that a real number is a limit point are intricate. As an illustration, we show here that the results in this survey are applicable for finding limit points of spectral radii of graphs.

Consider the starlike tree $T_{1,n,n}$ illustrated in Figure 25 obtained by connecting an isolated vertex to endpoints of two copies of P_n and to another isolated vertex.

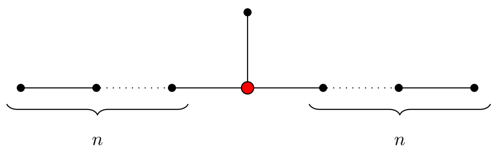


FIGURE 25. *The tree $T_{1,n,n}$.*

The following result from [19] implies that $\sqrt{2+\sqrt{5}}$ is a limit point with respect to the adjacency matrix. We reprove it here using our technique.

THEOREM 6.3 ([19]).

$$\lim_{n\to\infty} \rho(T_{1,n,n}) = \sqrt{2+\sqrt{5}} \approx 2.058.$$

Proof. Consider the tree $T_{1,n,n}$ for $n \geq 3$ and let $\lambda_n = \rho(T_{1,n,n})$ denote its spectral radius (with respect to the adjacency matrix). By a result of [43], $2 < \lambda_n < \frac{3}{\sqrt{2}} \approx 2.12$ for all $n \geq 3$.

Let v_n denote the unique vertex of degree three of $T_{1,n,n}$ and order the vertices in each branch of $T_{1,n,n}$ from the leaves towards the root. For each $n \geq 3$, the algorithm will be applied to the matrix $A(T_{1,n,n}) - \lambda_n I$. Note that the diagonalization algorithm is being applied infinitely many times to an infinite sequence of trees. If we restrict our analysis to the vertices on the branches P_n, we derive the recurrence relations $z_1^{(n)} = -\lambda_n$ and $z_i^{(n)} = -\lambda_n - 1/z_{i-1}^{(n)}$. This is exactly the relation discussed in Example 6.2 (see Equation (6.34)).

Since $\Delta_n = (-\lambda_n)^2 + 4(-1) = \lambda_n^2 - 4 > 0$, we have a type 2 recurrence with $\theta_n = \frac{-\lambda_n - \sqrt{\Delta_n}}{2}$, $\theta_n' = \frac{-\lambda_n + \sqrt{\Delta_n}}{2}$,

$$\beta_n = \frac{\theta_n'}{\theta_n} \left(\frac{\theta_n' - \theta_n}{x_1^{(n)} - \theta_n} - 1 \right) = -1.$$

Thus, for all $n \geq 3$ and $1 \leq j \leq n$, we have

$$z_j^{(n)} = \theta_n + \frac{\sqrt{\Delta_n}}{1 - \left(\frac{\lambda_n + \sqrt{\Delta_n}}{\lambda_n - \sqrt{\Delta_n}} \right)^j}.$$

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

Moreover, since $z_1^{(n)} < \theta_n$, part (a) of Theorem 6.1 applies $z_j^{(n)} < \theta_n < 0$ for all j and n.

We note that (λ_n) is an increasing sequence as $T_{1,n,n}$ is a proper subgraph of $T_{1,n+1,n+1}$, and the index of a proper subgraph of a connected graph is always less than the index of the graph itself. Hence, as (λ_n) is bounded, there is $\lambda_0 \in \mathbb{R}$ such that $\lim_{n\to\infty} \lambda_n = \lambda_0$. Moreover,

$$\frac{\lambda_n + \sqrt{\Delta_n}}{\lambda_n - \sqrt{\Delta_n}} = \frac{(\lambda_n + \sqrt{\Delta_n})^2}{\lambda_n^2 - \Delta_n} \geq \frac{\lambda_n^2}{4} \geq \frac{\lambda_3^2}{4} > 1,$$

so that $\lim_{n\to\infty} \left(\frac{\lambda_n+\sqrt{\Delta_n}}{\lambda_n-\sqrt{\Delta_n}}\right)^n = \infty$, and therefore, $\lim_{n\to\infty} z_n^{(n)} = \lim_{n\to\infty} \theta_n$.

The value assigned by the algorithm to the root is $a(v_n) = -\lambda_n - \frac{1}{-\lambda_n} - \frac{2}{z_n^{(n)}}$, and it satisfies $a(v_n) = 0$ because λ_n is an eigenvalue of $T_{1,n,n}$. It follows that

$$-\frac{\lambda_0 + \sqrt{\lambda_0^2 - 4}}{2} = \lim_{n\to\infty} \theta_n = \lim_{n\to\infty} z_n^{(n)} = \frac{-2\lambda_0}{\lambda_0^2 - 1}.$$

This implies that $\lambda_0^4 - 4\lambda_0^2 - 1 = 0$, an equation with a single positive solution, namely $\lambda_0 = \sqrt{2+\sqrt{5}}$. □

We observe that we may generalize the concept of limit point of graphs for other matrices M associated with a graph G. We say that a real number γ is an M-limit point of the M-spectral radii of graphs if there exists a sequence of graphs $\{G_k \mid k \in \mathbb{N}\}$ such that

$$\lim_{k\to\infty} \rho_M(G_k) = \gamma,$$

where $\rho_M(G_i) \neq \rho_M(G_j),\ i \neq j$ and M is a type of matrix associated with a graph, such as the adjacency matrix, the Laplacian matrix, the signless Laplacian matrix, etc.

Example 6.2, which involves H-shape trees, may also be viewed in connection with limit points, particularly with Hoffman-Smith (HS) limit points. These are limit points that appear in the context of *edge subdivisions* and their effect on the indices of two matrices associated with graphs, namely the adjacency index $\rho(G)$ and the signless Laplacian index $\kappa(G) = \rho(Q(G))$. To be precise, let $G = (V, E)$ be a graph and let $e = \{u, w\}$ be one of its edges. We say that $G' = (V', E')$ is the graph produced by subdividing e if $V' = V \cup \{v_e\}$, where v_e denotes a new vertex, and E' is obtained from E by replacing e by the edges $\{u, v_e\}$ and $\{v_e, w\}$. Let $\mathcal{S}(G)$ be the graph obtained from G by subdividing each of its edges exactly once, which is known as an application of the *subdivision operator* to G.

Hoffman and Smith [33] showed that, if G is a graph with maximum degree $\Delta(G) = \Delta$ and $\mathcal{S}^n(G) = (\mathcal{S} \circ \cdots \circ \mathcal{S})(G)$ is the graph obtained from G by applying the subdivision operator n times, then

$$\lim_{n\to\infty} \rho(\mathcal{S}^n(G)) = \frac{\Delta}{\sqrt{\Delta - 1}} \quad \text{and} \quad \lim_{n\to\infty} \kappa(\mathcal{S}^n(G)) = \frac{\Delta^2}{\Delta - 1}.$$

Because of this, the numbers $\frac{\Delta}{\sqrt{\Delta-1}}$ and $\frac{\Delta^2}{\Delta-1}$, where $\Delta \geq 2$ is a real number, are known as Hoffman-Smith limit points.

For the adjacency matrix, the graphs whose index does not exceed the HS-limit 2 (HS-limit point for $\Delta = 2$) are known as *Smith graphs.* The HS-limit points for $\Delta = 3$ are $\frac{3}{\sqrt{2}}$ for the adjacency matrix and $\frac{9}{2}$ for the signless Laplacian matrix. Recall that, in our discussion in connection with Example 6.2, we noted that,

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

for the index of an H-graph not to exceed $\frac{3}{\sqrt{2}}$, which is precisely the HS-limit for the adjacency matrix and $\Delta = 3$, it suffices to study the sign of the value $a(v)$ assigned to the root v in the application of the algorithm, as all values assigned to other vertices are necessarily negative. Since $\mathcal{S}(G)$ is an H-graph whenever G is an H-graph, the result of Hoffman and Smith implies that, for any H-graph G, $(\rho(\mathcal{S}^n(G)))$ is a sequence converging to $\frac{3}{\sqrt{2}}$. On the other hand, unlike the trees $T_{1,n,n}$ of Theorem 6.3, $\mathcal{S}^n(G)$ is *not* a subgraph of $\mathcal{S}^{n+1}(G)$, and therefore the argument in Theorem 6.3 used to justify that the sequence of spectral radii is monotone increasing does not apply (in fact, one may show that the sequence $(\rho(\mathcal{S}^n(G)))$ is not monotone).

A complete characterization of the graphs whose ρ-index is less than $\frac{3}{\sqrt{2}}$ or whose κ-index is less than $\frac{9}{2}$ is not known. This is in general a hard problem. For the κ index there is some progress in [4]. In particular, the set of H-trees whose κ-index is less than $\frac{9}{2}$ is completely determined using the technique shown in the example above.

7. Inertia and spectral characterization of cographs. The algorithm presented in Section 4 is from 2018 and is a generalization of an algorithm given in 2013 for locating eigenvalues in *threshold graphs*. This is an important hereditary subclass of cographs, namely the class of $\{P_4, C_4, 2K_2\}$-free graphs. It is shown in [53, Cor. 3.2] that the cotree of a threshold graph is a *caterpillar*, namely a tree that becomes a path when its leaves are removed.

It is interesting that in [40], it is shown that threshold graphs do not have eigenvalues in the interval (-1,0), which is somehow unexpected because the set of eigenvalues of graphs in general is *dense* in the real numbers: any nonempty open interval has an eigenvalue of some graph. In fact even more is true: any nonempty open interval of the real line contains an eigenvalue of a tree. This is a result due to J. Salez [52].

The fact that (-1,0) has no eigenvalue of cographs was proved in [44], and this may be used to characterize cographs in terms of their spectra, as observed by E. Ghorbani [31]. The remainder of this section is devoted to prove this characterization, which will done by using the algorithm `Diagonalize Cograph`. In the process, we obtain a formula for the inertia of cographs.

Theorem 7.1. *A graph G is a cograph if and only if no induced subgraph of G has an eigenvalue in $(-1, 0)$.*

We start with some technical lemmas.

Lemma 7.2. *Let G be a cograph with minimal cotree T_G. Let $\{v_k, v_j\}$ be a sibling pair processed by* `Diagonalize Cograph` *with parent w, for which $0 \le d_k, d_j < 1$, where d_i is the diagonal value of the vertex v_i.*

(*a*) *If $w = \oplus$, then d_k becomes permanently negative, and d_j is assigned a value in $(0, 1)$.*
(*b*) *If $w = \cup$, then d_k becomes permanently non-negative and d_j is assigned a value in $[0, 1)$.*

Proof. Item (a). By our assumptions, **subcase 1a** is executed. Hence

$$d_k \leftarrow \alpha + \beta - 2,$$
$$d_j \leftarrow \frac{\alpha\beta - 1}{\alpha + \beta - 2},$$

where α, β are the old values of d_k, d_j. Clearly $d_k < 0$. Now $d_j > 0$, as both numerator and denominator are negative. Since $(\alpha\beta - 1) - (\alpha + \beta - 2) = (\alpha - 1)(\beta - 1) > 0$, it follows that $d_j - 1 = \frac{(\alpha\beta-1)-(\alpha+\beta-2)}{\alpha+\beta-2} < 0$ or that $d_j < 1$.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

For item (b), we may have $d_k = d_j = 0$, and we execute **subcase 2b**; hence, d_k and d_j are assigned 0. In any other case, we execute **subcase 2a** and then

$$d_k \leftarrow \alpha + \beta,$$
$$d_j \leftarrow \frac{\alpha\beta}{\alpha+\beta},$$

where α, β are the old values of d_k, d_j. Clearly, $d_k > 0$. As for d_j, we observe that $d_j \geq 0$, since the denominator is positive and the numerator is non-negative. Since $(\alpha\beta) - (\alpha+\beta) = (\alpha-1)(\beta-1) - 1 < 0$, it follows that $d_j < 1$. ◻

We state a technical lemma with a slight change in the range of the initial values. The proof may be done in a similar way as the proof of the previous lemma.

Lemma 7.3. *Let G be a cograph with minimal cotree T_G. Let $\{v_k, v_j\}$ be a sibling pair processed by* `Diagonalize Cograph` *with parent w, for which $0 < d_k, d_j \leq 1$.*

(a) *If $w = \oplus$, then d_k becomes permanently nonpositive and $d_j \in (0, 1]$.*
(b) *If $w = \cup$, then d_k becomes permanently positive and $d_j \in (0, 1)$.*

Lemma 7.4. *Let G be a cograph with minimal cotree T_G and let $x \in \{0, 1\}$. Consider the execution of* `Diagonalize Cograph`*(T_G, x).*

(a) *If $x = 0$, then all diagonal values of vertices remaining on the cotree are in $[0, 1)$.*
(b) *If $x = 1$, then all diagonal values of vertices remaining on the cotree are in $(0, 1]$.*

Proof. As both proofs are similar, we prove item (a) and omit the proof of item (b). Initially, all values on T_G are zero. Suppose after m iterations of `Diagonalize Cograph,` all diagonal values of the cotree are in $[0, 1)$, and consider iteration $m+1$ with sibling pair $\{v_k, v_j\}$ and parent w. By assumption, $0 \leq d_k, d_l < 1$. If $w = \oplus$, then Lemma 7.2 (a) guarantees the vertex d_j remaining on the tree is assigned a value in $(0, 1)$. If $w = \cup$, Lemma 7.2 (b) guarantees $d_j \in [0, 1)$. This means after $m + 1$ iterations the desired property holds, completing the proof by induction. ◻

We recall that when a cograph G is disconnected, then its minimal cotree T_G has root of type $\cup$ and, in particular, it has $t \geq 0$ leaves representing the isolated vertices of G.

Remark 7.5. *Observe that if w is an internal node in T_G having t children, as the algorithm progresses bottom-up through the rules of Lemma 4.2, each internal child of w eventually is replaced by a leaf. Thus, when w is ready to be processed it will have t leaves as children. To simplify our analysis, without loss of generality, we can assume that all $t - 1$ sibling pairs are processed consecutively.*

Remark 7.6. *Consider a node w of type $\cup$ with $\ell = s + t$ children, where s children are internal (of type $\oplus$) and t children are leaves. In the execution of* `Diagonalize Cograph`$(T_G, 0)$*, from Lemmas 7.2 (a) and 7.4 (a), each $\oplus$ node will become positive. Thus when w is processed, it will have s leaves with positive values and t leaves with zero.*

Proposition 7.7. *Let G be a cograph with minimal cotree T_G having $t \geq 0$ isolated vertices. Assume T_G has k nodes of type $\oplus$ denoted by $\{w_1, w_2, \ldots, w_k\}$, and ℓ nodes of type $\cup$ denoted by $\{w_{k+1}, w_{k+2}, \ldots, w_{k+\ell}\}$. Let $q_i = s_i + t_i$ be the number of children of w_i, where s_i is the number internal children and t_i is the number of leaves of w_i, $i = 1, \ldots, k + \ell$. Let U be the set of $\cup$ nodes that have two or more leaves, that is, those in* *which $t_i \geq 2$. Then, the number $n_-(G)$ of negative eigenvalues of G, and the multiplicity $m_{A(G)}(0)$ of 0 as an eigenvalue of G, are given by*

$$n_-(G) = \sum_{i=1}^{k}(q_i - 1); \tag{7.39}$$

$$m_{A(G)}(0) = \sum_{w_i \in U}(t_i - 1) + \delta(t), \quad \text{where} \quad \delta(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t = 0 \end{cases}. \tag{7.40}$$

Proof. To prove equation 7.39, we execute `Diagonalize Cograph`$(T_G, 0)$, and consider any node w_i in T_G of type $\oplus$ with $q_i = s_i + t_i$ children, where t_i are leaves and s_i are internal nodes of type $\cup$. By Remark 7.7, when w_i is eligible to be processed, it will have q_i leaves as children and will process $q_i - 1$ sibling pairs. By Lemma 7.4 (a) all diagonal values on the cotree remain in $[0, 1)$. By Lemma 7.2 (a), each of the $q_i - 1$ sibling pairs will produce a permanent negative value before w_i is removed. This shows $n_-(G) \geq \sum_{i=1}^{k}(q_i - 1)$. However, Lemma 7.2 (b) shows that processing a sibling pair with parent $\cup$ can only produce non-negative permanent values. Hence the inequality is tight, completing the proof for $n_-(G)$.

To prove equation 7.40, let w_i be an internal node of type $\cup$ having $q_i = s_i + t_i$ children where s_i children are internal nodes and t_i are leaves. By Remark 7.6, when w_i is ready to be processed, it will have $q_i = s_i + t_i$ leaves, s_i positive values and t_i zeros. For every pair of zero values, we execute **subcase 2b** of algorithm `Diagonalize Cograph`. Each execution of **subcase 2b** produces a permanent zero value. This shows that w_i contributes with $t_i - 1$ zeros to the diagonal matrix, whenever $t_i > 1$. Additionally, we see from the proof of Lemma 7.2 (b), that the remaining permanent values produced while processing w_i are positive. This shows $m_G(0) \geq \sum_{w_i \in U}(t_i - 1)$. To obtain the value for $m_G(0)$, we first note that, by Lemma 7.2 (a), no zero can be created when processing a sibling pair whose parent is $\oplus$. If the cograph is connected, then we are done as $t = 0$ and the root is $\oplus$.

If G is disconnected, then the root of T_G has type $\cup$ with $s + t$ children, with $t \geq 0$ leaves and $s \geq 0$ internal nodes of type $\oplus$. We can assume $s > 0$, for otherwise the graph is a collection of isolated vertices. We claim that exactly t additional zeros are created. Indeed, when the root is processed, all the t leaves have value zero and the s internal nodes became leaves with positive values by Lemma 7.2 (a). From Lemma 7.2 (b), we apply $s - 1$ times **subcase 2a**, creating $s - 1$ permanent positive values and leaving a positive value in the cotree. Then, $t - 1$ zero permanent values are created through **subcase 2b**. The last iteration, when **subcase 2a** is applied, creates an additional zero. ◻

Corollary 7.8. *The inertia of the n vertex cograph G, having the same parameters of Proposition 7.7, is given by the triple*

$$(p, q, r),$$

where $q = \sum_{i=1}^{k}(q_i - 1)$, $\quad r = \sum_{w_i \in U}(t_i - 1) + \delta(t)$ and $p = n - q - r$.

Proposition 7.9. *Let G be a cograph with minimal cotree T_G. Assume T_G has k nodes of type $\oplus$ denoted by $\{w_1, w_2, \ldots, w_k\}$, and ℓ nodes of type $\cup$ denoted by $\{w_{k+1}, w_{k+2}, \ldots, w_{k+\ell}\}$. Let $q_i = s_i + t_i$ be the number of children of w_i, where s_i is the number internal nodes and t_i is the number of leaves of w_i, $i = 1, \ldots, k + \ell$. Let J be the set of $\oplus$ node having two or more leafs, that is, those for which $t_i \geq 2$. Then*

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

the number $m_{A(G)}(-1,\infty)$ of eigenvalues greater than -1, and the multiplicity of -1 as an eigenvalue of G are given by

$$m_{A(G)}(-1,\infty) = 1 + \sum_{i=k+1}^{k+\ell} (q_i - 1), \tag{7.41}$$

$$m_{A(G)}(-1) = \sum_{w_i \in J} (t_i - 1). \tag{7.42}$$

Proof. To show equation (7.42), we count the number of zeroes in the diagonal matrix outputted by the execution of `Diagonalize Cograph`$(T_G, 1)$. Let w_i be a node of type $\oplus$ with $q_i = s_i + t_i$ children, $t_i \geq 0$ leaves and $s_i \geq 0$ internal nodes of type $\cup$. As in the previous result, when w_i is processed, all the s_i internal nodes became leaves with positive values smaller than 1, by Lemma 7.3 (b). Whenever $t_i \geq 2$, we notice that $t_i - 1$ permanent zero values are produced, as we are in the case **subcase 1b** of the algorithm. Moreover, any other pair processed by node w_i produces no zero value by Lemma 7.2 (a). This means that $m_{A(G)}(-1) \geq \sum_{w_i \in J}(t_i - 1)$. To see that equality holds, we first notice that, by Lemma 7.4(b), all values that remain to be processed are in $(0, 1]$. Since these values will be processed by $\cup$ nodes, we see by Lemma 7.3 (b) that no additional zeroes are produced. This proves equation (7.42).

To show equation (7.41), we count the number of positive values in the diagonal matrix outputted by the execution of `Diagonalize Cograph`$(T_G, 1)$. Consider now a node w_i of type $\cup$ with $q_i = s_i + t_i$ children, where t_i are leaves and s_i are nodes of type $\oplus$, which eventually become leaves with positive values (by Lemma 7.4 (b)). Then, all values of the q_i leaves are in (0,1], and by Lemma 7.3 (b), $q_i - 1$ positive values are outputted, and, hence, this shows that $m_{A(G)}(-1,\infty) \geq \sum_{i=k+1}^{k+\ell}(q_i - 1)$.

We claim that the final iteration of the algorithm always outputs an additional positive value. To see this, let $q = s + t \geq 2$ be the number of leaves of the root. We can assume that $s > 0$ for otherwise G is either the complete graph (if the root is $\oplus$) or a collection of isolates (if the root is $\cup$). Let d_j and d_k be the last two values in the cotree. By Lemma 7.4 (b), $d_j, d_k \in (0, 1]$. If the root is $\cup$, then the final iteration executes **subcase 2a**, which outputs two positive values. One of them is already accounted for in the in the formula $\sum_{i=k+1}^{k+\ell}(q_i - 1)$.

If the root is $\oplus$, since $s > 0$ we may assume $d_j \in (0, 1)$ by Lemma 7.3 (b). Now the value d_k is in $(0, 1)$, when $t = 0$, while $d_k = 1$, when $t > 0$. In any case, the last iteration executes **subcase 1a**, which produces a positive value and a negative value. Therefore we have $m_{A(G)}(-1,\infty) \geq 1 + \sum_{i=k+1}^{k+\ell}(q_i - 1)$.

To show equality, we observe that, by Lemma 7.3 (a), no positive value is outputted by processing two siblings of a node $\oplus$ with values in (0,1]. ∎

We remark that the formulas for $m_{A(G)}(0)$ and $m_{A(G)}(-1)$ appear in Bıyıkoğlu, Simić and Stanić [7, Cor. 3.2]. We presented here an alternate proof using our algorithm.

Proof. (of Theorem 7.1) Let G be a cograph. The number $m_{A(G)}(-1, 0)$ of eigenvalues in (-1,0) is given by

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

$$\begin{aligned}
m_{A(G)}(-1,0) &= m_{A(G)}(-1,\infty) - m_{A(G)}[0,\infty)\\
&= m_{A(G)}(-1,\infty) - n_+(G) - m_{A(G)}(0)\\
&= m_{A(G)}(-1,\infty) - (n - n_-(G) - m_{A(G)}(0)) - m_{A(G)}(0)\\
&= m_{A(G)}(-1,\infty) - n + n_-(G).
\end{aligned}$$

Now, by equation (7.39) and equation (7.41), it follows that

$$\begin{aligned}
m_{A(G)}(-1,0) &= 1 + \sum_{i=k+1}^{k+\ell}(q_i-1) - n + \sum_{i=1}^{k}(q_i-1)\\
&= 1 + \sum_{i=1}^{k+\ell}(q_i-1) - n = 0.
\end{aligned}$$

As the number of leaves in the minimal cotree is $n = 1 + \sum_{i=1}^{k+\ell}(q_i-1)$, the last equality follows. This shows that no cograph has eigenvalue in (-1,0). As cographs are an hereditary class (see Chapter 4) of graphs, any induced subgraph of G is also a cograph and hence any induced subgraph of G has no eigenvalue in (-1,0).

Conversely, if G is not a cograph, then G has an induced P_4 whose eigenvalues are approximately $\pm 1.61803, \mp 0.61803$, hence P_4 has an eigenvalue in (-1,0). ◻

8. Other applications.

In this section, we describe briefly some applications of the eigenvalue location algorithms presented in the previous sections.

We first notice that our algorithms can be adapted easily for computing the characteristic polynomial $p_A(\lambda)$ of a symmetric matrix A. One way of doing this is by carrying out the computations prescribed by our procedures in a symbolic way. More precisely, as $p_A(\lambda) = \det(A - \lambda I)$, if we treat λ as an indeterminate in our diagonalization of $A - \lambda I$, each diagonal value will be a rational function of λ. Since the resulting matrix is diagonal and was obtained from $A - \lambda I$ by a sequence of congruence operations as in (1.3), which do not affect the determinant, the product of these diagonal elements gives the required determinant.

Example 8.1. *As an illustration, we compute the characteristic polynomial of the Laplacian matrix of the tree given in Fig. 3 or, equivalently, we compute*

$$\begin{vmatrix}
1-\lambda & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 1-\lambda & 1 & 0 & 0 & 0 & 0\\
1 & 1 & 3-\lambda & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 1-\lambda & 1 & 0 & 0\\
0 & 0 & 0 & 1 & 2-\lambda & 0 & 1\\
0 & 0 & 1 & 0 & 0 & 2-\lambda & 1\\
0 & 0 & 0 & 0 & 1 & 1 & 2-\lambda
\end{vmatrix}.$$

The application of the algorithm is depicted in Fig. 26. The initial value d_i assigned to each vertex appears on the first tree of Fig. 26. Recall that the algorithm does not process the leaves. The leaves' parents are assigned the values $b_1 := 3 - \lambda - \frac{2}{1-\lambda} = \frac{\lambda^2-4\,\lambda+1}{1-\lambda}$ and $b_2 := 2 - \lambda - \frac{1}{1-\lambda} = \frac{\lambda^2-3\,\lambda+1}{1-\lambda}$, respectively. Next, the algorithm computes the value $c_1 := 2 - \lambda - \frac{1}{b1} = -\frac{\lambda^3-6\,\lambda^2+8\,\lambda-1}{\lambda^2-4\,\lambda+1}$. Finally, it computes the value at the root:

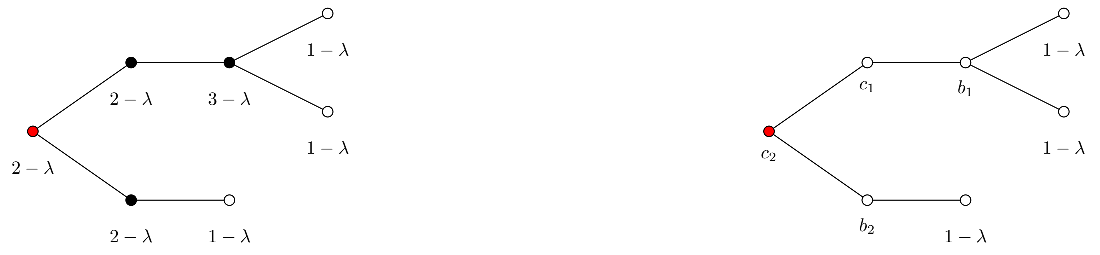


FIGURE 26. *Application of the tree algorithm.*

$c_2 := 2-\lambda-\frac{1}{b_2}-\frac{1}{c_1} = -\frac{\lambda\,(-1+\lambda)\left(\lambda^4-10\,\lambda^3+33\,\lambda^2-38\ ,\lambda+7\right)}{(\lambda^3-6\,\lambda^2+8\,\lambda-1)(\lambda^2-3\,\lambda+1)}$. *These final diagonal values are depicted on the second tree of Fig. 26.*

The characteristic polynomial of the graph is given by the product

$$(1-\lambda)^3\cdot b_1\cdot b_2\cdot c_1\cdot c_2 = \lambda\,(1-\lambda)^2\left(\lambda^4-10\,\lambda^3+33\,\lambda^2-38\,\lambda+7\right).$$

Suppose that A is a symmetric matrix of order n to which one of our algorithms apply (and that the treewidth or clique-width is bounded by an absolute constant if the corresponding algorithm is applied). We may use our algorithms to compute the characteristic polynomial $p_A(\lambda)$ of A in time $O(n^2)$ using interpolation. We sketch the approach. We first generate $n+1$ arbitrary distinct values $x_j\in\mathbb{R}, j=0,\ldots,n$. For each x_j, we apply the algorithm to $A-x_jI$, to obtain a diagonal $d=(d_1,\ldots,d_n)$. Computing $y_j=\prod_{i=1}^n d_i$ gives a point (x_j,y_j) such that $p_A(x_j)=y_j$. We then apply interpolation to the points $(x_0,y_0),\ldots,(x_n,y_n)$, to obtain $p_A(\lambda)$. Each pair (x_i,y_i) may be computed in time $O(n)$, so that it takes time $O(n^2)$ to generate the set of points. Finally, it is well known that interpolation of such a set of points can be done in time $O(n^2)$ and space $O(n)$. We refer to [39] for details.

The symbolic approach illustrated in Example 8.1 has been useful for matrices whose underlying graphs have a simple structure. In particular, it has been used for computing characteristic polynomials of trees of diameter 3 in [57]. It was the main tool for ordering these trees by their algebraic connectivity and by their Laplacian energy. For a graph G with Laplacian spectrum $0=\mu_1\leq\cdots\leq\mu_n$ and average degree $\overline{d}$, the *algebraic connectivity* is given by μ_2 and the *Laplacian energy* is given by

$$LE(G)=\sum_{i=1}^{n}|\mu_i-\overline{d}|.$$

In [26], the computation of characteristic polynomials of trees with small diameter was an important ingredient in an intricate proof of the following upper bound on the parameter $S_k(T)$, the sum of the k largest Laplacian eigenvalues of any n-vertex tree T:

$$S_k(T)\leq n-2+2k-\frac{2k-2}{n}. \tag{8.43}$$

This bound has led to the proof of the nice conjecture that the star $K_{1,n-1}$ is the tree with largest Laplacian energy.

In [27], the bound given by Equation (8.43) was improved further for trees having more than 5 vertices and diameter greater than 3. This enabled the authors to rank the trees having largest Laplacian energy.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

More precisely, given a positive integer n, they found a class $\mathcal{C}_n$ of cardinality approximately $\sqrt{n}$ such that if $T \in \mathcal{C}_n$ and $T' \notin \mathcal{C}_n$ are n-vertex trees, then $LE(T) > LE(T')$.

It is worth noticing that good upper bounds for $S_k(T)$ may be applied to prove the validity of the celebrated Brouwer's conjecture for other classes of graphs. We recall that Brouwer's conjecture states that for any graph G and $k \in \{1, \dots, n\}$,

$$S_k(G) \leq E(G) + \binom{k+1}{2}.$$

Indeed, as an illustration, Wang, Huang, and Liu [58] used inequality (8.43) to prove Brouwer's conjecture for unicyclic graphs, for bicyclic graphs (if $k \neq 3$) and for tricyclic graphs (with some restrictions). In the papers [27, 58], the authors prove that Brouwer's conjecture holds for k sufficiently large.

A graph G is said to be *M-integral* if the spectrum of the matrix M associated with G is composed only by integers. The eigenvalue location algorithms in this paper have been used to study integral graphs in several ways. In the original paper for trees [37], the authors studied the A-integrality of caterpillars, that is, the integrality with respect to the adjacency matrix. Patuzzi et. al [49] used the algorithm as a tool to study the A-integrality of trees of small diameter and determined infinite families of trees with integral spectral radius. In [18], Braga et. al introduced a location algorithm for unicyclic graph and, as an application, studied the A-integrality of closed caterpillars, which are obtained from cycles by attaching pendant paths. Using the cograph algorithm, Allem and Tura [1] studied the A-integrality of cographs. In a recent paper [3], Belardo et al. developed an algorithm for locating eigenvalue of signed graphs and found families of integral signed graphs.

Two nonisomorphic graphs are *M-cospectral* if the matrices M associated with them have the same spectrum. Two graphs that are not M-cospectral graphs are said to be *M-equienergetic* if they have the same M-energy. The algorithm for locating eigenvalues in threshold graphs, originally developed in [38], was used in [40] to find families of A-cospectral threshold graphs as well as families of A-equienergetic and integral threshold graphs

An application of eigenvalue location to the normalized Laplacian matrix of trees was studied in [15]. Among other things, the authors studied the multiplicity of normalized Laplacian eigenvalues of trees with small diameter. Their main result is the characterization of the trees that have 4 or 5 distinct normalized Laplacian eigenvalues. They also show that, for fixed diameter, these trees are determined by their normalized Laplacian spectrum.

The analytical approach made possible via the recurrence relation technique of Section 6 seems to be a powerful tool to study the spectral radius. A careful analysis of these recurrence relations (with different initial conditions) has been used to compare spectral radii in classes of trees, enabling one to solve combinatorial/algebraic problems where traditional techniques had failed.

In [46], for example, it was proved that, for a fixed n, all starlike trees on n vertices have distinct spectral radii. Moreover, the authors found that the order induced on starlike trees by their spectral radii coincides with the lexicographic order of their path lengths. In [5], the authors ordered infinite families of trees by their spectral radius. More precisely, the authors considered trees with spectral radius in the real interval $(2, \sqrt{2+\sqrt{5}})$ and their ordering with respect to the spectral radius. By doing so, the authors proved a conjecture about the ordering of trees with smallest spectral radius, open since 2002, and obtained the

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

first eight trees of even order with largest spectral radius. It is worth mentioning that the interval under consideration is very small and traditional tools from numerical analysis have failed.

Eigenvalue location seems also to be useful for studying the eigenvalues distribution of matrices. For the Laplacian matrix L (of a graph G) and an interval I, let $m_G(I)$ be the number of eigenvalues of L which lie in I. It is well known that $m_G[0, n] = n$, that is, for any Laplacian eigenvalue μ of G, $0 \leq \mu \leq n$. For an arbitrary n-vertex tree T, it was conjectured in [57] that

$$m_T[0, \overline{d}_n) \leq \left\lceil \frac{n}{2} \right\rceil, \tag{8.44}$$

where $\overline{d}_n = 2 - 2/n$ is the average degree, meaning that at most half of the Laplacian eigenvalues are larger than its average. A progress toward proving the validity of the conjecture was made in [17], where it was shown that $m_T(0, 2] \leq \lceil \frac{n}{2} \rceil$. The recurrence relation given by Equation (6.31), used in an ingenious way, was the main tool to prove that the conjecture is true in [36]. The conjecture was independently proved by Sin [55] also using a proof based on the eigenvalue location algorithm.

Acknowledgments. This work is partially supported by MATH-AMSUD under project GSA, Brazilian team financed by CAPES under project 88881.694479/2022-01, and by CNPq under grant number 408180/2023-4. C. Hoppen acknowledges the support of FAPERGS 19/2551-0001727-8 and CNPq (Proj. 315132/2021-3). V. Trevisan acknowledges partial support of CNPq grant 310827/2020-5 and FAPERGS grant PqG 17/2551-0001. We acknowledge our students Leonardo Consorte Veit, whose Master's thesis provided the figures of Section 3 and Nelson de Assis, whose Master's thesis gave Example 5.3. We thank an anonymous referee whose thoughtful recommendations improved the original version of this survey.

REFERENCES

[1] L.E. Allem and F. Tura. Integral cographs. *Discret. Appl. Math.*, 283:153–167, 2020.
[2] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebr. Discret. Meth.*, 8(2):277–284, 1987.
[3] F. Belardo, M. Brunetti, and V. Trevisan. Locating eigenvalues of unbalanced unicyclic signed graphs. *Appl. Math. Computat.*, 400:126082, 2021.
[4] F. Belardo, M. Brunetti, V. Trevisan, and J. Wang. On quipus whose signless laplacian index does not exceed 4.5. *J. Algebr. Comb.*, 55(4):1199–1223, 2022.
[5] F. Belardo, E.R. Oliveira, and V. Trevisan. Spectral ordering of trees with small index. *Linear Algebra Appl.*, 575:250–272, 2019.
[6] U. Bertelè and F. Brioschi. *Nonserial Dynamic Programming.* Elsevier, 1972.
[7] T. Bıyıkoğlu, S.K. Simić, and Z. Stanić. Some notes on spectra of cographs. *Ars Combin.*, 100:421–434, 2011.
[8] H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1–2):1–21, 1993.
[9] H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
[10] H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1):1–45, 1998.
[11] H.L. Bodlaender. *Treewidth of Graphs.* Springer New York, New York, NY, 2255–2257, 2016.
[12] H.L. Bodlaender and A.M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.
[13] H.L. Bodlaender and A.M.C.A. Koster. Treewidth computations i. upper bounds. *Inform. Computat.*, 208(3):259–275, 2010.
[14] H.L. Bodlaender and A.M.C.A. Koster. Treewidth computations ii. lower bounds. *Inform. Computat.*, 209(7):1103–1119, 2011.
[15] R.O. Braga, R.R. Del-Vecchio, V.M. Rodrigues, and V. Trevisan. Trees with 4 or 5 distinct normalized laplacian eigenvalues. *Linear Algebra Appl.*, 471:615–635, 2015.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

[16] R.O. Braga and V.M. Rodrigues. Locating eigenvalues of perturbed Laplacian matrices of trees. *TEMA (São Carlos) Brazilian Soc. Appl. Math. Comp.*, 18(3):479–491, 2017.
[17] R.O. Braga, V.M. Rodrigues, and V. Trevisan. On the distribution of Laplacian eigenvalues of trees. *Discrete Math.*, 313(21):2382–2389, 2013.
[18] R.O. Braga, V.M. Rodrigues, and V. Trevisan. Locating eigenvalues of unicyclic graphs. *Appl. Anal. Discrete Math.*, 11:273–298, 2017.
[19] S.M. Cioabă, E.R. van Dam, J.H. Koolen, and J.-H. Lee. Asymptotic results on the spectral radius and the diameter of graphs. *Linear Algebra Appl.*, 432(2):722–737, 2010.
[20] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Computat.*, 9(3):251–280, 1990. Computational algebraic complexity editorial.
[21] D.G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed, and U. Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discret. Appl. Math.*, 160(6):834–865, 2012. Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009.
[22] D.G. Corneil, H. Lerchs, and L.S. Burlingham. Complement reducible graphs. *Discrete Appl. Math.*, 3(3):163–174, 1981.
[23] D.G. Corneil, Y. Perl, and L.K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985.
[24] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.
[25] M.R. Fellows, F.A. Rosamond, U. Rotics, and S. Szeider. Clique-width minimization is NP-hard (extended abstract). In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 354–362. ACM, New York, 2006.
[26] E. Fritscher, C. Hoppen, I. Rocha, and V. Trevisan. On the sum of the Laplacian eigenvalues of a tree. *Linear Algebra Appl.*, 435:371–399, 2011.
[27] E. Fritscher, C. Hoppen, I. Rocha, and V. Trevisan. Characterizing trees with large laplacian energy. *Linear Algebra Appl.*, 442:20–49, 2014. Special Issue on Spectral Graph Theory on the occasion of the Latin Ibero-American Spectral Graph Theory Workshop (Rio de Janeiro, 27-28 September 2012).
[28] M. Fürer, C. Hoppen, D.P. Jacobs, and V. Trevisan. Locating the eigenvalues for graphs of small clique-width. In M.A. Bender, M. Farach-Colton, and M.A. Mosteiro (editors), *LATIN 2018: Theoretical Informatics.* Springer International Publishing, Cham, 2018, 475–489.
[29] M. Fürer, C. Hoppen, D.P. Jacobs, and V. Trevisan. Eigenvalue location in graphs of small clique-width. *Linear Algebra Appl.*, 560:56–85, 2019.
[30] M. Fürer, C. Hoppen, and V. Trevisan. Efficient diagonalization of symmetric matrices associated with graphs of small treewidth, 2021.
[31] E. Ghorbani. Spectral properties of cographs and p5-free graphs. *Linear Multilinear Algebra*, 67(8):1701–1710, 2019.
[32] R. Halin. S-functions for graphs. *J. Geom.*, 8(1):171–186, 1976.
[33] A.J. Hoffman and J.H. Smith. On the spectral radii of topologiacally equivalent graphs. In M. Fiedler (editor), *Recent Advanced in Graph Theory.* Academia, Prague, 273–281, 1975.
[34] C. Hoppen, D.P. Jacobs, and V. Trevisan. *Locating Eigenvalues in Graphs: Algorithms and Applications.* Springer Nature, 2022.
[35] D.P. Jacobs, C.M.S. Machado, and V. Trevisan. An o(n^2) algorithm for the characteristic polynomial of a tree. *J. Comb. Math. Comb. Comput.*, 54:213–221, 2005.
[36] D.P. Jacobs, E. Oliveira, and V. Trevisan. Most Laplacian eigenvalues of a tree are small. *J. Comb. Theory, B*, 146:1–33, 2021.
[37] D.P. Jacobs and V. Trevisan. Locating the eigenvalues of trees. *Linear Algebra Appl.*, 434(1):81–88, 2011.
[38] D.P. Jacobs, V. Trevisan, and F. Tura. Eigenvalue location in threshold graphs. *Linear Algebra Appl.*, 439(10):2762–2773, 2013.
[39] D.P. Jacobs, V. Trevisan, and F. Tura. Computing the characteristic polynomial of threshold graphs. *J. Graph Algor. Appl.*, 18(5):709–719, 2014.
[40] D.P. Jacobs, V. Trevisan, and F. Tura. Eigenvalues and energy in threshold graphs. *Linear Algebra Appl.*, 465:412–425, 2015.
[41] D.P. Jacobs, V. Trevisan, and F. Tura. Eigenvalue location in cographs. *Discrete Applied Mathematics*, 245:220–235, 2018.

Electronic Journal of Linear Algebra, ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 40, pp. 81-139, January 2024.

[42] T. Kloks. *Treewidth: Computations and Approximations*, vol. 842. *Lecture Notes in Computer Science.* Springer Verlag, 1994.

[43] M. Lepović and I. Gutman. Some spectral properties of starlike trees. In *Bull. Acad. Serbe Sci. Arts*, vol. 26. *Cl. Sci. Math. Nat.,Sci. Math.* Académie Serbe des Sciences et des Arts, 107–113, 2001.

[44] A. Mohammadian and V. Trevisan. Some spectral properties of cographs. *Discret. Math.*, 339(4):1261–1264, 2016.

[45] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[46] E.R. Oliveira, D. Stevanović, and V. Trevisan. Spectral radius ordering of starlike trees. *Linear Multilinear Algebra*, 68(5):991–1000, 2020.

[47] E.R. Oliveira and V. Trevisan. Applications of rational difference equations to spectral graph theory. *J. Diff. Equ. Appl.*, 27(7):1024–1051, 2021.

[48] S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory B*, 96(4):514–528, 2006.

[49] L. Patuzzi, M.A.A. de Freitas, and R.R. Del-Vecchio. Indices for special classes of trees. *Linear Algebra Appl.*, 442:106–114, 2014. Special Issue on Spectral Graph Theory on the occasion of the Latin Ibero-American Spectral Graph Theory Workshop (Rio de Janeiro, 27-28 September 2012).

[50] N. Robertson and P.D. Seymour. Graph minors I. excluding a forest. *J. Comb. Theory B*, 35:39–61, 1983.

[51] N. Robertson and P.D. Seymour. Graph minors II. Algorithmic aspects of tree-width. *J. Algor.*, 7(3):309–322, 1986.

[52] J. Salez. Every totally real algebraic integer is a tree eigenvalue. *J. Comb. Theory B*, 111:249–256, 2015.

[53] I. Sciriha and S. Farrugia. On the spectrum of threshold graphs. *ISRN Disc. Math.*, 2011:1–29, 2011.

[54] J.B. Shearer. On the distribution of the maximum eigenvalue of graphs. *Linear Algebra Appl.*, 114-115:17–20, 1989. Special Issue Dedicated to Alan J. Hoffman.

[55] C. Sin. On the number of laplacian eigenvalues of trees less than the average degree. *Discret. Math.*, 343(10):111986, 2020.

[56] V. Strassen. Relative bilinear complexity and matrix multiplication. *J. für die reine und angewandte Mathematik* 1987(375-376):406–443, 1987.

[57] V. Trevisan, J.B. Carvalho, R.R. Del Vecchio, and C.T.M. Vinagre. Laplacian energy of diameter 3 trees. *Appl. Math. Lett.*, 24(6):918–923, 2011.

[58] S. Wang, Y. Huang, and B. Liu. On a conjecture for the sum of laplacian eigenvalues. *Math. Comput. Model.*, 56(3):60–68, 2012.

[59] E. Wanke. k-NLC graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2):251–266, 1994.

[60] R. Woo and A. Neumaier. On graphs whose spectral radius is bounded by $\frac{3}{2}\sqrt{2}$. *Graphs Comb.*, 23:713–726, 2007.