EIGENVALUE LOCATION IN GRAPHS OF SMALL CLIQUE-WIDTH

MARTIN FÜRER, CARLOS HOPPEN, DAVID P. JACOBS, AND VILMAR TREVISAN

ABSTRACT. Finding a diagonal matrix congruent to $A - cI$ for constants c, where A is the adjacency matrix of a graph G allows us to quickly tell the number of eigenvalues in a given interval. If G has clique-width k and a corresponding k-expression is known, then diagonalization can be done in time $O(poly(k)n)$ where n is the order of G.

Keywords: adjacency matrix, eigenvalue, small clique-width.

1. INTRODUCTION

Throughout this paper we use standard terminology for graph theory and linear algebra. The main concern of spectral graph theory is to determine properties of a graph through the eigenvalues of matrices associated with it. Even if we restrict ourselves to the adjacency and the Laplacian matrices, eigenvalues and eigenvectors have been particularly useful for isomorphism testing and embedding graphs in the plane [2], for graph partitioning and clustering [22], in the study of random walks on graphs [5, 21] and in the geometric description of data sets [6], just to mention a few examples. An obvious step in any such application is to calculate the spectrum of the input graph, or at least to accurately estimate a subset of its eigenvalues. In fact, the distribution of eigenvalues of graphs in a given class of graphs generated by a given random graph model has been studied intensively (see [11, 19, 23] and the references therein).

We say that an algorithm *locates eigenvalues for a class* $\mathcal{C}$ if, for any graph $G \in \mathcal{C}$ and any real interval I, it finds the number of eigenvalues of G in the interval I. In recent years, efficient algorithms have been developed for the location of eigenvalues in trees [14], threshold graphs [15] (also called *nested split graphs*), and chain graphs [1]. A rich class of graphs which contain threshold graphs are the graphs with no induced subgraph isomorphic to P_4, which are often called P_4-*free* graphs or *cographs*. Eigenvalue location in cographs and threshold graphs has been widely studied [28, 29, 30, 3, 16, 25].

It turns out that there is a strong connection between eigenvalue location and congruence of matrices, which we now describe. Two matrices R and S are *congruent*, which we write $R \cong S$, if there exists a nonsingular matrix P for which $R = P^T S P$. Let G be a graph with adjacency matrix A, and consider real numbers $c < d$. If we can construct a diagonal matrix $D_c \cong B = A - cI$, then Sylvester's Law of Inertia [24, p. 568] implies that the number n_1 of eigenvalues of A greater than c equals the number positive entries in D_c. (Similarly, the number of eigenvalues equal to c, or less than c, are given by the number of zero diagonal entries, or by the number of negative entries in D_c, respectively.) Hence, the number n_2 of positive entries in a diagonal matrix $D_d \cong A - dI$ is the number of eigenvalues of A greater than d. Thus $n_1 - n_2$ is the

number of eigenvalues in $(c, d]$. This is why we want to design a fast algorithm to find a diagonal matrix that is congruent to $A - cI$.

Tura and two of the current authors [16] designed such a diagonalization algorithm for cographs. This algorithm runs in linear time and performs congruence operations on the matrix $A - cI$ using the *cotree* representation of G (see [7] for more information about cographs and the terminology associated with them). The algorithm works bottom-up on the cotree, and at each stage it diagonalizes the rows and columns associated with either one or two vertices (so-called *siblings*), and then removes the corresponding leaves from the tree. Here, we generalize this approach to arbitrary graphs, using a parse tree representation that is closely connected to the hierarchical decomposition of graphs known as *clique-width*. In spite of this similarity, the new algorithm requires several new ingredients. Indeed, unlike the cograph algorithm, the new algorithm does not diagonalize a given number of vertices at each stage, and needs to pass information up the tree (in a very compact way).

Clique-width is a powerful concept which was introduced in 2000 by Courcelle and Olariu [10], and turns out to be interesting for algorithmic purposes. Its main motivation was to extend the well-known concept of tree-width due to Robertson and Seymour [27] to denser graphs. In general, graph widths have been used to design algorithms for NP-complete or even harder problems that are efficient on graphs of bounded width. (Interested readers are referred to [4, 26], and the references therein. See also [8] for relations between tree-width and clique-width.)

A *k-expression* is an expression formed from atoms $i(v)$, two unary operations $\eta_{i,j}$ and $\rho_{i \to j}$, and a binary operation $\oplus$ as follows.

- $i(v)$ creates a vertex v with label i, where i is from the set $[k] = \{1, \ldots, k\}$.
- $\eta_{i,j}$ creates edges (not already present) between every vertex with label i and every vertex with label j for $i \neq j$.
- $\rho_{i \to j}$ changes all labels i to j.
- $\oplus$ produces the disjoint union of two labeled graphs.

Finally, the graph generated by a k-expression is obtained by deleting the labels. The *clique-width* $cw(G)$ of a graph G is the smallest k such that the graph can be defined by a k-expression [9, 10].

Any graph can be constructed in this way, provided that k is large enough. For instance, cographs are exactly the graphs for which $cw(G) \leq 2$, and one can show that $cw(T) \leq 3$ for any tree T. See [20] for a discussion of the clique-width of many classical classes of graphs. Computing the clique-width is NP-hard [12]. Thus, one usually assumes that a graph is given together with a k-expression.

The purpose of this paper is to give an $O(poly(k)n)$ time diagonalization algorithm for graphs having clique-width k. Note that the adjacency matrices of graphs with clique-width k often have $\Omega(n^2)$ nonzero entries, and that the clique-width may be a small constant even if other parameters, such as the tree-width, are linear in n. While there is a strong connection between tree-width and Gaussian elimination, the main application area for graph widths has been the design of efficient algorithms for NP-complete or even harder problems. The goal there is to find problems fixed parameter tractable (FPT), by providing an algorithm with a running time of $O(f(k)n^c)$, for a constant c and an arbitrary computable function f. Typically, f is at least exponential, but, for small values of k, such algorithms are often very practical.

Here, we return to a polynomial time solvable problem. Nevertheless, the parameterized complexity view is very useful. For bounded clique-width, we turn a cubic

time solution into a linear time solution, a drastic improvement for graphs of small clique-width. Despite being inspired by the cograph approach of [16], the extension is not at all straightforward. In fact, a property that is particularly crucial to us is that subgraphs generated by subexpressions are induced subgraphs, which does not hold for k-expressions, where an edge creating operation applied after a join of G and H typically introduces edges within G and within H. To deal with this, we introduce a new graph decomposition that may be easily obtained from a k-expression (and translated back into a k-expression).

Our approach is reminiscent of a parameter closely related to clique-width, the lesser known *NLC-width*, due to Wanke [31], and initiated by node label controlled (NLC) graph grammars [17, 18]. Graphs of NLC-width at most k are defined by NLC k-expressions. These expressions contain the operators $i(v)$ and $\rho_{i \to j}$ for vertex creation and relabeling. But new edges are created in *combination* with the join operation, using a binary operation $\oplus_S$, where $S \subseteq [k] \times [k]$. When $G \oplus_S H$ is applied, then, for each $(i, j) \in S$, edges are introduced between vertices labeled i in G and vertices labeled j in H. This has the effect that a subgraph generated by a subexpression is always an induced subgraph, a property important to us.

In representing graphs, we will actually use a minor but simplifying modification of NLC-width, which we call *slick clique-width* and is much more convenient than clique-width for our purposes. Here a single operator performs the join, edge creation and relabelling, resulting in an expression whose parse tree is the simplest among graph representations. It also has the property that subgraphs generated by subexpressions are induced subgraphs.

The remainder of this paper is organized as follows. In Section 2 we define slick k-expressions and slick clique-width, denoted $scw(G)$. We also prove that $scw(G) \leq cw(G) \leq 2\,scw(G)$ and observe that there are linear-time transformations to translate a k-expression to a slick k-expression, and a slick k-expression to a $2k$-expression. In Section 3 we describe our $O(poly(k)n)$ time diagonalization algorithm for graphs of slick clique-width k, and in Section 4 illustrate the algorithm by computing the inertia of a certain graph of order seven. Some applications of this algorithm can be found in Section 5, while concluding remarks appear in Section 6.

2. Slick Clique-Width

In the following definition a single operator is used for performing the union, creating edges and relabeling. A *slick k-expression* is an expression formed from atoms $i(v)$ and a binary operation $\oplus_{S,L,R}$, where L, R are functions from $[k]$ to $[k]$ and S is a binary relation on $[k]$, as follows.

(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 by the following operations. 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 i of the *right component* H is replaced by $R(i)$.

Two slick clique-width expressions are said to be *equivalent* if they produce the same labeled graph. Finally, the *graph generated by a slick clique-width expression* is obtained by deleting the labels of the labeled graph produced by it. The *slick clique-width*

$scw(G)$ of a graph G is the smallest k such that the graph can be defined by a slick k-expression.

Note that cographs are precisely the graphs with slick clique-width equal to one. Indeed, recall that G is a cograph if and only if either G is a single vertex or the union $G_1 \cup G_2$ or join $G_1 \otimes G_2$ of cographs G_1 and G_2 (see [3]). On the other hand, when there is a single label available, the function L and R are trivial identities, so that $\oplus_{S,L,R}$ either creates a disjoint union (if $S = \emptyset$) or adds all possible edges with ends in the two operands (if $S = \{(1,1)\}$). The graph G of Figure 1 may be constructed with the following slick 2-expression:

$$\left[1(f) \oplus_{\{(1,2)\},id,id} 2(g)\right] \oplus_{\{(1,2)\},\{2\to1\},\{2\to1\}}$$
$$\left[\{1(e)\} \oplus_{\{(1,2)\},id,\{1\to2\}} \left\{\left((1(c) \oplus_{\{(1,2)\},id,id} 2(d))\right) \oplus_{\{(2,2)\},id,id} \left((1(a) \oplus_{\{(1,2)\},id,id} 2(b)))\right)\right\}\right]$$

or, representing the triples that define each operation by letters,

$$\left[1(f)\ \mathbf{B}\ 2(g)\right]\ \mathbf{A}\ \left[\{1(e)\}\ \mathbf{C}\ \{((1(c)\ \mathbf{E}\ 2(d)))\ \mathbf{D}\ ((1(a)\ \mathbf{F}\ 2(b)))\}\right].$$

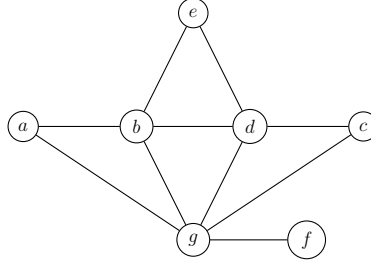Since G is not a cograph (it contains an induced copy of P_4), we know that $scw(G) = 2$.

FIGURE 1. Graph with $k = 2$.

Note that the definition of a slick k-expression implies that edges can only be placed between different components, so if two vertices are in the same component after some steps in the above construction, but are not adjacent, they will never become adjacent. As shown in Figure 2, a slick k-expression can also be represented as a parse tree T where the leaves contain the operators $i(v)$ and the internal nodes contain the $\oplus_{S,L,R}$ operations. Two vertices v and w are adjacent if and only if their least common ancestor $\oplus_{S,L,R}$ in T connects them, similar to the cotree representation for cographs in [3].

We can define the *depth $d(r)$* of a slick k-expression r recursively. The expression $i(v)$ has depth 0, and $d(A \oplus_{S,L,R} B) = 1 + \max\{d(A), d(B)\}$. This is equivalent to the depth of the parse tree for r. In a similar way, we can define the depth of a k-expression.

The next result shows that the concepts of clique-width and slick clique-width are closely related.

Theorem 1. *If G is a graph then $scw(G) \leq cw(G) \leq 2\,scw(G)$.*

Proof. This follows from Lemma 2 and Lemma 3 below. □

Lemma 1. *For any slick k-expression s, we can write $\eta_{i,j}(s)$ as a slick k-expression s' where $d(s) = d(s')$.*

Proof. We apply induction on $d(s)$. When $d(s) = 0$, then s has the form $i'(v)$ and $\eta_{i,j}(s)$ is equivalent to $i'(v)$. Now let $d(s) > 0$, where $s = s_1 \oplus_{S,L,R} s_2$, for slick expressions s_1
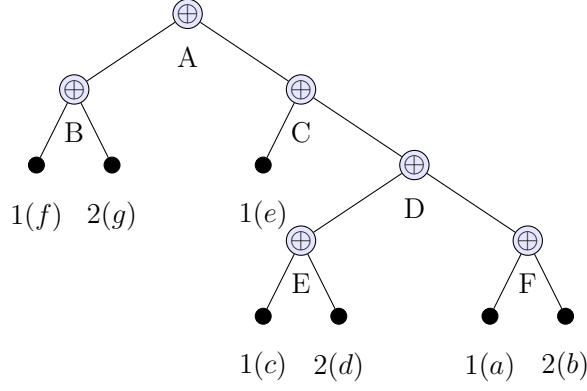
FIGURE 2. A parse tree.

and s_2, and assume the statement holds for slick k-expressions of smaller depth. We must construct edges between all pairs of vertices whose labels are i and j *after* L and R are applied. The edges created by $\eta_{i,j}$ are either *between* left and right components or *within* a component. To produce the edges between components, let

$$S' = \{(i', j')|L(i') = i \text{ and } R(j') = j\} \cup \{(j', i')|L(j') = j \text{ and } R(i') = i\}$$

To produce edges within components let

$$S_{LL} = \{(i', j')|L(i') = i \text{ and } L(j') = j\}$$
$$S_{RR} = \{(i', j')|R(i') = i \text{ and } R(j') = j\}$$

If S_{LL} and S_{RR} are both empty, then $\eta_{i,j}$ must not place any new edges within the components. Hence we can complete the induction with the slick expression

$$s_1 \oplus_{S \cup S', L, R} s_2. \tag{1}$$

However, suppose

$$S_{LL} = \{(i'_1, j'_1), (i'_2, j'_2), \ldots, (i'_k, j'_k)\}$$

is not empty. To place edges into the left side of (1) we construct the composition

$$\eta_{i'_1, j'_1} \ \eta_{i'_2, j'_2} \ \cdots \eta_{i'_k, j'_k} \ s_1 \tag{2}$$

Since $d(s_1) < d(s)$, by repeated use of the induction assumption, the expression (2) can be written as a single slick clique-width expression s'_1 where $d(s'_1) = d(s_1)$. A similar argument shows that if $S_{RR} \neq \emptyset$, correct edges can be placed into into right side of (1) as a single slick clique-width expression s'_2, where $d(s'_2) = d(s_2)$. Hence we can write $\eta_{i,j}(s)$ as

$$s'_1 \oplus_{S \cup S', L, R} s'_2,$$

whose depth is that of s. This completes the induction. $\square$

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

Proof. Consider a k-expression r composed of the operators $i(v)$, $\eta_{i,j}$, $\rho_{i \to j}$ and $\oplus$. It suffices to construct an *equivalent* slick k-expression, that is, one that produces the same labelled graph. We use induction on $d(r)$, the base case being clear. Let $d(r) > 0$, and assume all k-expressions of smaller depths have equivalent slick k-expressions. There are three cases. Either $r = \rho_{i \to j}(r_1)$, $r = r_1 \oplus r_2$, or $r = \eta_{i,j}(r_1)$, where the r_i are k-expressions.

If $r = \rho_{i \to j}(r_1)$, then by induction r_1 has an equivalent slick k-expression s_1. If s_1 has the form $i(v)$ we may rewrite r as the slick k-expression $j(v)$. If $s_1 = a \oplus_{S,L,R} b$, we define the functions

$$L'(t) = \begin{cases} L(t) & \text{if } L(t) \neq i \\ j & \text{otherwise} \end{cases} \qquad R'(t) = \begin{cases} R(t) & \text{if } R(t) \neq i \\ j & \text{otherwise} \end{cases}$$

Then r is equivalent to the slick k-expression $a \oplus_{S,L',R'} b$.

Suppose $r = r_1 \oplus r_2$. Then by induction, there are slick k-expressions s_1 and s_2 equivalent to r_1 and r_2 respectively. Then r must be equivalent to the slick k-expression $s_1 \oplus_{\emptyset,id,id} s_2$.

Finally, assume $r = \eta_{i,j}(r_1)$. By induction r_1 is equivalent to a slick k-expression s, so r is equivalent to $\eta_{i,j}(s)$. By Lemma 1, this is equivalent to some slick k-expression, and we are done. $\qquad\square$

We note that Lemma 2 (with Lemma 1) gives an algorithm for translating a k-expression to a slick k-expression, and in fact is linear-time for constant k. Similarly, Lemma 3 below provides a linear-time algorithm for translating a slick k-expression to a $2k$-expression.

Lemma 3. *If G is a graph then $cw(G) \leq 2\, scw(G)$.*

Proof. Suppose $k = scw(G)$, and s is a slick k-expression for G. It suffices to construct an equivalent $2k$-expression r. We will show this by induction on the depth of s. This is clear when $s = i(v)$. Otherwise, let $s = s_1 \oplus_{S,L,R} s_2$, for slick k-expressions s_1 and s_2. By the induction hypothesis, we can assume that each slick k-expression s_i has an equivalent $2k$-expression r_i, producing the same labeled graph. So we have

$$s = r_1 \oplus_{S,L,R} r_2.$$

Note that, even though r_1 and r_2 may involve labels between $k+1$ and $2k$, they produce labeled graphs with labels in $[k]$. To complete the induction we translate the behavior of $\oplus_{S,L,R}$ into the operators $\rho_{i \to j}$, $\eta_{i,j}$ and $\oplus$. We relabel the vertices on the left side mapping each label $i \in [k]$ to $i + k$, and then form the union. Suppressing parentheses, we obtain a subexpression

$$w = (\rho_{1 \to k+1}\ \rho_{2 \to k+2}\ \cdots\ \rho_{k \to 2k}\ r_1) \oplus r_2. \tag{3}$$

To obtain edges, for each $(i,j) \in S$ we apply the operators $\eta_{i+k,j}$ to w, obtaining a new expression w'. The relabeling in (3) ensures that new edges are placed *only* between the left and right sides. Depending on the functions L and R, we finally relabel the left and right sides back to their desired values in $[k]$.

We need to relabel according to the function $f : [2k] \to [k]$ defined by

$$f(i) = \begin{cases} L(i - k) & \text{if } i > k \\ R(i) & \text{if } i \leq k. \end{cases}$$

It is tempting to simply apply all the relabelings $\rho_{i \to f(i)}$, but one operation could modify earlier operations. Instead, we do this relabeling in three rounds.

First, we reduce the number of labels to at most k, using the $2k$-expression

$$w'' = \rho_{1 \to g(1)} \cdots \rho_{2k \to g(2k)} w'$$

where

$$g(i) = \max\{j \mid f(i) = f(j)\}.$$

Note that the set $\{j \mid f(i) = f(j)\}$ is not empty, as it contains the label i. Also note that the order of operations and $i \le g(i)$ ensure that no $\rho_{i \to g(i)}$ changes an earlier one.

Now, we move the set of current labels $I = \{i_1, \ldots, i_q\}$ with $i_1 < i_2 < \cdots < i_q$ and $q \le k$ up into the interval $[2k - q + 1, 2k]$ using the $2k$-expression

$$w''' = \rho_{i_1 \to h(i_1)} \cdots \rho_{i_q \to h(i_q)} w''$$

where $h : I \to [2k-q+1, 2k]$ is given by $h(i_j) = 2k-q+j$. Since $i_j \le h(i_j)$, no operation can affect earlier ones. Note that the break down into the previous two rounds is just to simplify the description. These relabelings could have been combined into one round.

In the third round, we choose the proper new labels with the $2k$-expression

$$w'''' = \rho_{2k-q+1 \to f(h^{-1}(2k-q+1))} \, \rho_{2k-q+2 \to f(h^{-1}(2k-q+2))} \cdots \rho_{2k \to f(h^{-1}(2k))} \, w'''$$

As each operation maps a label in $[k + 1, 2k]$ to one in $[k]$, no relabeling can injure others. After the third round, f is achieved. Finally, to obtain r, we delete all the $\rho_{i \to i}$ operations, or declare them as having no effect. $\qquad\square$

The cograph diagonalization algorithm in [16] exploited the fact that in any cograph of order $n \ge 2$, there exist two vertices u and v for which either $N(u) = N(v)$ or $N[u] = N[v]$, so-called siblings. 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 row and column of v is annihilated except in one off diagonal position. The following analog is crucial to our algorithm.

Remark 1. *Let T_G be a parse tree for a graph G with adjacency matrix A, and Q a node in T_G. If two vertices u and v have the same label at Q, then their rows (columns) will agree outside of the matrix for the subtree rooted at Q.*

Two matrices are congruent if one can obtain the other by a sequence of *pairs* of elementary operations, each pair consisting of a row operation followed by the *same* column operation. In our algorithm we only use congruence operations that permute rows and columns or add a multiple of a row and column to another row and column respectively. To achieve linear-time we must operate on a sparse representation of the graph, rather than the adjacency matrix. Moreover, we must represent the undiagonalized portion of the matrix with constant space.

3. THE ALGORITHM

We now describe our diagonalization algorithm. Let $G = (V, E)$ be a graph on $n = |V|$ vertices, and adjacency matrix A, given by a slick k-expression Q_G. We wish to find a diagonal matrix congruent to $B = A - cI_n$. The expression Q_G defines its parse tree T having $2n - 1$ nodes, a rooted binary tree whose nodes are the subexpressions of Q_G, and whose edges are the pairs of nodes $\{Q_\ell, Q\}$ and $\{Q_r, Q\}$ for some subexpression Q of Q_G with $Q = Q_\ell \oplus_{L,R,S} Q_r$ for some L, R, and S. The algorithm `diagonalize` works bottom-up in the parse tree T of the slick k-expression Q_G. In particular, our implementation does a post-order traversal and operates on Q after both children have been processed.

A node Q in the tree produces a data structure that we call a *k-box* b_Q, namely a 4-tuple $[k', k'', M, \Lambda]$ where k' and k'' are nonnegative integers bounded above by k, M is a symmetric matrix of order $m \le 2k$ and Λ is a vector of whose m components are labels in $\{1, \ldots, k\}$. In a high level description, the algorithm traverses the parse tree from the leaves to the root so that, at each node of the parse tree, the algorithm either initializes a box, or it combines the boxes produced by the node's children into its own

box, transmitting it to its parent. While processing the node, the algorithm may also produce diagonal elements of a matrix congruent to $A - cI_n$. These diagonal elements are appended to a global array as they are produced.

```
diagonalize(G,c)
input:  the parse tree T of slick k-expression Q_G for G, a scalar c
output:  diagonal entries in D ≅ B = A(G) − cI
Order the 2n − 1 vertices of T as Q_1, Q_2, ..., Q_{2n−1} = Q_G in post order
for t from  1 to 2n − 1 do
    if is-leaf(Q_t) then construct b_{Q_t}=LeafBox(Q_t, c)
    else if Q_t = Q_ℓ ⊕_{S,L,R} Q_r
        then construct b_{Q_t}=CombineBoxes(b_{Q_ℓ}, b_{Q_r})
DiagonalizeBox(b_{Q_{2n−1}})
```

FIGURE 3. High level description of the algorithm `diagonalize`.

In the remainder of this section, we shall describe each of these stages in detail and prove that the algorithm `diagonalize` yields the desired output.

At each node Q, our 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.

Recall that any subexpression Q of Q_G is also a node in the parse tree T of Q_G. For a node Q of T, let n_Q be the order of the graph $G(Q)$ generated by the expression Q. Note that $G(Q)$ is the subgraph of G induced by the vertices of $G(Q)$. Let A_Q be the adjacency matrix of $G(Q)$, let I_Q be the $n_Q \times n_Q$ identity matrix, and $B_Q = A_Q - cI_Q$. For simplicity, an entry vw of such a matrix M will always refer to the vertices of the graph G indexing the corresponding rows in M. (This avoids the need of keeping track of permutations of rows and columns when describing the algorithm.)

The goal at each node Q_t is to construct, by means of congruence operations, a matrix $\widetilde{B_{Q_t}}$ which is diagonal except for at most $2k$ rows and columns, having the form:

$$\widetilde{B_{Q_t}} = \begin{pmatrix} d_1 & & 0 & & & \\ & \ddots & & & 0 & \\ 0 & & d_\ell & & & \\ & & & M^{(0)} & & M^{(1)} \\ & & 0 & & & \\ & & & M^{(1)T} & & M^{(2)} \end{pmatrix}. \quad (4)$$

Here, $M^{(0)}$ and $M^{(2)}$ are square matrices of dimensions $k' \times k'$ and $k'' \times k''$ respectively, $M^{(1)}$ is a $k' \times k''$ matrix with $0 \leq k' \leq k'' \leq k$. Note that k' can be zero in which case

we regard $M^{(0)}$ as empty. The criterion that defines the partition of the matrix

$$
M_{Q_t} = \begin{array}{|c|c|}
\hline
M^{(0)} & M^{(1)} \\
\hline
M^{(1)T} & M^{(2)} \\
\hline
\end{array}
\tag{5}
$$

may be visualized in equation (6).

As it turns out, the matrix $\widetilde{B}_{Q_t}$ will be congruent to B_{Q_t}. Moreover, it is crucial to control its relationship to the matrix $\widetilde{B}_{Q_t}^+$. This is the $n \times n$ matrix that *would be* obtained by performing the same congruence operations on B that are actually performed up to stage t, that is, on $\widetilde{B}_{Q_1}, \widetilde{B}_{Q_2}, \ldots, \widetilde{B}_{Q_t}$. Thus the invariant $B \cong \widetilde{B}_{Q_t}^+$ is maintained. Equation (6) illustrates how the matrix $\widetilde{B}_{Q_t}$ fits into $\widetilde{B}_{Q_t}^+$, which is never actually computed by the algorithm. We assume, for clarity, that the rows and columns corresponding to vertices that have already been diagonalized or lie in $G(Q_t)$ appear first.

$$
\widetilde{B}_{Q_t}^+ =
\left(
\begin{array}{ccc}
D & 0 & 0 \\[2ex]
0 & \begin{array}{|c|c|}\hline M^{(0)} & M^{(1)} \\ \hline M^{(1)T} & M^{(2)} \\ \hline\end{array} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \\ \beta_1^1 & \cdots & \beta_1^s \\ \vdots & & \vdots \\ \beta_{k''}^1 & \cdots & \beta_{k''}^s \end{matrix} \\[6ex]
0 & \begin{matrix} 0 & \cdots & 0\, \beta_1^1 & \cdots & \beta_{k''}^1 \\ \vdots & & \vdots\ \vdots & & \vdots \\ 0 & \cdots & 0\, \beta_1^s & \cdots & \beta_{k''}^s \end{matrix} & M'
\end{array}
\right)
\tag{6}
$$

The right side of (6) shows how the remainder of the large matrix B is transformed as a side effect of operating on the small submatrix M_{Q_t}. The diagonal matrix D represents *all* diagonalized elements produced up until stage t in the algorithm. Also the k' rows and columns of $M^{(0)}$ extend with zero vectors, defining the boundary 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 later (see Lemma 4). It is important to observe 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 stages. The submatrix M' in (6) contains all undiagonalized rows $w \notin G(Q_t)$, may include vertices in $M_{Q_{t'}}$ for $t' \neq t$, and is empty after the last iteration of the algorithm.

It will be convenient to define the k' rows in $M^{(0)}$ as having *type-i*, and to define the k'' rows of $M^{(2)}$ as having *type-ii*. It is useful to understand that a row begins as a type-ii row, then becomes a type-i row, and finally becomes diagonalized.

As mentioned earlier, diagonal elements in diagonalized rows and columns are not transmitted by nodes to their parents. But label information of vertices corresponding to the rows in M_{Q_t} must be maintained, and for simplicity, we will say that rows have labels. The important information is $M = M_{Q_t}$, these labels, and the integers k' and k'', which are stored in the k-box (or simply 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 $k'' \leq k$ in the matrix transmitted, each label appears in at most one type-ii row.

When the node Q_t is a leaf corresponding to a subexpression $i(v)$, $B_{Q_t} = [-c]$. Therefore the box contains a 1×1 matrix $M_{Q_t} = [-c]$, whose row is labeled i, $k' = 0$, and $k'' = 1$.

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

FIGURE 4. Procedure `LeafBox`.

To conclude the description of the algorithm `diagonalize`, we need to define the procedure `CombineBoxes`. Before describing this process, we state a lemma that summarizes facts about the algorithm that help establish its correctness. We observe that the rows (and columns) in the matrix $\widetilde{B_{Q_t}}$ that are not represented in M_{Q_t} have been diagonalized earlier in the process, so that $\widetilde{B_{Q_t}}$ contains three types of rows: type-i, type-ii and diagonalized. For simplicity, when referring to operations, we always mention the row operations, with the understanding that the corresponding column operations are also performed. We also identify vertices with their rows and columns.

Lemma 4. *Let $Q_1, Q_2, \ldots, Q_{2n-1} = Q_G$ be the nodes of the parse tree of the k-expression Q_G that defines G, listed in post order. Consider the matrices B, M_{Q_τ}, $\widetilde{B_{Q_\tau}}$ and $\widetilde{B_{Q_\tau}^+}$ defined above, where $\tau \in \{1, \ldots, 2n-1\}$, and let v and w be vertices of G.*

(a) *If row v is diagonalized at stage τ, then stage τ applied CombineBoxes. Moreover, if row v was diagonalized when processing $Q_{\tau'}$ for some $1 \leq \tau' < \tau$, then no components in this row are modified in stage τ.*

(b) *Suppose that row v has type-i in $M_{Q_{\tau'}}$, for some $\tau' \in \{1, \ldots, \tau\}$. Suppose also that w is a row such that, for all $j \in \{\tau', \ldots, \tau\}$, v and w are not simultaneously in $G(Q_j)$. Then the entry vw in $\widetilde{B_{Q_\tau}^+}$ is 0.*

(c) *Suppose that row v has type-ii in M_{Q_τ} and $w \notin G(Q_\tau)$. Suppose also that, for all $\tau' < \tau$, w has type-ii in $\widetilde{B_{Q_{\tau'}}}$ whenever $w \in G(Q_{\tau'})$. Then $\widetilde{B_{Q_\tau}^+}$ and B are equal in position vw.*

(d) *Suppose that, for some $\tau' \in \{0, \ldots, \tau - 1\}$, the rows v and w are both not in $G(Q_j)$ for all $\tau' < j \leq \tau$. Then $\widetilde{B_{Q_\tau}^+}$ and $\widetilde{B_{Q_{\tau'}}^+}$ are equal in position vw (where $\widetilde{B_{Q_0}^+} = B$).*

(e) *Suppose that row v has type-i in $M_{Q_{\tau'}}$, for some $\tau' \in \{1, \ldots, \tau - 1\}$. For all $\tau' < j \leq \tau$, if v is in M_{Q_j}, then it has type-i. That is, a type-i vertex cannot become type-ii again.*

Next we will show how `CombineBoxes` works, that is, we explain how a node produces its box from the boxes transmitted by its children. We shall assume inductively that Lemma 4 holds for all τ up to step $t-1$. This can be done because all items hold trivially at the beginning of the algorithm and cannot cease to hold after processing a leaf node, as no congruence operations are performed. Suppose that Q_t is a node with children Q_ℓ and Q_r, that is

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

Let M_{Q_ℓ} and M_{Q_r} denote the matrices in the boxes transmitted respectively, by Q_ℓ and Q_r, containing the undiagonalized rows in $\widetilde{B_{Q_\ell}}$ and $\widetilde{B_{Q_r}}$. The goal of Q_t is to combine M_{Q_ℓ} and M_{Q_r} into a single matrix of size at most $2k$ representing $\widetilde{B_{Q_t}}$. More precisely, based on M_{Q_ℓ} and M_{Q_r}, node Q_t first constructs the submatrix M of $\widetilde{B^+_{Q_{t-1}}}$ induced by the undiagonalized rows in $\widetilde{B_{Q_\ell}}$ and $\widetilde{B_{Q_r}}$. Recall that this cannot be done directly, as $\widetilde{B^+_{Q_{t-1}}}$ is never actually computed by the algorithm.

Lemma 5. *Suppose that $Q_t = Q_\ell \oplus_{S,L,R} Q_r$ and consider the submatrix M of $\widetilde{B^+_{Q_{t-1}}}$ induced by the undiagonalized rows in $\widetilde{B_{Q_\ell}}$ and $\widetilde{B_{Q_r}}$. The following hold:*

  (a) *If $s \in \{\ell, r\}$ and $v, w \in G(Q_s)$, then the entry vw in M is equal to the entry vw in M_{Q_s}.*
  (b) *If $v \in G(Q_\ell)$, $w \in G(Q_r)$, and at least one of v and w has type-i, then the entry vw in M is 0.*
  (c) *If $v \in G(Q_\ell)$, $w \in G(Q_r)$, and both have type-ii, then the entry vw in M is 1 if $(i,j) \in S$, where i is the type of v and j is the type of w, otherwise it is 0.*

Proof. Consider the entry vw in a submatrix M as in the statement of the lemma. By the definition of slick expression, $V(Q_\ell) \cap V(Q_r) = \emptyset$.

For part (a), assume that both v and w are on the same side, say $v, w \in M_{Q_\ell}$. Because the nodes of the parse tree are processed in post-order, v and w cannot be in $V(Q_j)$ for $\ell < j < t$. If $\ell < t-1$, Lemma 4(d) applied to $\tau' = \ell$ and $\tau = t-1$ implies that the entry vw is the same in $\widetilde{B^+_{Q_{t-1}}}$ and $\widetilde{B^+_{Q_\ell}}$. If $\ell = t-1$, this equality is trivial. By the definition of M_{Q_ℓ} in (4), the entries vw in $\widetilde{B^+_{Q_\ell}}$ and M_{Q_ℓ} are equal. The desired conclusion follows because M is a submatrix of $\widetilde{B^+_{Q_{t-1}}}$. Clearly, the analogous statement holds for $v, w \in M_{Q_r}$ and the matrices $\widetilde{B^+_{Q_{t-1}}}$ and $\widetilde{B^+_{Q_r}}$.

For part (b), fix $v \in M_{Q_\ell}$ and $w \in M_{Q_r}$. First assume that v has type-i in M_{Q_ℓ}. Note that $w \notin M_{Q_\ell}$ and v is not in $G(Q_j)$ for $j \in \{\ell+1, \ldots, t-1\}$. By Lemma 4(b) for $\tau' = \ell$ and $\tau = t-1$, the entry vw in $\widetilde{B^+_{Q_{t-1}}}$ is equal to 0. The same conclusion would be achieved if we assumed that w has type-i in M_{Q_r}.

We finally consider the case where v and w have type-ii in M_{Q_ℓ} and M_{Q_r}, respectively. Without loss of generality, assume that $\ell > r$. We know that w has type-ii in M_{Q_r} and that $w \notin G(Q_j)$ for all $j \in \{r+1, \ldots, t-1\}$. Moreover, Lemma 4(e) implies that, if $j \in \{1, \ldots, r-1\}$ and $w \in G(Q_j)$, then w has type-ii. We may apply Lemma 4(c) for $\tau = \ell$ and $\tau' = r$ to conclude that $\widetilde{B^+_{Q_\ell}}$ and B are equal in position vw. Part (d) of the same lemma, for $\tau' = \ell$ and $\tau = t-1$, implies that $\widetilde{B^+_{Q_{t-1}}}$ and B are equal in position vw. To conclude the proof, since $Q_t = Q_\ell \oplus_{S,L,R} Q_r$, we know that the entry vw in B

is 1 if $(i, j) \in S$, where i is the type of v and j is the type of w, otherwise it is 0, as required. □

Lemma 5 implies that, when processing Q_t, the matrix M may be constructed by first taking the disjoint union of the matrices transmitted by both 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 any element of S. Let F be the block of ones defining these edges. Then node Q_t starts with

$$
M_{Q_t} = \begin{pmatrix}
\begin{array}{|c|c|}
\hline
M_\ell^{(0)} & M_\ell^{(1)} \\
\hline
M_\ell^{(1)T} & M_\ell^{(2)} \\
\hline
\end{array} & \begin{matrix} 0 \\[1em] 0 \quad\quad F \end{matrix} \\[3em]
\begin{matrix} 0 \\[2em] 0 \quad\quad F^T \end{matrix} & \begin{array}{|c|c|}
\hline
M_r^{(0)} & M_r^{(1)} \\
\hline
M_r^{(1)T} & M_r^{(2)} \\
\hline
\end{array}
\end{pmatrix}
\tag{7}
$$

(Here, and in the remainder of the description of the procedure CombineBoxes, we abuse the notation slightly and use M_{Q_t} to refer to the matrix M obtained by merging the boxes transmitted by the children, even before it is a proper matrix for the box b_{Q_t}.) Next Q_t relabels the rows of M_{Q_ℓ} and M_{Q_r}, using the functions L and R, respectively. Note that the type-i rows from M_{Q_ℓ} and M_{Q_r} are still type-i in (7). Using permutations of rows and columns we combine the k'_ℓ type-i rows from M_{Q_ℓ} with the k'_r type-i rows of M_{Q_r}, and combine the k''_ℓ type-ii rows in M_{Q_ℓ} with the k''_r type-ii rows in M_{Q_r}, obtaining a matrix M_Q in the form of (5). In this matrix, $k' = k'_\ell + k'_r$ and $k'' = k''_\ell + k''_r$. By induction, we can assume that $k' \leq k''$, however we are not guaranteed that $k'' \leq k$.

The next three lemmas explain how to produce a box from M_{Q_t} in a way that Lemma 4 holds. This requires achieving $k'' \leq k$ without losing $k' \leq k''$ using congruence operations. This may also create new permanent elements in the diagonal submatrix D of $\widetilde{B^+_{Q_t}}$. The matrix is shrunk in three steps. First, $k'' \leq k$ is obtained by transforming type-ii rows into type-i rows. Next, if necessary, $M^{(0)}$ is made 0 or empty, creating diagonal elements. Finally, $k' \leq k''$ is obtained, creating additional diagonal elements.

Lemma 6. *If M_{Q_t} contains two type-ii rows with the same label, then we can produce a matrix that is congruent to it where k'' decreases by one and k' increases by one.*

Proof. Suppose two type-ii rows j and j' have the same label. Then by Remark 1 and Lemma 4 their rows and columns must agree outside of M_{Q_t}. Therefore in (6), $\beta_j^i = \beta_{j'}^i$ for all i. Performing the operations

$$
\begin{aligned}
R_{j'} &\leftarrow R_{j'} - R_j &\tag{8}\\
C_{j'} &\leftarrow C_{j'} - C_j &\tag{9}
\end{aligned}
$$

transforms the type-ii row j' into a type-i row, decreasing k'' by one, and increasing k' by one. □

We keep applying the operations (8) and (9) to *any pair of type-ii* rows with the same label. By the pigeon-hole principle, this will force $k'' \leq k$. However, at the end of this step we may have $k' > k''$. To reduce k' we first make $M^{(0)} = 0$.

Lemma 7. *Given a matrix M_{Q_t} as in* (5), *we can make $M^{(0)}$ zero or empty.*

Proof. Let R_i be the i-th row and C_i be the i-th column of M_{Q_t}. If some diagonal element m_{ii} of $M^{(0)}$ is nonzero, then subtract m_{ij}/m_{ii} times the i-th row of M_{Q_t} from the j-th row. Do likewise for the columns. In other words, operate on M_{Q_t} as follows for all $j \neq i$.

$$R_j \quad \leftarrow \quad R_j - \frac{m_{ij}}{m_{ii}} R_i,$$
$$C_j \quad \leftarrow \quad C_j - \frac{m_{ij}}{m_{ii}} C_i.$$

Note that due to the zero extension of the j-th row (see Lemma 4(b)), when the same operations are performed to produce $\widetilde{B^+_{Q_t}}$, all the work is restricted to the small matrix M_{Q_t}, and the *shape* of $\widetilde{B^+_{Q_t}}$ does not change. We have removed a row and a column from $M^{(0)}$, and $\widetilde{B^+_{Q_t}}$ has one more diagonal element m_{ii}.

If the diagonal of $M^{(0)}$ is 0, but an off-diagonal element m_{ij} is nonzero, then do the operations

$$R_j \quad \leftarrow \quad R_j + \frac{1}{2} R_i,$$
$$C_j \quad \leftarrow \quad C_j + \frac{1}{2} C_i,$$

followed by

$$R_i \quad \leftarrow \quad R_i - R_j,$$
$$C_i \quad \leftarrow \quad C_i - C_j.$$

The relevant entries of $M^{(0)}$ are modified as follows:

$$\begin{pmatrix} 0 & m_{ij} \\ m_{ij} & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & m_{ij} \\ m_{ij} & m_{ij} \end{pmatrix} \rightarrow \begin{pmatrix} -m_{ij} & 0 \\ 0 & m_{ij} \end{pmatrix}. \tag{10}$$

Now there are two nonzero diagonal elements whose rows can be annihilated as above, producing two more diagonalized rows. Again, note that due to the surrounding zero pattern, only M_{Q_t} is modified when these operations are performed, so that the portion of $\widetilde{B^+_{Q_t}}$ outside of M_{Q_t} remains unchanged. □

The operations in Lemma 7 create diagonal elements, decrease k' and make $M^{(0)}$ either 0 or empty. If we now have $k' \leq k''$ we achieve our goal. Otherwise the next lemma ensures this, creating $k' - k''$ zero diagonal elements.

Lemma 8. *Consider the matrix M_{Q_t} obtained after applying the operations of Lemma 7. Then we can make $k' \leq k''$.*

```
CombineBoxes(b_Qℓ,b_Qr)
  input:  two k-boxes b_Qℓ and b_Qr
  output:  a k-box b_Q
  form matrix M in (7);
  relabel rows with functions L and R;
  combine type-i rows (columns), combine type-ii rows (columns);
  ensure type-ii rows have distinct labels (Lemma 6);
  if k' > k'' then make M^(0) empty or zero (Lemma 7) and output diagonals;
  if k' > k'' then ensure k' ≤ k'' (Lemma 8) and output diagonals;
```

FIGURE 5.   Procedure `CombineBoxes`.

Proof. This is obvious if $M^{(0)}$ is empty, so assume that $M^{(0)} = 0$ and $k' > k''$. With simple row operations (subtracting multiples of one row from another one, and doing permutations of rows) $M^{(1)}$ can be made upper triangular. Doing the same operations on the columns of $M^{(1)T}$, the matrix $\widetilde{B^+_{Q_t}}$ remains symmetric. Since $k' > k''$ at least $k' - k''$ rows of $M^{(1)}$ have become 0. Thus the diagonal part of $\widetilde{B^+_{Q_t}}$ has grown by $k' - k''$ with zeros in the diagonal. The new k' is at most k'' meaning that $M^{(1)}$ is at most as high as wide. □

After applying the computations in Lemma 6, Lemma 7 and Lemma 8 if necessary, node Q_t is guaranteed that $k' \leq k'' \leq k$, and returns M_{Q_t}. This concludes the description of the procedure CombineBoxes, which appears in Figure 5. Note that all operations performed in the lemmas 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 the operation in Lemma 6, we will obtain $k'' = 1$. Applying the operations in Lemma 7 and Lemma 8 will make $M^{(0)}$ 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

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

and can be made fully diagonal using the transformations in Lemma 7. This is what we call `DiagonalizeBox` in Figure 3.

Remark 2. *When defining `diagonalize`, we added several features to simplify the description, but which are not crucial for the algorithm to work, and which would not necessarily be used in an efficient implementation of the algorithm.*

(a) *Since the matrix $\widetilde{B^+_Q}$ is not computed, one may easily write `diagonalize` 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 relabelled 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.

(d) *When performing* `CombineBoxes`*, and after applying Lemma 6, the algorithm uses Lemma 7 to make $M^{(0)}$ zero or empty if $k' > k''$. In fact, it is not necessary to get to this point, the procedure could have stopped at any point after he operations of Lemma 7 have produced enough diagonal elements so that $k' \leq k''$. Morever, if we still have $k' > k''$ after making $M^{(0)}$ zero or empty,* `CombineBoxes` *asks us to ensure that $k' \leq k''$ using operations of Lemma 8. We could have asked the algorithm to perform more operations to turn $M^{(1)}$ into a matrix with more structure (upper triangular, for example).*

To prove the correctness of the algorithm, we shall now see that Lemma 4 holds by induction on t.

Proof of Lemma 4. Assertion (e) is trivial, as LeafBox only produces rows of type-ii and no operation in CombineBoxes may turn a row of type-i into type-ii.

For assertion (a), the fact that diagonal elements are obtained through some application of CombineBoxes is obvious. Now, let v be a row that was diagonalized when processing $Q_{\tau'}$ for some $\tau' \leq t - 1$. First observe that any entry vw with $w \neq v$ is 0 in $\widetilde{B}^{+}_{Q_{t-1}}$. This is trivial for $\tau' = t - 1$ and holds by induction for $\tau' \leq t - 2$. Such an entry remains 0 in $\widetilde{B}^{+}_{Q_t}$ because all the operations in Lemmas 6, 7 and 8 involve undiagonalized rows and columns in $G(Q_t)$. Then, when a multiple of some column w' is added to w, it adds some multiple of vw' to vw, and both are 0.

Regarding assertion (b), suppose that row v has type-i in $M_{Q_{\tau'}}$, for some $\tau' \in \{1, \ldots, t\}$. Suppose also that w is a row such that, for all $j \in \{\tau', \ldots, t\}$, v and w are not simultaneously in $G(Q_j)$. We wish to show that entry vw in $\widetilde{B}^{+}_{Q_t}$ is 0. By part (a), this is immediate if v or w has been diagonalized up this stage, so assume that this is not the case. First suppose that $\tau' = t$, so that $w \notin G(Q_t)$. The first possibility is that the algorithm turned v to type-i at this stage, performing an operation of Lemma 6, which zeroed entry vw in $\widetilde{B}^{+}_{Q_t}$. After this, type-i rows are only added to one another (see Lemmas 7 and 8), so that such zero entries are not altered. The second option is that v was already of type-i in an earlier stage t'. Since w has not been diagonalized, the same assertion (b) must hold replacing $\tau' = t$ by $\tau' = t'$, and we would be in the case $\tau' < t$.

For $\tau' < t$, by induction the entry vw in $\widetilde{B}^{+}_{Q_{t-1}}$ is 0. If $v \in G(Q_t)$, then the fact that v has not been diagonalized and assertion (e) ensure that v has type-i in M_{Q_t}. Note that $w \notin G(Q_t)$. Adding the multiple of any row v' to v does not change entry vw, as v' must have type-i and therefore entry $v'w$ is zero by Lemma 6 (if v' turned to type-i at stage t), or by induction (if this happened at an earlier stage). Next assume that $v \notin G(Q_t)$. Then the only way to change entry vw is to add a multiple of some column $w' \in G(Q_t)$ to column w, and in particular $w', w \in G(Q_t)$. However, we may use the induction hypothesis for v and w' to conclude that the entry vw' in $\widetilde{B}^{+}_{Q_{t-1}}$ is also 0, and hence no change occurs.

Assertion (d) holds for the following reason. Assume that rows v and w are not in M_{Q_j} for all $\tau' < j \leq t$, where $0 \leq \tau' < t$. By induction, $\widetilde{B}^{+}_{Q_{\tau'}}$ and $\widetilde{B}^{+}_{Q_{t-1}}$ are equal in position vw (recall that $\widetilde{B}^{+}_{Q_0} = B$). Note that this position could be changed only if the multiple of some row was added to row v or the multiple of some column was added to

column w, but this cannot happen, as all the operations in Lemmas 6, 7 and 8 involve rows and columns in $G(Q_t)$, and neither v nor w is in M_{Q_t}.

To conclude, we prove assertion (c). Suppose that row v is of type-ii in M_{Q_t}, $w \notin G(Q_t)$, and w has type-ii in $\widetilde{B}_{Q_j}$ whenever $w \in G(Q_j)$ for some $j < t$. First observe that v and w cannot be both in some $G(Q_j)$ for $j < t$: by the ordering of the nodes of the parse tree, since $v \in M_{Q_t}$, when $j < t$, v belongs to precisely those $G(Q_j)$ on the path from its leaf to Q_t. Since $w \notin G(Q_t)$, it cannot belong to any of these $\widetilde{G}(Q_j)$.

We claim that B and $\widetilde{B}^+_{Q_{t-1}}$ are equal in position vw. Let τ' be the largest $j < t$ such that either v or w is in M_{Q_j} ($\tau' = 0$ if there is no such j). If $\tau' = 0$, the claim holds by substituting $\tau' = 0$ and $\tau = t - 1$ in part (d). If $\tau' > 0$, then either v or w is of type-ii in $M_{Q_{\tau'}}$ (w by hypothesis, v by part (e)). By induction, we see that B and $\widetilde{B}^+_{Q_{\tau'}}$ are equal in position vw. By the choice of τ' we may use assertion (d) for τ' and $\tau = t - 1$, and we see that $\widetilde{B}^+_{Q_{\tau'}}$ and $\widetilde{B}^+_{Q_{t-1}}$ are equal in position vw, which proves that B and $\widetilde{B}^+_{Q_{t-1}}$ and agree at position vw. To complete the proof, we show that $\widetilde{B}^+_{Q_{t-1}}$ and $\widetilde{B}^+_{Q_t}$ are equal in position vw. Note that no row (or multiple thereof) can be added to row w, as $w \notin G(Q_t)$. Moreover, if a multiple of a row u is added to row v, then, because v has type-ii in M_{Q_t} and by the description of `CombineBoxes`, u must have been of type-i at the moment when the operation has been performed. There are two possibilities: either u had type-i in $M_{Q_{\tau'}}$, for some $\tau' < t$, (and thus $\widetilde{B}^+_{Q_{t-1}}$ is 0 in position uw because of part (b) with $\tau = t - 1$) or u became of type-i at stage t (this happened due to an operation of Lemmas 6, and performing the corresponding operation on the matrix derived from $\widetilde{B}^+_{Q_{t-1}}$ turns the entry uw to 0. In both cases, adding a multiple of u to v does not change the entry vw in $\widetilde{B}^+_{Q_t}$, as required. □

Theorem 2. *Let G be a graph with adjacency matrix A, given by a slick k-expression Q_G with parse tree T, and let $c \in \mathbb{R}$. Algorithm `diagonalize` correctly outputs the diagonal elements of a diagonal matrix congruent to $B = A - cI$. Moreover, this is done in $O(k^3 n)$ operations.*

Proof. It is clear that the operations performed by `diagonalize` are congruence operations. Indeed, besides simultaneous permutations of rows and columns, the operations performed (in Lemma 6, Lemma 7 and Lemma 8) only add multiples of some row i to some other row j, followed by adding the same multiple of column i to column j. We now argue that the elements that the algorithm outputs are precisely the diagonal elements of a diagonal matrix congruent to $B = A - cI$. As above, let $\widetilde{B}^+_{Q_t}$ be the $n \times n$ matrix obtained from B by performing the same congruence operations on B that are actually performed on the corresponding rows of matrices M_{Q_t} up to stage t. Lemma 5 ensures that, at the beginning of each stage t, the algorithm always starts from a submatrix M of $\widetilde{B}^+_{Q_{t-1}}$. Also, Lemma 4 guarantees that equation (6) holds at the end of each stage: part (a) ensures that a row that was diagonalized at some stage cannot be modified at later stages. If $v \in G(Q_t)$ and $w \notin G(Q_t)$ (and has not been diagonalized), part (b) ensures (applying it twice with the roles of v and w interchanged) that the element β_i^j corresponding to the entry vw in $\widetilde{B}^+_{Q_t}$ is equal to 0 if v has type-i or if w has type-i in M_{Q_τ} for some $\tau < t$. Finally, part (c) ensures that in the remaining cases the entry vw is equal to the corresponding entry in B. We conclude that, at

each stage, the elements in the output of `CombineBoxes` (which are called *diagonals* in Figure 5) are indeed diagonal elements of the final matrix produced. The only nonzero elements of the final matrix that are not of this form are obtained by the application of `DiagonalizeBox`, which clearly outputs the final diagonal elements of a diagonal matrix D congruent to B.

A time bound of $O(k^3n)$ is straightforward, because the parse tree has $O(n)$ nodes ($2n-1$ to be precise), $n = |V|$ leaves, and $n-1$ internal nodes with 2 children each. At each node, since the algorithm acts on an $O(k) \times O(k)$ matrix, the number of row and column operations performed by Lemma 6, Lemma 7 and Lemma 8 is $O(k^2)$. Each such operation requires $O(k)$ sums and products. □

Now, we can apply Sylvester's law of inertia. Symmetric matrices over the reals have n real eigenvalues (with multiplicities). The inertia of a symmetric real matrix B is the triple (n_+, n_0, n_-) giving the number of eigenvalues of B that are positive, zero, and negative respectively. Sylvester's law says that congruent real symmetric matrices have the same inertia.

Corollary 1. *The number of eigenvalues of A in a real interval I can be computed in time $O(k^3n)$ for graphs of clique-width k.*

Proof. The eigenvalues of $B = A - cl$ are obtained by subtracting c from the eigenvalues of A. To compute the number of eigenvalues of A in a nonempty interval $I = [a, b]$, for instance, we run our algorithm with $c = a$ and $c = b$. From the output, we see the numbers of positive, zero, and negative diagonal elements. Let them be (n_+^a, n_0^a, n_-^a) and (n_+^b, n_0^b, n_-^b). Then, obviously the number of eigenvalues in (a, b) is $n_-^b - n_-^a - n_0^a$. Of course, there is nothing special about open (or bounded) intervals, and a similar argument would lead to the number of eigenvalues in any given real interval. □

It is worth stating that the whole algorithm is very fast, as there are no large constants hidden in the O-notation. In fact, it is not hard to modify Algorithm `diagonalize` to obtain an algorithm that requires only $O(k^2n)$ operations. Instead of performing the operations of Lemmas 6, 7 and 8 successively, we may keep $M^{(0)}$ equal to 0 or empty and $M^{(1)}$ as an upper triangular matrix, namely a matrix (a_{ij}) such that $a_{ij} = 0$ for every $i > j$. We insert row vectors (namely a type-ii vertex that becomes type-i) one at a time. At the end of the step, inserting such a vector may incur a cost of $O(k^2)$ in terms of $M^{(1)}$ operations. It does not matter that up to k row vectors are inserted in one node, because every row vector is inserted only once, and there are only n rows, one for each vertex in the graph.

Every time a new row v is inserted (and to simplify our description we assume that it becomes the first row of the new matrix), instead of performing the operations of Lemma 7 using the diagonal element, we choose the largest index j such that j has type-i and position vj is nonzero. We use this element to eliminate all the other nonzero elements of row (and column) v in $M^{(0)}$. The choice of j ensures that the operations performed do not destroy the upper triangular nature of $M^{(1)}$ (if we ignore row v). Once this is done, we replace the two remaining nonzero off-diagonal elements by diagonal elements as in Lemma 7 and either diagonalize new rows or get $M^{(0)} = 0$. Then we perform operations of Lemma 8 (and possibly exchange rows and columns) to turn $M^{(1)}$ into the upper triangular matrix mentioned in the previous paragraph. The cases where the diagonal element is the single such nonzero element in this row, and where all the elements of this row are zero may be treated similarly.

4. EXAMPLE

To see how the algorithm acts on a concrete example, we go back to the graph of Figure 1, whose parse tree is given in Figure 2. We apply the algorithm to the graph defined by this slick 2-expression for $c = 0$. Since $k = 2$, the boxes created by the leaves may be of the following two types:

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

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

The nodes B, E and F of the parse tree perform identical operations to produce

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

meaning that $M^{(0)}$ is empty, $k'' = 2$, and $M^{(2)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and the labels of its rows are 1 and 2, respectively. Up to this point, we did not need to apply Lemmas 6, 7 or 8.

When the algorithm starts processing node D, it uses the boxes produced by E and F together with $S = \{(2, 2)\}$ to form the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

It then uses row and column operations described in Lemma 6 (with (j', j) given by $(1, 3)$ and $(2, 4)$) to produce

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

This means that

$$k' = k'' = 2, \quad M^{(0)} = \begin{pmatrix} 0 & 2 \\ 2 & -2 \end{pmatrix}, \quad \text{and } M^{(2)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

However, there is no need to resort to Lemmas 7 or 8, as the relation $k' \le k'' \le k \le 2$ holds.

Next, node C receives boxes from a leaf and node D that, together with $S = \{(1, 2)\}$, produce the following matrix and vector of labels (recall that the entries created by S only affect type-ii rows and columns.):

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & -1 \\ 0 & 2 & -2 & -1 & 1 \\ 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 2 \end{pmatrix}.$$

Vertices of the right component are relabelled $1 \to 2$ and, as in the description of the algorithm, we exchange rows (and columns) 1 and 3 to keep type-i and type-ii rows

together. This leads to

$$M = \begin{pmatrix} -2 & 2 & 0 & -1 & 1 \\ 2 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 1 & 1 & 0 \end{pmatrix}, \text{ and labels } \begin{pmatrix} 2 \\ 2 \\ 1 \\ 2 \\ 2 \end{pmatrix}.$$

At this point $M^{(2)}$ is a 3×3 matrix and $k'' = 3 > k = 2$. We first apply Lemma 6 to $j' = 4$ and $j = 5$ in order to preserve the uniqueness of labels for type-ii. Moreover, as in the above description of the algorithm, we exchange rows 3 and 4 to keep types together:

$$M = \begin{pmatrix} -2 & 2 & -2 & 0 & 1 \\ 2 & 0 & 1 & 0 & -1 \\ -2 & 1 & -2 & -1 & 1 \\ 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & 1 & 1 & 0 \end{pmatrix} \text{ with labels } \begin{pmatrix} 2 \\ 2 \\ 1 \\ 2 \\ 2 \end{pmatrix}.$$

Now $k'' = 2 < 3 = k'$, so we need to reduce

$$M^{(0)} = \begin{pmatrix} -2 & 2 & -2 \\ 2 & 0 & 1 \\ -2 & 1 & -2 \end{pmatrix}.$$

The diagonal element of the first line of $M^{(0)}$ is -2, hence we can diagonalize that row as described in Lemma 7, so that M becomes

$$\begin{pmatrix} -2 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & \frac{1}{2} \end{pmatrix}$$

We have found the diagonal element -2, which is stored by the algorithm, and the first row (and column) are removed from M. We may repeat the argument for the next two rows of $M^{(0)}$, which leads to diagonal elements 2 and $-\frac{1}{2}$. At the end of the process, we have $k' = 0$, $k'' = 2$, and $M^{(2)} = \begin{pmatrix} 2 & 1 \\ 1 & \frac{1}{2} \end{pmatrix}$. Node C transmits the box

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

We finally process node A, which combines the boxes produced by B and C. Note that, in this matrix, we have $k' = 0$, $k'' = 4$ and, after adding the edge of $S = \{(1, 2)\}$ and then relabeling, we have

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 1 & 0 & 1 & \frac{1}{2} \end{pmatrix} \text{ with labels } \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Applying Lemma 6 for $(j', j) \in \{(1, 4), (2, 4), (3, 4)\}$, we obtain the following matrix, with $k' = 3$ and $k'' = 1$:

$$\frac{1}{2} \cdot \left(\begin{array}{ccc|c} -3 & 1 & -3 & 1 \\ 1 & 1 & -1 & -1 \\ \hdashline -3 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 \end{array} \right).$$

The algorithm now performs operations of Lemma 7, which leads to

$$\left(\begin{array}{cccc} -\frac{3}{2} & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{array} \right),$$

and the final diagonal values are $D = (-2, 2, -\frac{1}{2}, -\frac{3}{2}, \frac{2}{3}, \frac{1}{2}, 0)$. This means that the graph has three negative eigenvalues, three positive eigenvalues and 0 is an eigenvalue with multiplicity one. In fact, the actual spectrum may be approximated by

$$-1.9098; -1.6180; -1.2726; 0; 0.6180; 0.8692; 3.3132.$$

5. APPLICATIONS

Section 3 showed how to compute, in linear time, the number of eigenvalues of a graph in a given interval. Here we give some alternate ways to bound this number. If λ is an eigenvalue in a graph G, we denote its multiplicity with $m_G(\lambda)$. If G is represented by a slick k-expression Q we will also use $m_Q(\lambda)$ to denote the multiplicity of λ in G. We can deduce some simple properties about multiplicity by studying the behavior of algorithm `diagonalize`.

Remark 3. *If G has slick k-expression $Q = Q_\ell \oplus_{L,R,S} Q_r$ then for any eigenvalue λ*

$$m_G(\lambda) \le m_{Q_\ell}(\lambda) + m_{Q_r}(\lambda) + 4k$$

Proof. Note that $m_G(\lambda)$ is exactly the number of zero diagonal values produced by executing `diagonalize`(Q, λ). Assume the call to the left subtree Q_ℓ produces m_ℓ zeros, the call to the right subtree Q_r produces m_r zeros, and m zeros are produced from the matrix M in (7). Then we have $m_G(\lambda) = m_\ell + m_r + m$. Since $m_\ell \le m_{Q_\ell}(\lambda)$ and $m_r \le m_{Q_r}(\lambda)$ we have

$$m_G(\lambda) \le m_{Q_\ell}(\lambda) + m_{Q_r}(\lambda) + m$$

The result follows since m is precisely the nullity of M whose size is at most $4k \times 4k$. □

It follows that if $m_G(\lambda) > 4k$ then either $m_{Q_\ell}(\lambda) > 0$ or $m_{Q_r}(\lambda) > 0$. Another immediate consequence of Remark 3 is the following.

Remark 4. *If I is an interval in which neither Q_ℓ or Q_r contain eigenvalues and G has slick k-expression $Q = Q_\ell \oplus_{L,R,S} Q_r$, then for each $\lambda \in I$ we have $m_G(\lambda) \le 4k$.*

It is interesting to relate $m_G(\lambda)$ to the operations in `diagonalize`(Q, λ). During execution, only the operations in Lemmas 7 and 8 can create diagonal elements. A careful look at Lemma 7 reveals that if $M^{(0)} \ne 0$ and has rank r, then it creates r *nonzero* diagonal elements. When $M^{(0)} = 0$ and $k' > k''$, the operations in Lemma 8 produce $k' - k''$ zero diagonal elements. If z denote the number of zeros created in Lemma 8 (in the entire algorithm), since the algorithm may return a 2×2 matrix at the end of its execution, we must have $z \le m_G(\lambda) \le z + 2$.

We can also bound the *total* number of eigenvalues in an interval.

Theorem 3. *If G has slick k-expression $Q = Q_\ell \oplus_{L,R,S} Q_r$, where Q_ℓ and Q_r generate graphs having no eigenvalues in (a, b), then G has at most $8k$ eigenvalues in (a, b).*

Proof. Let (n_+^a, n_0^a, n_-^a) be the number of positive, zero and negative diagonal elements produced by $\mathtt{diagonalize}(Q, a)$, and let (n_+^b, n_0^b, n_-^b) be the number of positive, zero and negative diagonal elements produced by $\mathtt{diagonalize}(Q, b)$. Then, using Sylvester's Law of Inertia, the number of eigenvalues of G in (a, b) is

$$n_+^a - n_{ge}^b$$

where $n_{ge}^b = n_0^b + n_+^b$.

During the execution of $\mathtt{diagonalize}(Q, a)$, let ℓ_+^a be the number of positive diagonal values produced by the call $\mathtt{diagonalize}(Q_\ell, a)$, and let r_+^a be the number of positive diagonal values produced by the call $\mathtt{diagonalize}(Q_r, a)$. Then

$$n_+^a = \ell_+^a + r_+^a + n_1 \tag{11}$$

where $n_1 \leq 4k$, as the remaining values will come from a matrix of order at most $4k \times 4k$.

Similarly during $\mathtt{diagonalize}(Q, b)$, let ℓ_{ge}^b denote the number of nonnegative diagonal values produced during the left subtree call and let r_{ge}^b denote the number of nonnegative diagonal values produced during the right subtree call. Then

$$n_{ge}^b = \ell_{ge}^b + r_{ge}^b + n_2 \tag{12}$$

where $n_2 \leq 4k$.

In the graph defined by Q_ℓ, let $n_{\ell,+}^a$ denote the number of eigenvalues greater than a. Since the ℓ_+^a positive diagonal elements produced in the left subtree call are valid diagonal elements in the diagonalization of Q_ℓ, and there can be at most $2k$ more,

$$\ell_+^a \leq n_{\ell,+}^a \leq \ell_+^a + 2k \tag{13}$$

Similarly, letting $n_{\ell,ge}^b$ denote the number of eigenvalues in Q_ℓ greater than or equal to b, we have

$$\ell_{ge}^b \leq n_{\ell,ge}^b \leq \ell_{ge}^b + 2k \tag{14}$$

As Q_ℓ has no eigenvalues in (a, b), we must have $n_{\ell,+}^a = n_{\ell,ge}^b$. From (13) and (14) we have

$$|\ell_+^a - \ell_{ge}^b| \leq 2k. \tag{15}$$

Since Q_r also has no eigenvalues in (a, b), a similar argument shows

$$|r_+^a - r_{ge}^b| \leq 2k. \tag{16}$$

Using (11) and (12) we have

$$\begin{aligned} n_+^a - n_{ge}^b &= (\ell_+^a + r_+^a + n_1) - (\ell_{ge}^b + r_{ge}^b + n_2) \\ &= (\ell_+^a - \ell_{ge}^b) + (r_+^a - r_{ge}^b) + (n_1 - n_2) \end{aligned}$$

Using (15), (16) and the fact that $|n_1 - n_2| \leq 4k$ we conclude that $n_+^a - n_{ge}^b \leq 8k$, completing the proof. $\square$

Consider the following construction.

Construction 1. *Let $G_\ell = (V_\ell, E_\ell)$ and $G_r = (V_r E_r)$ be arbitrary cographs, and let $W_\ell \subseteq V_\ell$ and $W_r \subseteq V_r$ be arbitrary sets of vertices in G_ℓ and G_r respectively. Let G be the graph formed by taking $G_\ell \cup G_r$ and adding edges $W = W_\ell \times W_r$.*
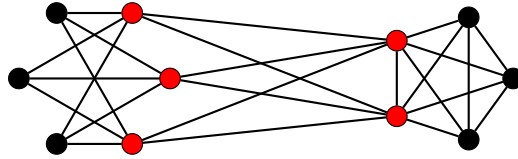
FIGURE 6.   Slick join of two cographs.

We claim that the graph G in Construction 1 can be defined with a slick 2-expression $Q_\ell \oplus_{L,R,S} Q_r$ where Q_ℓ and Q_r generate cographs. Indeed, since any cograph can be constructed with a slick 1-expression, there must exist a slick 2-expression Q_ℓ that constructs G_ℓ, assigning label 2 exactly to vertices in W_ℓ. Similarly, there exists a slick 2-expression Q_r that constructs G_r, assigning label 2 exactly to vertices in W_r. The graph G can be formed with $Q_\ell \oplus_{L,R,S} Q_r$ where $S = \{(2,2)\}$. Since cographs have no eigenvalues in $(-1,0)$ (see [25]), by Remark 4 and Theorem 3 it follows that $m_G(\lambda) \leq 4k = 8$ for each $\lambda \in (-1,0)$ and that G has at most $8k = 16$ eigenvalues in the interval $(-1,0)$. Observe that we may want to place only certain edges in W between G_ℓ and G_r but this will require additional labels, thus increasing k.

Note that Construction 1 yields another cograph if $\min\{|W_\ell|, |W_r|\} = 0$ or if $W_\ell = V_\ell$ and $W_r = V_r$ since the disjoint union and the join of two cographs are cographs. Figure 6 illustrates a graph constructed in this way, and for simplicity the cographs are small. The left and right cographs are isomorphic to $K_{3,3}$ and K_5 respectively. The vertices in $W_\ell \cup W_r$ are depicted in red. It is easy to see this graph is not a cograph since it has an induced P_4 (for example take the bottom four vertices).

Construction 1 seems interesting because the bounds of 8 and 16 are independent of both n and $|W|$. Indeed, we can place *arbitrarily* many edges between *arbitrary* cographs and yet the number of eigenvalues in $(-1,0)$ is bounded by a constant. Among the graphs G constructed in this way, how close are these bounds? That is, for $\lambda \in (-1,0)$, how large can $m_G(\lambda)$ be, and how many eigenvalues can G have in $(-1,0)$?

To underscore the somewhat unexpected nature of a constant bound when adding arbitrarily many edges, consider taking an arbitrary graph H having no eigenvalues in (a,b), and adding t edges to it. Using the interlacing result from [13, Thr. 3.9], one may show that the number of eigenvalues in (a,b) of the resulting graph G is bounded by $2t$. The following construction shows that this bound is tight.

Construction 2. *Consider the graph H on $4t$ vertices having $2t$ components, each a K_2, whose spectrum is $-1^{2t}; 1^{2t}$. Form G by adding t edges so that G has t components, each a P_4. Then G has spectrum $-1.618^t; -.618^t; .618^t; 1.618^t$. In particular, H has no eigenvalues in $(-1,1)$, while G has $2t$.*

6. Concluding Remarks

Given a graph G and a real number c, we have designed an algorithm to find a diagonal matrix congruent to $A - cI$, where A is the adjacency matrix of G. This algorithm is very efficient if G has clique-width bounded by a constant k. The reason for this is that the algorithm uses an encoding of the graph, called a *slick expression*, which is closely related to the expression defining the graph by means of its clique-width. In fact, for constant k, we describe linear-time algorithms to translate one type of expression into the other. We applied this algorithm to the problem of locating graph eigenvalues, which is a basic problem in Spectral Graph Theory.

Several natural questions arise from this work. Most importantly, one may ask about ways of adapting this approach to locating eigenvalues of general graph matrices, i.e., of matrices $M = (m_{ij})$ of order n such that there is a graph G on the vertex set $\{1, 2, \ldots, n\}$ with the property that $m_{ij} \neq 0$ if and only if $\{i, j\} \in E(G)$. It is easy to see that essentially the same algorithm would work for M (and still run in time $O(k^3 n)$) if the following conditions hold:

(i) All nonzero off-diagonal entries of M have the same value.
(ii) The algorithm has (oracle) access to the diagonal entries of M, that is, it is allowed to query directly the diagonal entry of a node produced by the slick k-expression.

Indeed, condition (i) ensures that the crucial relationship between the small submatrix that is processed at each stage and the actual matrix being diagonalized is preserved (see equations (4), (5) and (6)), while condition (ii) allows us to initialize the matrices at each leaf node of the parse tree. For instance, this would allow us to locate the eigenvalues of the Laplacian and of the signless Laplacian matrices associated with a graph as long as, together with the slick k-expression, the algorithm were given the degree of each vertex v when $i(v)$ appears in the expression. (Of course, it is possible to compute, in polynomial-time, the degree of each vertex of a graph given by a k-expression, but we do not know how to do this in time $O(poly(k)n)$, though time $O(n + m)$ is possible.) A nice open question would be to adapt this approach to matrices whose off-diagonal values may be assigned more values. More generally, it would certainly be interesting to find extensions of this work to other width parameters, as well as to the computation of other related matrix tasks.

Another natural research direction would be to further investigate the notion of slick clique-width. Even though it was introduced here with the sole objective of simplifying the description of the algorithm, the slick-clique width may be interesting for its own sake. As mentioned in the introduction, the graphs with slick clique-width equal to 1 are precisely the cographs. In general, If H is an induced subgraph of G, then $scw(H) \leq scw(G)$ (for H, use the k-expression of G, ignoring the generation of any vertices in $V(G) - V(H)$. Remove all unnecessary operations from the expression.) Of course, this does not happen for general subgraphs, as all complete graphs are cographs and therefore satisfy $scw(K_n) = 1$. So, given k, it makes sense to ask for the set $\mathcal{F}_k$ of all graphs G such that $scw(G) > k$, but $scw(H) \leq k$ for any proper induced subgraph H of G. In other words, the elements of $\mathcal{F}_k$ are the minimal graphs that cannot be represented by a slick k-expression (minimality is with respect to the partial order given by induced subgraphs). It is common to characterize graph classes through their set of minimal forbidden substructures, and the theory of cographs tells us that $\mathcal{F}_1 = \{P_4\}$. The situation for $k = 2$ is already much more complicated: a simple argument shows that $scw(T) \leq 2$ for any tree T, and it is also possible to show that distance hereditary graphs have slick clique-width at most 2. However, for cycles C_n, we have

$$scw(C_n) = \begin{cases} 1, & \text{if } n \in \{3, 4\}; \\ 2, & \text{if } n \in \{5, 6\}; \\ 3, & \text{if } n \geq 7. \end{cases}$$

This already implies that the set $\mathcal{F}_2$ is infinite, as it contains C_n for any $n \geq 7$.

Finally, we think that it is possible to derive theoretical results about the location of eigenvalues for special classes of graphs through the analysis of the behavior of our

algorithm on these graphs. Of course, this would require a better description of slick expressions that define graphs in such a class.

References

[1] A. Alazemi, M. Anđelić, and S. K. Simić. Eigenvalue location for chain graphs. *Linear Algebra Appl.*, 505:194–210, 2016.

[2] L. Babai, D. Y. Grigoryev, and D. M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 310–324, New York, NY, USA, 1982. ACM.

[3] T. Bıyıkoğlu, S. K. Simić, and Z. Stanić. Some notes on spectra of cographs. *Ars Combin.*, 100:421–434, 2011.

[4] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, May 2008.

[5] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[6] R. R. Coifman and S. Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5 – 30, 2006. Special Issue: Diffusion Maps and Wavelets.

[7] D. G. Corneil, H. Lerchs, and L. S. Burlingham. Complement reducible graphs. *Discrete Appl. Math.*, 3(3):163–174, 1981.

[8] D. G. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, Apr. 2005.

[9] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993.

[10] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.

[11] I. Dumitriu and S. Pal. Sparse regular random graphs: Spectral density and eigenvectors. *Ann. Probab.*, 40(5):2197–2235, 09 2012.

[12] 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*, pages 354–362. ACM, New York, 2006.

[13] F. J. Hall, K. Patel, and M. Stewart. Interlacing results on matrices associated with graphs. *J. Combin. Math. Combin. Comput.*, 68:113–127, 2009.

[14] D. P. Jacobs and V. Trevisan. Locating the eigenvalues of trees. *Linear Algebra Appl.*, 434(1):81–88, 2011.

[15] D. P. Jacobs, V. Trevisan, and F. Tura. Eigenvalue location in threshold graphs. *Linear Algebra Appl.*, 439(10):2762–2773, 2013.

[16] D. P. Jacobs, V. Trevisan, and F. Tura. Eigenvalue location in cographs. *Discrete Applied Mathematics*, 2017. in press http://dx.doi.org/10.1016/j.dam.2017.02.007.

[17] D. Janssens and G. Rozenberg. On the structure of node-label-controlled graph languages. *Inf. Sci*, 20(3):191–216, 1980.

[18] D. Janssens and G. Rozenberg. Restrictions, extensions, and variations of NLC grammars. *Inf. Sci*, 20(3):217–244, 1980.

[19] T. Jiang and X. Ding. Spectral distribution of adjacency and laplace matrices of random graphs. *Ann. Appl. Probab.*, 20:2086–2117, 2010.

[20] M. Kaminski, V. V. Lozin, and M. Milanic. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009.

[21] L. Lovász. Random walks on graphs: A survey. In D. Miklós, V. T. Sós, and T. Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty*, volume 2, pages 353–398. János Bolyai Mathematical Society, Budapest, 1996.

[22] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007.

[23] B. D. McKay. The expected eigenvalue distribution of a large regular graph. *Linear Algebra and its Applications*, 40:203 – 216, 1981.

[24] C. Meyer. *Matrix analysis and applied linear algebra*. Societsmall clique-width,y for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.).

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

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

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

[28] G. F. Royle. The rank of a cograph. *Electron. J. Combin.*, 10:Note 11, 7 pp. (electronic), 2003.

[29] T. Sander. On certain eigenspaces of cographs. *Electr. J. Comb*, 15(1):8, 2008.

[30] Z. Stanić. On nested split graphs whose second largest eigenvalue is less than 1. *Linear Algebra and its Applications*, 430(8–9):2200–2211, Apr. 2009.

[31] E. Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2):251 – 266, 1994.

Dept. of Computer Science and Engineering, Pennsylvania State University
E-mail address: furer@cse.psu.edu

Instituto de Matemática, Universidade Federal do Rio Grande do Sul
E-mail address: choppen@ufrgs.br

School of Computing, Clemson University
E-mail address: dpj@clemson.edu

Instituto de Matemática, Universidade Federal do Rio Grande do Sul
E-mail address: trevisan@mat.ufrgs.br