Fast evaluation of interlace polynomials on graphs of bounded treewidth

Markus Bläser, Christian Hoffmann

October 26, 2018

Abstract

We consider the multivariate interlace polynomial introduced by Courcelle (2008), which generalizes several interlace polynomials defined by Arratia, Bollobás, and Sorkin (2004) and by Aigner and van der Holst (2004). We present an algorithm to evaluate the multivariate interlace polynomial of a graph with n vertices given a tree decomposition of the graph of width k. The best previously known result (Courcelle 2008) employs a general logical framework and leads to an algorithm with running time $f(k) \cdot n$, where $f(k)$ is doubly exponential in k. Analyzing the $GF(2)$-rank of adjacency matrices in the context of tree decompositions, we give a faster and more direct algorithm. Our algorithm uses $2^{3k^2+O(k)} \cdot n$ arithmetic operations and can be efficiently implemented in parallel.

1 Introduction

Inspired by some counting problem arising from DNA sequencing [ABCS00], Arratia, Bollobás, and Sorkin defined a graph polynomial which they called interlace polynomial [ABS04a]. It turned out that the interlace polynomial is related [ABS04a, Theorem 24] to the Martin polynomial, which counts the number of edge partitions of a graph into circuits. This polynomial has been defined in Martin's thesis from 1977 [Mar77] and generalized by Las Vergnas [LV83]. Further work on the Martin polynomial has been pursued [LV81, LV88, Jae88, EM98, EM99, Bol02], including a generalization to isotropic systems [Bou87, Bou88, Bou91, BBD97]. In particular, the Tutte polynomial of a planar graph and the Martin polynomial of its medial graph are related. This implies a connection between the Tutte polynomial and the interlace polynomial (see [EMS07] for an explanation).

One way to define the interlace polynomial is by a recursion that uses a graph operation. Arratia et. al. used a pivot operation for edges [ABS04a]. This operation is a composition of local complementations to neighbor vertices (see [AvdH04], where the operations are called switch operations). The orbits of graphs under local complementation are related to error-correcting codes and quantum states, and so is the interlace polynomial as well [DP08].

The interlace polynomial can also be defined by a closed expression using the $GF(2)$-rank of adjacency matrices [AvdH04, Bou05, EMS06]. This linear algebra approach has been used in several generalizations of the interlace polynomial. In this paper, we consider the multivariate interlace polynomial $C(G)$ defined by Courcelle [Cou08] (see Definition 2.1 below) as it subsumes the two-variable interlace polynomial of Arratia, Bollobás, and Sorkin [ABS04b] and the weighted versions of Traldi [Tra08], as well as the interlace polynomials defined by Aigner and van der Holst [AvdH04]. Furthermore, the interlace polynomials $Q(x, y)$ and Q_n^{HN}, which have emerged from a spectral view on the interlace polynomials [RP06], are also special cases of Courcelle's multivariate interlace polynomial.

1.1 Results and related work

Our aim is to present an algorithm that, given a graph $G = (V, E)$ and an evaluation point, i.e. a tuple of numbers $((x_a)_{a\in V}, (y_a)_{a\in V}, u, v)$, evaluates the multivariate interlace polynomial $C(G)$ at $((x_a)_{a\in V}, (y_a)_{a\in V}, u, v)$. Whereas this is a #P-hard problem in general [BH08], it is fixed parameter tractable with cliquewidth as parameter [Cou08, Theorem 23, Corollary 33]. This is a consequence of the fact that the interlace polynomial is a monadic second order logic definable polynomial. Such graph polynomials can be evaluated in time $f(k) \cdot n$, where n is the number of vertices of the graph and k is the cliquewidth. The function $f(k)$ can be very large and is not explicitly stated in most cases. In general, it grows as fast as a tower of exponentials the height of which is proportional to the number of quantifier alternations in the formula describing the graph polynomial [Cou08, Page 34]. In the case of the interlace polynomial, this formula involves two quantifier alternations [Cou08, Lemma 24], [CiO07]. If a graph has tree width k, its cliquewidth is bounded by 2^{k+1} [CO00]. Thus, the machinery of monadic second order logic implies the existence of an algorithm that evaluates the interlace polynomial of an n-vertex graph in time $f(k)\cdot n$, where k is the tree width of the graph and $f(k)$ is at least doubly exponential in k. (In particular, the interlace polynomial of graphs of treewidth 1, that is, of trees, can be evaluated in polynomial time, which also has been observed by Traldi [Tra08].)

The monadic second order logic approach is very general and can be applied not only to the interlace polynomial but to a much wider class of graph polynomials [CMR01]. However, it does not consider characteristic properties of the actual graph polynomial. In this paper, we restrict ourselves to the interlace polynomial so as to exploit its specific properties and to gain a more efficient algorithm (Algorithm 2). Our algorithm performs $2^{3k^2+O(k)}n$ arithmetic operations to evaluate Courcelle's multivariate interlace polynomial (and thus any other version of interlace polynomial mentioned above) on an n-vertex graph given a tree decomposition of width k (Theorem 6.4). The algorithm can be implemented in parallel using depth polylogarithmic in n (Section 7.2). Apart from evaluating the interlace polynomial, our approach can also be used to compute coefficients of the interlace polynomial, for example so called d-truncations [Cou08, Section 5] (Section 7.3). Our approach is not via logic but via the $GF(2)$-rank of adjacency matrices, which is specific to the interlace polynomial.

1.2 Obstacles

It has been noticed that the Tutte polynomial and the interlace polynomial are similar in some respect [ABS04b]: Both can be defined by a recursion using a graph operation, both can be defined as closed sums over edge/vertex subsets involving some kind of rank. These similarities suggest that evaluating the interlace polynomial using tree decompositions might work completely analogously to the respective approaches for the Tutte polynomial [And98, Nob98]. This is not the case because of the following problems.

Andrzejak's algorithm [And98] to evaluate the Tutte polynomial uses the deletion-contraction recursion for the Tutte polynomial (via Negami's splitting formula [Neg87]). Deletion and contraction of an edge has the nice property that it is compliant with tree decompositions: If we are given the tree decomposition of a graph and we delete (or contract) an edge, the original tree decomposition (or, in the case of edge contraction, a simple modification of it) is a tree decomposition of the modified graph. For the interlace polynomial, on the other hand, the respective graph operation is not compliant with tree decompositions: If we perform the pivot operation from [ABS04a] on a graph, it is not clear how to obtain a tree decomposition of the modified graph. In particular, a single pivot operation can turn a tree (treewidth 1) into a circle (treewidth 2), see Fig. 1.

Another problem is that in the Tutte case the recursion formula naturally generalizes from the simplest versions (chromatic polynomial) to the most general ones (it is the defining recursion of the Bollobás-Riordan graph invariant [BR99]; cf. also the

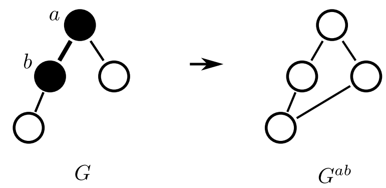

Figure 1: Edge pivoting, a central graph operation for the interlace polynomial, increases treewidth.

recurrence relation of the polynomial of Averbouch, Godlin and Makowsky, which generalizes the Tutte polynomial and the matching polynomial [AGM08]). The interlace polynomial, in contrast, needs more and more complicated recursions when generalizing the vertex-nullity interlace polynomial to the multivariate interlace polynomial[1] (see [Cou08, Proposition 12]).

When we consider Noble's algorithm [Nob98] and concentrate on the definition of the Tutte/interlace polynomial by sums involving ranks, another problem emerges. In the Tutte case, the rank is an easy to understand graph theoretic value, namely the number of vertices minus the number of connected components. Noble observes that the set of all partitions of a set of extension vertices captures all possible types of "behavior" of the rank (i.e. number of connected compontents) when adding some or all extension vertices. – For the interlace polynomial on the other hand, the rank used in the definition is the rank over $GF(2)$ of the adjacency matrix. Even though there exists a graph theoretic interpretation of this rank [Tra09], it is substantially more involved. Furthermore, an appropriate tool to capture the "rank behavior" when extending a graph (such as vertex partitions in the case of the Tutte polynomial) seems to be missing. The main contribution of this work is to devise such a tool and to prove that it works well with tree decompositions.

1.3 Outline

We compute the interlace polynomial by dynamic programming on the tree decomposition of a graph. To this end, we analyze the behavior of the $GF(2)$-rank of the adjacency matrix of a graph when the graph is extended by a fixed number of vertices (including the respective edges).

Section 2 contains the definition of Courcelle's multivariate interlace polynomial, which we will consider in this work. We will also fix our notation for tree decompositions there. In Section 3 we present our approach in detail. This includes the

[1]But note that Traldi reduced a three-term recursion to a two-term recursion [Tra08, Corollary 1].

motivation and definition of two central terms: extended graphs and scenarios. A scenario captures the behavior of the rank of an adjacency matrix when adding vertices. To define this precisely, we introduce symmetric Gaussian elimination in Section 4. In Section 5, we collect properties of scenarios which enable us to use scenarios with tree decompositions. In Section 6, we describe and analyze our algorithm, which evaluates the interlace polynomial by splitting it into parts according to scenarios. In Section 7 we discuss how our algorithm can be parallelized and used to compute (some of the) coefficients of the interlace polynomial. Finally, in Section 8, we mention directions for further research.

2 Preliminaries

We consider undirected graphs without multiple edges but with self loops allowed. Let $G = (V, E)$ be such a graph and $A \subseteq V$. By $G[A]$ we denote the subgraph of G induced by A, i.e. $(A, \{e \mid e \in E, e \subseteq A\})$. $G \nabla A$ denotes the graph G with self loops in A toggled, i.e. the graph obtained from G by performing the following operation for each vertex $a \in A$: if a has a self loop, remove it; if a does not have a self loop, add one.

The adjacency matrix of G is a symmetric square matrix with entries from $\{0, 1\}$. As the matrices that we will consider are adjacency matrices of graphs, we use vertices as column/row indices. Thus, the adjacency matrix of G is a $V \times V$ matrix $M = (m_{uv})$ over $\{0, 1\}$ with $m_{uv} = 1$ iff $uv \in E$. Furthermore, we will refer to entries and submatrices by specifying first the rows and then the columns: the (u, v)-entry of $M = (m_{uv})$ is m_{uv}, the $A \times B$ submatrix of M is the submatrix of the entries of M with row index in A and column index in B. All matrix ranks will be ranks over the field with two elements, $\{0, 1\} = GF(2)$, i.e. $+$ is XOR and $\cdot$ is AND. Slightly abusing notation we write $\mathrm{rk}(G)$ for the rank of the adjacency matrix of the graph G. The nullity (or co-rank) of an $n \times n$ matrix M is $\mathrm{n}(M) = n - \mathrm{rk}(M)$. If G is a graph, we write $\mathrm{n}(G)$ for the nullity of the adjacency matrix of G.

Graph polynomials are, from a formal perspective, mappings of graphs to polynomials that respect graph isomorphism. We will consider a *multivariate* graph polynomial, the multivariate interlace polynomial. To define such a polynomial, one has to distinguish "ordinary" indeterminates from *G-indexed indeterminates.* For instance, x being a G-indexed indeterminate means that for each vertex a of G there is a different indeterminate x_a. If $A \subseteq V$, we write x_A for $\prod_{a \in A} x_a$. Also, if $\mathcal{S}$ is a set, we write $\sum \mathcal{S}$ for the sum of all the elements in the set.

Definition 2.1 (Courcelle [Cou08]). *Let $G = (V, E)$ be an undirected graph. The multivariate interlace polynomial is defined as*

$$C(G) = \sum\{x_A y_B u^{\mathrm{rk}((G\nabla B)[A\cup B])} v^{\mathrm{n}((G\nabla B)[A\cup B])} \mid A, B \subseteq V, A\cap B = \emptyset\},$$

where u, v are called ordinary indeterminates and x, y G-indexed indeterminates.

2.1 Tree Decompositions

We borrow most of our notation from Bodlaender and Koster [BK08]. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ where T is a tree and each node $i \in I$ has a subset of vertices $X_i \subseteq V$ associated to it, called the bag of i, such that the following holds:

1. Each vertex belongs to at least one bag, that is $\bigcup_{i\in I} X_i = V$.
2. Each edge is represented by at least one bag, i.e. for all $e = vw \in E$ there is an $i \in I$ with $v, w \in X_i$.
3. For all vertices $v \in V$, the set of nodes $\{i \in I \mid v \in X_i\}$ induces a subtree of T.

The width of a tree decomposition $(\{X_i\}, T)$ is $\max\{|X_i| \mid i \in I\} - 1$. The treewidth of a graph G, $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of G.

Computing the treewidth of a graph is NP-complete. But given a graph with n vertices, we can compute a tree decomposition of width k (or detect that none exists) using Bodlaender's algorithm in time $2^{O(k^3)}n$ [Bod96].

To evaluate the interlace polynomial we will use *nice* tree decompositions. Note that our definition slightly deviates from the usual one[2]. This has no substantial influence on the running time of the algorithms discussed in this work but it simplifies the presentation of our algorithm. In a nice tree decomposition $(\{X_i\}, T)$, one node r with $|X_r| = 0$ is considered to be the root of T, and each node i of T is of one of the following types:

- Leaf: node i is a leaf of T and $|X_i| = 0$.
- Join: node i has exactly two children j_1 and j_2, and $X_i = X_{j_1} = X_{j_2}$.
- Introduce: node i has exactly one child j, and there is a vertex $a \in V$ with $X_i = X_j \cup \{a\}$.

[2]Usually, there is no special restriction on the bag size of the root node, and the leaf nodes contain exactly *one* vertex.

- Forget: node i has exactly one child j, and there is a vertex $a \in V$ with $X_j = X_i \cup \{v\}$.

A tree decomposition of width k with n nodes can be converted into a nice tree decomposition of width k with $O(n)$ nodes in time $O(n) \cdot \mathsf{poly}(k)$ [Klo94, Lemma 13.1.2, 13.1.3].

For a graph G with a nice tree decomposition $(\{X_i\}, T)$, we define

$$V_i = \Big(\bigcup \{X_j \mid j \text{ is in the subtree of } T \text{ with root } i\}\Big) \setminus X_i \quad \text{and} \quad G_i = G[V_i].$$

We can think of G_i as the subgraph of G induced by all vertices that have already been forgotten below node i.

3 Idea

We will now sketch our idea how to evaluate the interlace polynomial. Our approach is dynamic programming similar to the work of Noble [Nob98]. Let G be a graph for which we want to evaluate the interlace polynomial and $(\{X_i\}, T)$ a nice tree decomposition of G. For each node i of the tree decomposition, we have defined the graph G_i that consists of all vertices in the bags below i that are not in X_i. We will compute "parts" of the interlace polynomial of G_i. These parts are essentially defined by the answer to the following question: How does the rank of the adjacency matrix of some subgraph of G_i increase when we add (some or all) vertices of X_i? For the leaves these parts are trivial. Our algorithm traverses the tree decomposition bottom-up. We will show how to compute the parts of an introduce, forget, or join node from the parts of its child node (children nodes, resp.). At the root node, there is only one part left. This part is the interlace polynomial of G.

Before we go into details, let us remark that the answer to the above question ("How does the rank of the adjacency matrix increase when adding some vertices?") depends on the internal structure of the graph being extended. Consider the graph on the left hand side in Figure 2. If we extend it by the black vertices, the rank increases by 2. But if we use the graph on the left hand side in Figure 3, the *same extension* causes a rank increase by 4.

Let us see how we handle this issue. We start with the following definition.

Definition 3.1 (Extended graph)**.** *Let $G = (V, E)$ be some graph, $V', U \subseteq V$, $V' \cap U = \emptyset$. Then we define $G[V', U]$ to denote $G[V' \cup U]$ and call $G[V', U]$ an* extended graph*, the graph obtained by* extending $G[V']$ by U according to G. *We call U the* extension of $G[V', U]$.

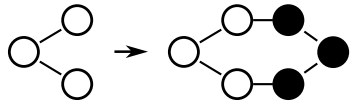

Figure 2: Interlace polynomial and rank behavior: Rank over $GF(2)$ of the adjacency matrix increases by 2 (from 2 to 4).

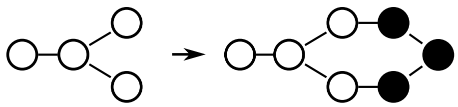

Figure 3: Interlace polynomial and rank behavior: Rank over $GF(2)$ of the adjacency matrix increases by 4 (from 2 to 6).

Let us fix an extension U. We consider all $V' \subseteq V(G)$ such that $G[V']$ may be extended by U according to the input graph G. For any such extended graph we ask: "How does the rank of $G[V']$ increase when adding some vertices of U?". Our key observation is that the answer to this question can be given without inspecting the actual G if we are provided with a compact description (of size independent of $n = |V(G)|$), which we call the scenario of $G[V', U]$.

The scenario of $G[V', U]$ (Definition 4.4) will be constructed in the following way. Consider M, the adjacency matrix of $G[V' \cup U]$. Perform symmetric Gaussian elimination on M *using only the vertices in* V' (for the details see Section 4). The resulting matrix M' is symmetric again and has the same rank as M. Furthermore, M' is of a form as in Figure 4: The $V' \times V'$ submatrix is a symmetric permutation matrix with some additional zero columns/rows. The nonzero entries correspond to edges or self loops (not of the original graph G but of some modified graph that is obtained from G in a well-defined way) "ruling" over their respective columns/rows: The edge between v_1 and v_8 rules over columns and rows v_1 and v_8. Here, "to rule" means that the only 1s in these columns and rows are the 1s at (v_1, v_8) and (v_8, v_1). Similarly, the self loop at vertex v_5 rules over column and row v_5. The columns and rows that are ruled by some edge or self loop in V' are also empty (i.e. entirely zero) in the $U \times V'$ submatrix of M'. Some columns/rows are not ruled by any edge or self loop in V', for instance column/row v_4. This is because there is neither a self loop at vertex v_4 nor does it have a neighbor in V'. However, v_4 may have neighbors in U. Thus, column v_4 of the $U \times V'$ submatrix may be any value from $\{0, 1\}^U$, which is indicated by the question marks. Also, the contents of the $U \times U$ submatrix is not known to us.

Let us choose a basis of the subspace spanned by the nonzero columns of the

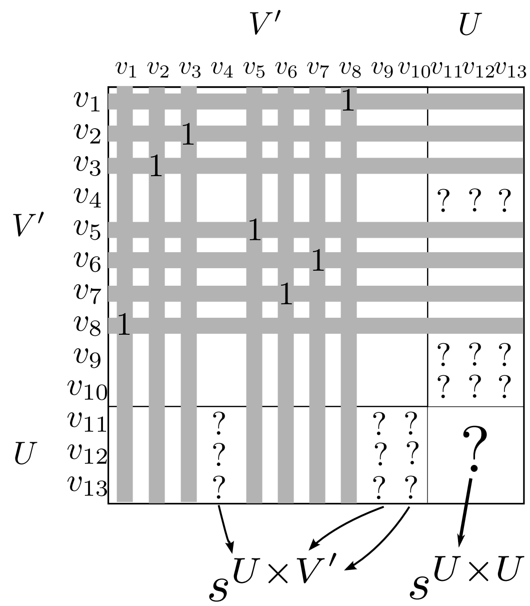

Figure 4: Adjacency matrix of $G[V' \cup U]$ after symmetric Gaussian elimination using V'. Empty entries are zero.

$U \times V'$ submatrix and call it $s^{U\times V'}$. Let $s^{U\times U}$ be contents of the $U \times U$ submatrix. By this construction, we are able to describe the rank of M' as the rank of its $V' \times V'$ submatrix plus a value that can be computed solely from $s^{U\times V'}$ and $s^{U\times U}$.

This will solve our problem that the rank increase depends on the internal structure of the graph $G[V']$ being extended: all we need to know is the scenario $s = (s^{U\times V'}, s^{U\times U})$ of $G[V', U]$. From s, without considering $G[V']$, we can compute in time $\mathsf{poly}(|U|)$ how the rank of the adjacency matrix of $G[V']$ increases when we add some vertices from U. This motivates the following definition.

Definition 3.2 (Scenario)**.** *Let U be an extension, i.e. a finite set of vertices. A* scenario of U *is a tuple $s = (s^{U\times V'}, s^{U\times U})$ where $s^{U\times V'}$ is an ordered set of linear independent vectors spanning a subspace of $\{0,1\}^U$ and $s^{U\times U}$ is a symmetric $(U\times U)$-matrix with entries from $\{0,1\}$. A* scenario for k vertices *is a scenario of some vertex set U with $|U| = k$.*

Let us come back to the evaluation of the interlace polynomial of G using a tree decomposition. Recall that at a node i of the tree decomposition we want to compute "parts" of the interlace polynomial of $G[V_i]$. Essentially every scenario s of X_i will define such a part: The interlace polynomial itself is a sum over *all* induced subgraphs with self loops toggled for some vertices. The part of the interlace polynomial corresponding to scenario s will be the respective sum not over all these graphs but only over the ones such that s is the scenario of $G[V_i, X_i]$. This will lead us to (6.1) in Section 6. To compute the parts of a join, forget and introduce node from the parts of its children nodes (child node, resp.), we will employ Lemma 6.1, 6.2 and 6.3. These are based on the fact that scenarios are compliant with tree decompositions, which we will prove in Section 5 (Lemma 5.1, Lemma 5.3 and Lemma 5.5). An example for the overall procedure of the algorithm is depicted in Figure 5.

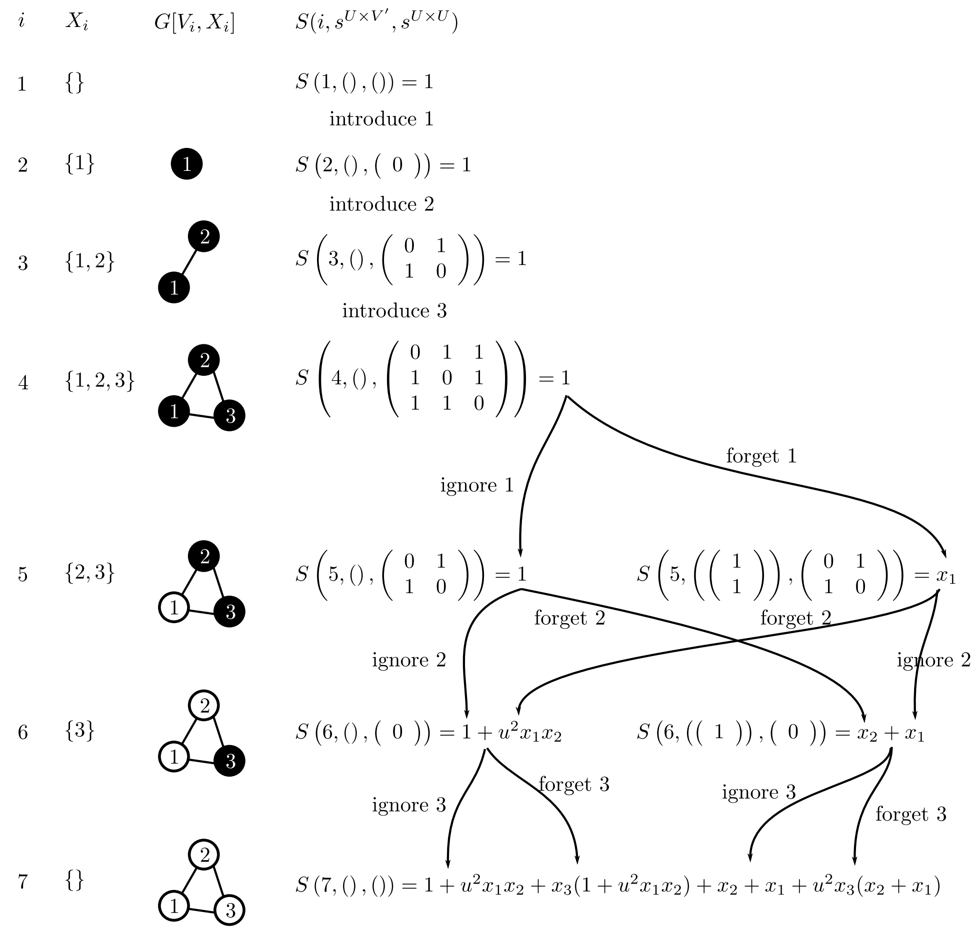

Figure 5: Computation of the interlace polynomial $C(G; y = 0, v = 1)$ of a triangle. In order to simplify the illustration, we ignore parameter D in (6.1), which handles the "self loop toggling feature" of the interlace polynomial.

The time bound of our algorithm stems from the following observation: The number of parts managed at a node i of the tree decomposition is essentially bounded by the number of scenarios of its bag X_i. This number is independent of the size of G and single exponential in the bag size (and thus single exponential in the treewidth of G):

Lemma 3.3. *Let U be an extension, i.e. a finite set of vertices, $|U| = k$. There are less than $2^{(3k+1)k/2}$ scenarios of U.*

Proof. The number of symmetric $\{0,1\}$-matrices of dimension $k \times k$ is $2^{(k+1)k/2}$, as a symmetric matrix is determined by its left lower half. Thus, there are $2^{(k+1)k/2}$ possibilities for $s^{U \times U}$.

For $s^{U \times V'}$, there less than 2^{k^2} possibilities: As there are $2^k - 1$ non-zero elements of $\{0,1\}^k$, the number of linear independent subsets of $\{0,1\}^U$ with d elements is bounded by $\binom{2^k-1}{d}$. Thus, the number of *all* linear independent subsets of $\{0,1\}^U$ is at most

$$\sum_{0 \le d \le k} \binom{2^k - 1}{d} \le (k+1)\binom{2^k - 1}{k} < 2^{k^2}. \qquad \square$$

4 Symmetric Gaussian Elimination

We want to convert adjacency matrices into matrices of a form as in Figure 4 without touching the rank. In order to achieve this, we introduce a special way of performing Gaussian elimination that differs from standard Gaussian elimination in the following way. First, it is symmetric, as in general every column operation is followed by a corresponding row operation. In this way, we maintain the correspondence between rows/columns of the matrix we are manipulating and vertices of a graph. Second, we adhere to a particular order when deciding which entry to use for the next pivot operation. This order is (partially) fixed by the tree decomposition. It is crucial for our proofs of the statements in Sect. 5 that the elimination process proceeds according to this order. Third, we perform symmetric Gaussian elimination using only vertices in a *subset* V' of the vertices: When seeking a pivot entry in a particular row/column, we do not consider all entries of the row/column but only the ones that correspond to edges between vertices in V'.

4.1 Elimination with a single vertex

Assume we are given a graph G, its adjacency matrix M and a vertex v. We would like to compute the rank of M as the "effect of v on the rank" plus the rank of a submatrix in which we have deleted v. This might not immediately be possible using

M itself, but we can achieve it modifying M by a *symmetric Gaussian elimination step* on M using v. This is defined in the following way:

- If v is an isolated vertex without a self loop, we have situation (1) of Figure 6. Vertex v has no influence on the rank of the adjacency matrix and we can delete the column and row corresponding to v without changing the rank of the adjacency matrix. The result of the elimination step is just M.

- If v has a self loop, there is a 1 in the (v, v)-entry of M. The elimination step consists of the following operations. Except for entry (v, v), we remove all 1s in the v-column and v-row using the following pair of operations for each neighbor u of v: First, add the v-column to the u-column. Then, in the modified matrix, add the v-row to the u-row. We denote the result of the whole process by $M \rtimes v$, which is depicted as (2) in Figure 6. Note that $M \rtimes v$ is symmetric again. The rank of M equals 1 plus the rank of $M \rtimes v$ with v-column and v-row deleted.

- If v is neither isolated nor has a self loop, there is a neighbor u of v. Assume that u does not have a self loop. The (u, v)- and (v, u)-entries of M equal 1. The elimination step consists of the following operations. In the first stage, except for (u, v) and (v, u), we remove all 1s in the v-column and v-row. This is accomplished by the following pair of operations for each neighbor u' of v, $u' \neq u$: First, add the u-column to the u'-column. Then, in the modified matrix, add the u-row to the u'-row. Again, performing such a pair of column/row operations keeps a symmetric matrix symmetric. At the end of the first stage the v-column and v-row consist entirely of 0s, except for the entry at the u-position, which is 1. The second stage proceeds as follows: we add the v-column to every column which has a 1 in the u-row, and we also add the v-row to every row which has a 1 in the u-column. At the end of this stage also the u-column and u-row consist only of 0s except at the v-position. The result of the second stage is a symmetric matrix again, which we denote by $M \rtimes vu$. It is depicted as (3) in Figure 6. We do not swap columns/rows, as we must keep the vertices in a particular order, which is determined by the tree decomposition, cf. Section 4.2. The rank of M equals 2 plus the rank of $M \rtimes vu$ with u- and v-column and u- and v-row deleted.

  If u has a self loop we proceed analogously to obtain a matrix with a structure as (4) in Figure 6. Then we can eliminate the self loop at u by, say, adding column v to column u. (As at this point column v is zero everywhere except at u, only entry (u, u) of the matrix is changed by this operation and the symmetry is not destroyed.) Thus, we obtain a matrix exactly as (3) in Fig. 6.

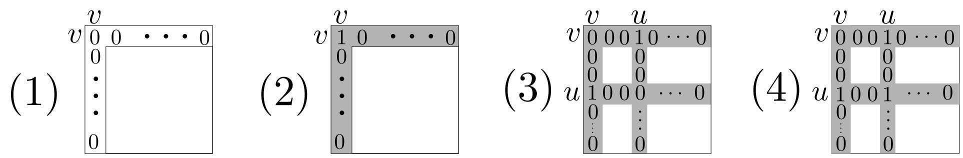

Figure 6: Effect of a symmetric Gaussian elimination step. Adjacency matrix with isolated unlooped vertex v (1), adjacency matrix after eliminating with a self loop at v (2), adjacency matrix after eliminating with edge vu (3).

We can describe the effect of a symmetric elimination step on the entries of the matrix (aside from the entries being set to 0) in the following way.

Lemma 4.1. *Let $M = (m_{ij})$ be an adjacency matrix, let a be a vertex with a self loop, and m_{yx} some entry of M which is not in column or row a, i.e. $a \notin \{x, y\}$. Then, after symmetric Gaussian elimination using a, the (y, x)-entry of M will be*

$$(M \rtimes a)_{yx} = m_{yx} + m_{ax}m_{ya}.$$

Lemma 4.2. *Let $M = (m_{ij})$ be an adjacency matrix, let a be a vertex without a self loop, ab an edge and m_{yx} some entry of M which is not in column or row a or b, i.e. $\{x, y\} \cap \{a, b\} = \emptyset$. Then, after symmetric Gaussian elimination using ab, the (y, x)-entry of M will be*

$$(M \rtimes ab)_{yx} = m_{yx} + m_{ax}m_{yb} + m_{ya}m_{bx} + m_{ax}m_{ya}m_{bb}.$$

We prove the statement about edge elimination, the case of self loop elimination is completely analogously.

Proof of Lemma 4.2. Let us assume that $x \leq y$ (the case $x > y$ is analogous). The situation is depicted in Figure 7. Depending on the (a, x)-entry being 1 or not, column b is added to column x, which adds the (y, b)-entry to the (y, x)-entry. This gives the term $m_{ax}m_{yb}$. After that, depending on the (y, a)-entry, row b is added to row y. This adds the actual value of the (b, x)-entry to the (y, x)-entry. By the previous column addition, the actual (b, x)-entry is $m_{bx} + m_{ax}m_{bb}$. Thus, the row addition contributes a term $m_{ya}(m_{bx} + m_{ax}m_{bb})$. The second stage has no effect on the (y, x) entry: Column a may be added to some other columns. But at this point of time, column a is entirely zero, except at the b entry. Thus, addition of the a column has no effect on the (y, x) entry. The same is true for addition of the a row. $\square$

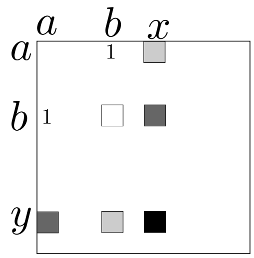

Figure 7: During a symmetric Gaussian elimination step using edge ab, entry (y, x) is affected only by the entries at (a, x), (y, b), (y, a), (b, x) and (b, b).

4.2 Vertex order, elimination with vertex sets, and the scenario of an extended graph

We want to define symmetric Gaussian elimination using a whole set $V' \subseteq V$ of vertices. This means that we perform elimination steps using each vertex from V'. The result of this process depends on the order in which we use the vertices for elimination steps. Therefore we introduce an order on the vertices of the graph, which will be computed before the computation of the interlace polynomial starts. We will use this order throughout the rest of the paper. Whenever there could be any ambiguity, we proceed according to this order.

The vertex order we are using must be compliant with the tree decomposition we are using: Whenever a vertex is forgotten, it must be greater than all the vertices which have been forgotten before. Or, equivalently, the vertices in the extension X_i must be greater than the vertices in V_i for each node i of the tree decomposition. Such an order can be obtained by Algorithm 1.

Now we are ready to define elimination using a set of vertices.

Definition 4.3. *Let $V' \subseteq V$ be a set of vertices of a graph $G = (V, E)$ with adjacency matrix M.* Symmetric Gaussian elimination on G using V' *is defined as the following process: If $V' = \emptyset$, we are done and M is the output of the symmetric Gaussian elimination process using V'. Otherwise, we let v be the minimum vertex in V'. If v has a self loop we let $M' = M \rtimes v$. Otherwise, we check whether v has a neighbor u in V'. If yes, we let $M' = M \rtimes vu$, where u is the minimum neighbor of v. If no, we let $M' = M$. This concludes the processing of v. To complete the elimination using V', we continue recursively with $V' \setminus \{v\}$ in the role of V' and M' in the role of M.*

We also order vertex vectors (i.e. elements from $\{0, 1\}^U$, U some vertex set) and sets of vertex vectors according to the vertex order (lexicographically). This induced order is used for choosing a "minimal" basis in the following definition.

Algorithm 1 Supplying a vertex order.

```
procedure SupplyVertexOrder
    c ← 1
    for all nodes i, in the order of bottom-up traversal, i.e. each father node is
      visited after all its children do
        if i is a forget node then
            a ← vertex being forgotten at node i
            give vertex a number c in the vertex order
            c ← c + 1
        end if
    end for
end procedure
```

Definition 4.4 (Scenario of an extended graph)**.** *Let $G[V', U]$ be an extended graph obtained by extending $G[V']$ by U according to graph $G = (V, E)$. Let the vertex order be such that $v' < u$ for all $v' \in V'$ and $u \in U$. Then the scenario* $\mathrm{scen}(G[V', U])$ *of $G[V', U]$ is defined as follows: Let M be the adjacency matrix of $G[V' \cup U]$. Perform symmetric Gaussian elimination on M using V' to obtain M'. Let $M'_{UV'}$ be the $U \times V'$ submatrix of M'. Consider the column space W of $M'_{UV'}$. We can choose a basis of W from the column vectors of $M'_{UV'}$. Let $s^{U \times V'}$ be the minimal such basis. Let $s^{U \times U}$ be the contents of the $U \times U$ submatrix of M'. We define* $\mathrm{scen}(G[V', U])$ *to be $(s^{U \times V'}, s^{U \times U})$.*

The minimal basis $s^{U \times V'}$ in the preceding definition can by obtained by the following steps: Start with an empty set of columns and then as often as possible take the minimum column of $M'_{UV'}$ which is not in the span of the so far collected columns.

5 Scenarios and nice tree decompositions

Consider a join node i with children j_1 and j_2 in a nice tree decomposition of a graph G the interlace polynomial of which we want to evaluate. By the properties of tree decompositions, this implies a situation as depicted in Figure 8: $G_{j_1} = G[V_{j_1}]$ and $G_{j_2} = G[V_{j_2}]$ are disjoint graphs with a common extension $X_{j_1} = X_{j_2} = X_i$. $G_i = G[V_i] = G[V_{j_1} \cup V_{j_2}]$ is the disjoint union of G_{j_1} and G_{j_2}. Assume that we have computed all parts (see Section 3 and (6.1)) of the interlace polynomial of G_{j_1} and all parts of the interlace polynomial of G_{j_2}. From this we want to compute the parts

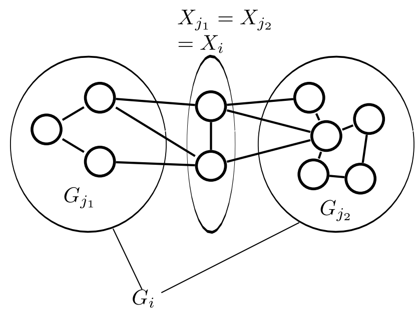

Figure 8: Graphs corresponding to a join node i and its child nodes j_1, j_2.

of the interlace polynomial of G_i. Consider one such part, say the one corresponding to some scenario s of X_i. Somehow we have to find out for which subgraphs[3] $G[V']$ of G_i the scenario of the extended graph $G[V', X_i]$ is s. Fortunately, these are exactly the subgraphs $G[V_1 \cup V_2]$, $V_1 \subseteq V_{j_1}$, $V_2 \subseteq V_{j_2}$, with the property that the "join" of the scenario of $G[V_1, X_{j_1}]$ and the scenario of $G[V_2, X_{j_2}]$ is s. This is guaranteed by the following lemma.

Lemma 5.1 (Join)**.** *Let $G = (V, E)$ be a graph, $U \subseteq V$, and s_1, s_2 two scenarios of U. Then there is a unique scenario s_3 of U such that the following holds: If $G[V_1]$ and $G[V_2]$ are disjoint subgraphs of G that may be extended by U according to G, $\mathrm{scen}(G[V_1, U]) = s_1$, and $\mathrm{scen}(G[V_2, U]) = s_2$, then $\mathrm{scen}(G[V_1 \cup V_2, U]) = s_3$. Moreover, s_3 can be computed from s_1, s_2 and $G[U]$ within $\mathsf{poly}(|U|)$ steps.*

Proof. We will apply Definition 4.4 to determine s_3. We will see that s_3 is uniquely defined by s_1, s_2 and $G[U]$, and can be computed from these within the claimed time bound. This will prove the lemma.

Let $G_1 = G[V_1]$ and $G_2 = G[V_2]$. Let M be the adjacency matrix of $G[V_1 \cup V_2 \cup U]$. As G_1 and G_2 are disjoint, M has a form as depicted on the left hand side in Figure 9, the $V_1 \times V_2$ submatrix as well as the $V_2 \times V_1$ submatrix of M consists only of 0s.

By Definition 4.4, symmetric Gaussian elimination using $V_1 \cup V_2$ has to be performed on M to obtain M', which is of the form depicted on the right hand side in Figure 9 and from which s_3 can be read off. Let us analyze a single elimination step occurring during the elimination process in detail, say eliminating with a self loop at a vertex $v \in V_1$. One action in this step is that the 1 in the (v, v) entry will be used to eliminate another 1 in the v-row by adding the v-column to the respective

[3] In fact induced subgraphs with self loops toggled at some vertices — but we will ignore this detail for the rest of the section as it is not important to understand the idea.

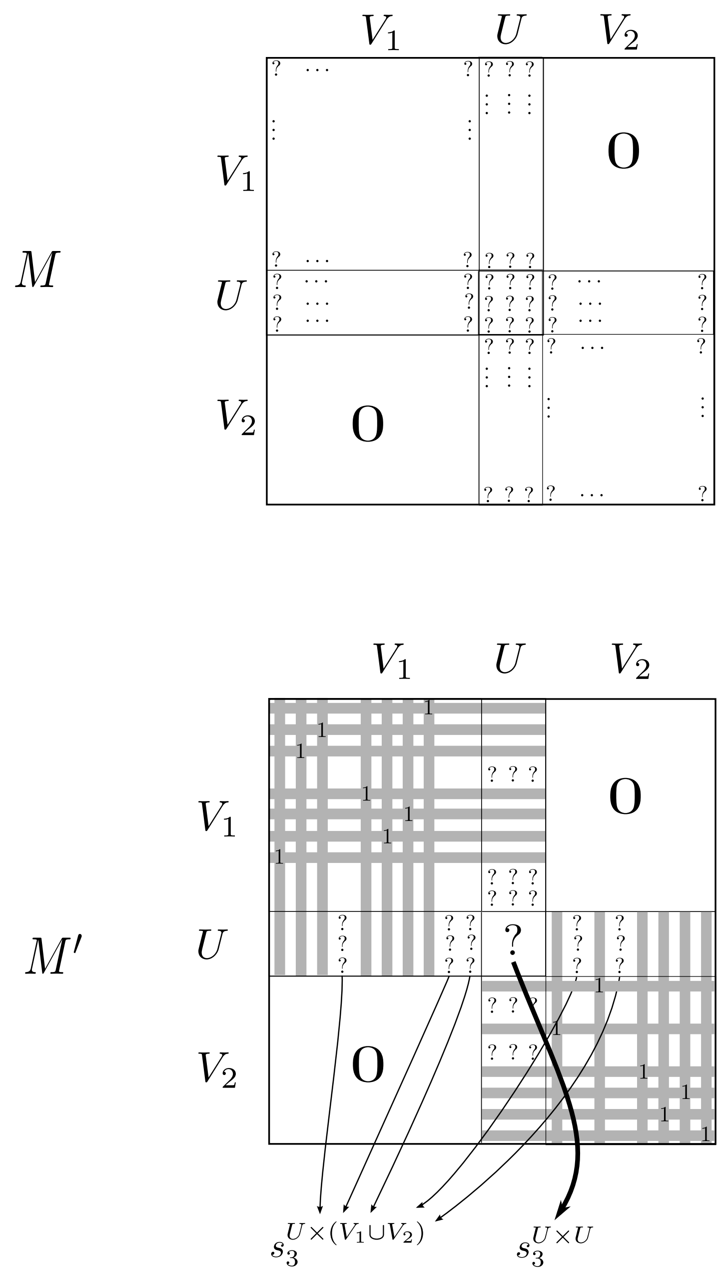

Figure 9: Effect of symmetric Gaussian elimination to gain the scenario of $G[V_1\dot{\cup}V_2, U]$.

column u. Let us argue that this does not affect neither the $V_1 \times V_2$ submatrix of M nor the $V_2 \times V_1$ submatrix of M. As $v \in V_1$, in the v-row the V_2-entries are already 0. Thus we know that $u \notin V_2$, i.e. the v-column will be added to a column from $V_1 \cup U$. Thus, the $V_1 \times V_2$ submatrix is not changed. Again as $v \in V_1$, the V_2-entries in the v-column are 0 and addition of the v-column to any other column u does not change the V_2-entries of column u. Thus, the $V_2 \times V_1$ submatrix of M is not changed.

Analogous observations can be made for the role of columns and rows reversed (i.e. when adding the v-row to other rows to eliminate 1s in the v-column), as well as for elimination steps using an edge between different vertices (instead of self loops). We conclude that symmetric Gaussian elimination steps with V_1-vertices affect only the $(V_1 \cup U) \times (V_1 \cup U)$ submatrix of M, but not the $V_1 \times V_2$ or $V_2 \times V_1$ submatrix. Analogously, elimination steps with V_2-vertices affect only the $(U \cup V_2) \times (U \cup V_2)$ submatrix of M. Thus, except for the $U \times U$ submatrix, when performing symmetric Gaussian elimination on M using $V_1 \cup V_2$, the same things happen as when performing symmetric Gaussian elimination first on $G[V_1 \cup U]$ using V_1 and then on $G[V_2 \cup U]$ using V_2. The only difference may be that depending on the vertex order elimination steps with V_1-vertices are interlaced with steps using V_2 vertices. But we argued that V_1-elimination steps do not influence parts of M which are relevant for V_2-elimination steps and vice versa, so this is not an issue.

As elimination on M using $V_1 \cup V_2$ (yielding M') on the one hand does the same as elimination on $G[V_1 \cup U]$ using V_1 (yielding, say, $M^{(1)}$) and elimination on $G[V_2 \cup U]$ using V_2 (yielding, say, $M^{(2)}$) on the other hand, the $U \times (V_1 \cup V_2)$ submatrix of M' is just the union of $M^{(1)}_{UV_1}$, the $U \times V_1$ submatrix of M_1, and $M^{(2)}_{UV_2}$, the $U \times V_2$ submatrix of M_2. Recall that ${s_1}^{U \times V_1}$ and ${s_2}^{U \times V_2}$ are minimum bases of the column space of $M^{(1)}_{UV_1}$, $M^{(2)}_{UV_2}$, resp, taken from the columns of these matrices. To compute ${s_3}^{U \times (V_1 \cup V_2)}$, the minimum basis of the column space of the $U \times (V_1 \cup V_2)$ submatrix of M' taken from the columns of this matrix, we proceed in the following way: Start with the empty set and as long as possible add the minimum vector of ${s_1}^{U \times V_1} \cup {s_2}^{U \times V_2}$ which is not in the span of the so far collected vectors. This can be done in time polynomial in $|U|$ using standard Gaussian elimination.

The $U \times U$ submatrix is the only part of M which is affected by both, eliminations with V_1-vertices and eliminations with V_2-vertices. However, the use of the $U \times U$ submatrix is “write-only” during the elimination process: Consider symmetric Gaussian elimination in general, say on some extended graph $G[V' \cup U]$ using V'. Recall that by Definition 4.3 all the elimination steps will involve only vertices from V' in the sense that the step is either $M \rtimes v$ or $M \rtimes vu$ with $u, v \in V'$. Thus, the contents of the $U \times U$ submatrix has no influence on what elimination steps will be performed. All that happens with this submatrix is that column/row vectors are

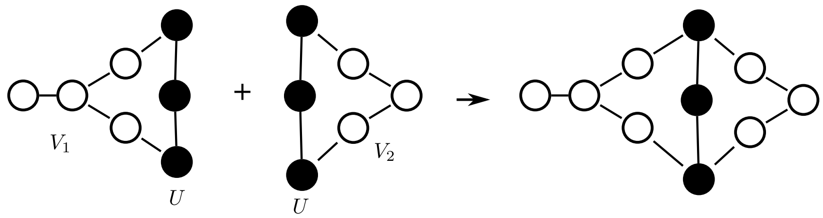

Figure 10: Joining the extended graphs $G[V_1, U]$ and $G[V_2, U]$.

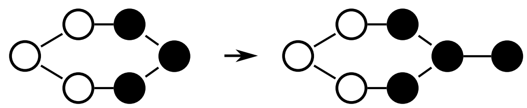

Figure 11: Adding a vertex to an extension.

added to it.

Thus, the effect on the $U \times U$ submatrix of all the elimination steps during symmetric Gaussian elimination of $G[V_1 \cup U]$ using V_1 can be described as adding a matrix, say A_1 to the adjacency matrix of $G[U]$. We can compute A_1 as $A_1 = {s_1}^{U\times U} - M(G[U])$, where $M(G[U])$ denotes the adjacency matrix of $G[U]$. Analogously, we can compute A_2 which describes the effect of symmetric Gaussian elimination of $G[V_2 \cup U]$ using V_2 on the $U \times U$ submatrix. Because of the "write-only" property, the effect of symmetric Gaussian elimination of M using $V_1 \cup V_2$ on the $U \times U$ submatrix of M can be described by $A_1 + A_2$. Thus we have ${s_3}^{U\times U} = M(G[U]) + A_1 + A_2$, which is the second component of s_3. □

Definition 5.2. *In the situation of Lemma 5.1 we write $s_{\text{join}}(s_1, s_2, G[U])$ for s_3.*

To handle join nodes of the tree decomposition we proved Lemma 5.1: from the scenario of two extended graphs $G[V_1, U]$ and $G[V_2, U]$ with a common extension U we can compute the scenario of the joined extended graph $G[V_1 \cup V_2, U]$ (cf. Figure 10). To handle also introduce and forget nodes we prove two more lemmas (cf. Figure 11, Figure 12).

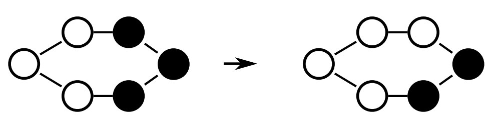

Figure 12: Transforming an extending vertex into a normal vertex.

Lemma 5.3 (Introduce vertex)**.** *Let $G = (V, E)$ be a graph, $U \subseteq V$, s a scenario of U, $u \in V \setminus U$. Then there is a unique scenario $\tilde{s}$ of $\tilde{U} = U \cup \{u\}$ such that the following holds: If $G[V']$ may be extended by $\tilde{U}$ according to G, u is not connected to V' in G, and* $\mathrm{scen}(G[V', U]) = s$*, then* $\mathrm{scen}(G[V', \tilde{U}]) = \tilde{s}$*. Moreover, $\tilde{s}$ can be computed from s and $G[\tilde{U}]$ in* $\mathsf{poly}(|U|)$ *steps.*

Proof. As u is not connected to V', $\tilde{s}^{\tilde{U} \times V'}$ is $s^{U \times V'}$ with a zero component for u added to all the basis vectors. Also, $\tilde{s}^{\tilde{U} \times \tilde{U}}$ is just $s^{U \times U}$ with a row and column added representing the neighbors of u in $\tilde{U}$. □

Definition 5.4. *In the situation of Lemma 5.3 we write $s_{\mathrm{introduce}}(s, u, G[\tilde{U}])$ for $\tilde{s}$.*

Lemma 5.5 (Forget vertex)**.** *Let $G = (V, E)$ be a graph, $u \in U \subseteq V$, $\tilde{U} = U \setminus \{u\}$, $\tilde{V} = V' \cup \{u\}$, and s a scenario of U. Then there is a unique scenario $\tilde{s}$ of $\tilde{U}$ and $r, n \in \{0, 1, 2\}$ such that the following holds: If $G[V']$ is a subgraph of G that may be extended by U according to G, $u > v'$ for all $v' \in V'$, and* $\mathrm{scen}(G[V', U]) = s$*, then* $\mathrm{scen}(G[\tilde{V}, \tilde{U}]) = \tilde{s}$ *and the rank (nullity) of the adjacency matrix of $G[\tilde{V}]$ equals the rank (nullity, resp.) of the adjacency matrix of $G[V']$ plus r (n, resp.). Moreover, $\tilde{s}$ and r can be computed from s and $G[U]$ in* $\mathsf{poly}(|U|)$ *steps, and we have $n = 2 - r$.*

Proof. Consider the situation after symmetric Gaussian elimination on $G[V' \cup U] = G[\tilde{V} \cup \tilde{U}]$ using V' (Figure 13). We distinguish three cases: (1) there is a basis vector of the $(U \times V')$ column space with a 1 in the u-component, (2) there is no such basis vector, but the (u, u)-entry of the $U \times U$ submatrix equals 1, (3) neither case (1) nor (2).

Let us first consider cases (2) and (3). As all u-components of the vectors in $s^{U \times V'}$ are zero, we know that symmetric Gaussian elimination on $G[\tilde{V} \cup \tilde{U}]$ using $\tilde{V}$ will consist of the following two stages: first, exactly the same operations will be performed as in symmetric Gaussian elimination on $G[V' \cup U]$ using V' (which will end up in the situations depicted in Figure 13 (2), (3)), and then elimination using vertex u will be performed if possible.

Thus, in case (3), $\tilde{s}$ can be obtained from s in the following way: remove the u component of each vector of $s^{U \times V'}$ to gain $\bar{s}^{\tilde{U} \times \tilde{V}}$. Let a be the first column of $s^{U \times U}$. Remove the first component of a. With standard Gaussian elimination, check in time $\mathsf{poly}(|U|)$ if a is in the span of $\bar{s}^{\tilde{U} \times \tilde{V}}$. If it is, let $\tilde{s}^{\tilde{U} \times \tilde{V}} = \bar{s}^{\tilde{U} \times \tilde{V}}$, otherwise let $\tilde{s}^{\tilde{U} \times \tilde{V}} = \bar{s}^{\tilde{U} \times \tilde{V}} \cup \{a\}$. Let $\tilde{s}^{\tilde{U} \times \tilde{U}}$ be $s^{U \times U}$ with first column and first row deleted. We have $r = 0$ and $n = 2$. (Note that the step from $G[V']$ to $G[\tilde{V}]$ adds a row *and* a column to the adjacency matrix, so the nullity increases by 2.)

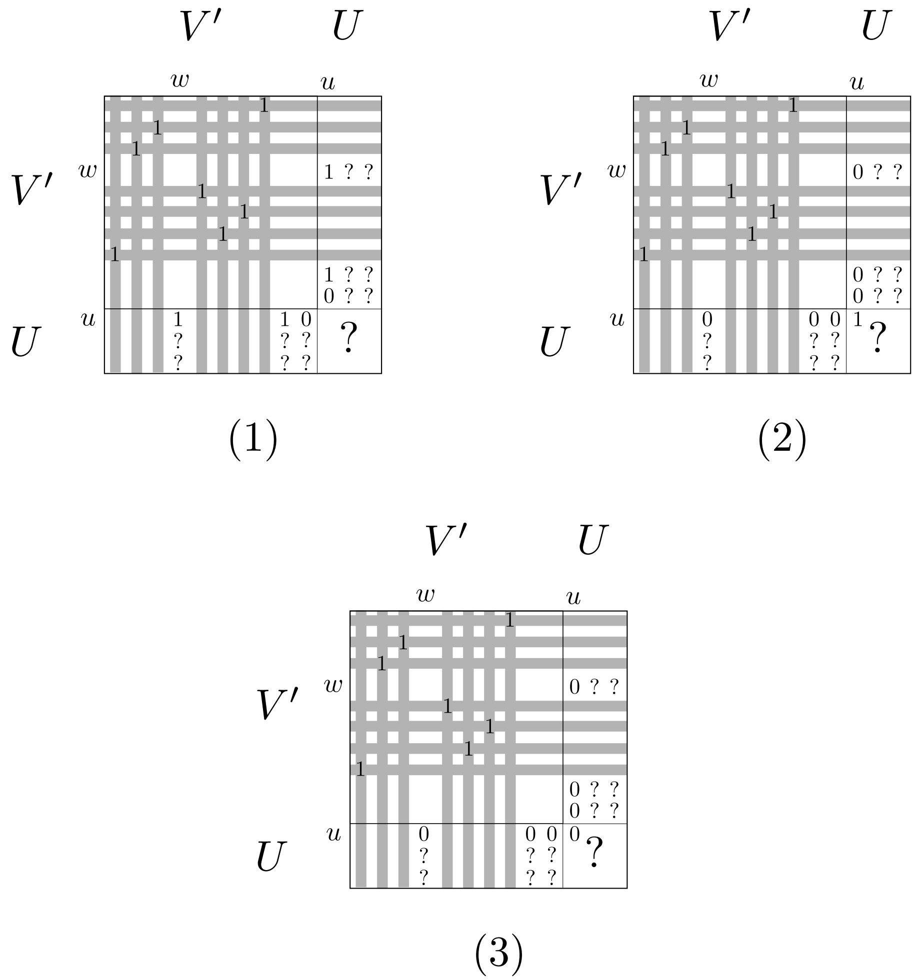

Figure 13: Cases when “forgetting” an extension vertex u.

In case (2), we first perform an elimination step with the 1 at the (u, u)-entry: let $\bar{s}^{U\times U} = s^{U\times U} \rtimes u$. Then we continue as in case (3) but with $\bar{s}^{U\times U}$ in the role of $s^{U\times U}$. We have $r = n = 1$.

The rest of this proof deals with case (1). Let $w \in V'$ be the vertex corresponding to the minimum vector of $s^{U\times V'}$ with a 1 in the u-component (cf. Figure 13 (1)). Compare symmetric Gaussian elimination on $G[V' \cup U]$ using V' (which is performed to obtain s) to symmetric Gaussian elimination on $G[\tilde{V} \cup \tilde{U}]$ using $\tilde{V}$ (which is performed to obtain $\tilde{s}$). Before these two processes reach w, they are equal, but from w on they will differ: Using V', the edge uw will not be used for elimination and the process will continue with the next vertex in V' immediately. Using $\tilde{V}$, the edge uw will be used for elimination (which will not affect the $V' \times V'$ submatrix, but possibly change the contents of the $U \times (V' \cup U)$ and the $(V' \cup U) \times U$ submatrices). Only after that, the process will continue with the next vertex in $\tilde{V}$. However, we will prove in Lemma 5.8 that we can defer the elimination using edge uw until all vertices of V' have been proceeded and still obtain $\tilde{s}$. Thus, $\tilde{s}$ can be computed in the following way: perform the same steps as with symmetric Gaussian elimination on $G[V' \cup U]$ using V'. Then, simulate the effect of a symmetric Gaussian elimination step using edge uw in a similar way as in cases (2) and (3).

This simulation can be done as follows: Let $\vec{w}$ be the minimum vector of $s^{U\times V'}$ with the u-component equal to 1. Let $\bar{s}^{U\times V'} = s^{U\times V'} \setminus \{\vec{w}\}$ and $\bar{s}^{U\times U} = s^{U\times U}$. For each row i, $i \neq u$, with the $\vec{w}_i = 1$ simulate addition of column/row u to column/row i doing the following:

1. For each vector $\vec{c}$ of $\bar{s}^{U\times V'}$, add component u of $\vec{c}$ to component i of $\vec{c}$.
2. Change $\bar{s}^{U\times U}$ by first adding the u column to the i column and then, in the modified matrix, the u row to the i row.

We have $\tilde{s}^{\tilde{U}\times\tilde{V}} = \bar{s}^{U\times V'}$, and $\tilde{s}^{\tilde{U}\times\tilde{U}}$ is $\bar{s}^{U\times U}$ with first column and first row removed. Note that after an elimination step using edge wu, the u column/row will consist entirely of zeros (except at (u, w) and (w, u)). Thus, the first column of $\bar{s}^{U\times U}$ will be zero after the elimination with wu and we do not need to incorporate it into $\tilde{s}^{\tilde{U}\times\tilde{V}}$.

Finally note that we have $r = 2$ and $n = 0$ in case (1). $\square$

Definition 5.6. *In the situation of Lemma 5.5 we write* $s_{\text{forget}}(s, u, G[U])$ *for* $\tilde{s}$, $\Delta r_{\text{forget}}(s, u, G[U])$ *for* r, *and* $\Delta n_{\text{forget}}(s, u, G[U])$ *for* n.

The operation defined in Definition 5.6 deletes a vertex u from a scenario in the sense that u is deleted from the extension but added to the graph being extended. We also need a notation for deleting a vertex completely from a scenario, i. e. ignoring some vertex of the extension.

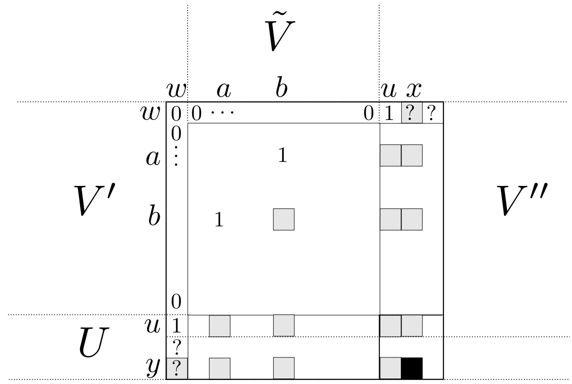

Figure 14: Symmetric Gaussian elimination using $\tilde{V}$ (including steps such as eliminating with edge ab) and eliminating with edge wu can be swapped without changing the result.

Definition 5.7. *Let $s = (s^{U\times V'}, s^{U\times U})$ be a scenario of an extension U and $u \in U$. Then $s_{\text{ignore}}(s, u)$ is the scenario obtained from s in the following way: Delete the u-components from the elements of $s^{U\times V'}$ to obtain s_1. Choose the minimum (according to the vertex order) basis s_1' for the span of s_1 from the elements of s_1 using standard Gaussian elimination. Delete the u-column and u-row from $s^{U\times U}$ to obtain s_2. We define $s_{\text{ignore}}(s, u) = (s_1', s_2)$.*

The following lemma is used in the proof of Lemma 5.5.

Lemma 5.8. *Let $G = (V, E)$ be a graph, $u \in U \subseteq V$ and $G' = G[V']$ a subgraph of G which may be extended by U and $u > v'$ for all $v' \in V'$. Let w be the minimum vertex of V' and assume that u is the minimum neighbor of w (which implies that w has no neighbor in V'). Let $V'' = V' \cup \{u\}$, $\tilde{V} = V' \setminus \{w\}$ and M be the adjacency matrix of $G[V' \cup U]$ (cf. Figure 14). Then the following two sequences of operations on M lead to the same result:*

1. *Symmetric Gaussian elimination on M using V'', i.e. first the elimination step using edge wu and then the elimination steps using $\tilde{V}$.*

2. *Symmetric Gaussian elimination on M using V' (i.e. the elimination steps using $\tilde{V}$, as w has no neighbor in V') and after that, on the result, the elimination step using edge wu.*

Proof. Elimination with edge wu will add the u column (row, resp.) to all columns (rows, resp.) which have a 1 in the w-row (column, resp.), and will then eliminate any remaining 1 in the u column (row, resp.). As the V'-part of the w row (column., resp.) is entirely zero, this has no influence on the $\tilde{V} \times \tilde{V}$ submatrix of M. Thus, the only difference between 1. and 2. is whether the elimination step using edge wu is performed before or after symmetric Gaussian elimination using $\tilde{V}$. Also, it is enough to consider the U-columns and U-rows of M. We will ignore the $V' \times V'$ submatrix of M in the following.

We will prove the following: every elimination step using an edge ab (a self loop at a, resp.) in $\tilde{V}$ can be swapped with elimination using wu, i.e. the results of $\rtimes ab \rtimes wu$ and $\rtimes wu \rtimes ab$ ($\rtimes a \rtimes wu$ and $\rtimes wu \rtimes a$, resp.) are equal. Applying this observation repeatedly proves the lemma. We only prove the case of an edge ab in $\tilde{V}$, the case of a self loop at a in $\tilde{V}$ can be dealt with similarly.

Let ab an edge in $\tilde{V}$. First, let us consider the column and rows of a, b, w and u. It is not hard to see, that, no matter whether we use first ab for elimination and then wu or vice versa, in the end these columns will consist entirely of zeros, except for (u, w), (w, u), (a, b), (b, a). Thus, it is sufficient to examine the effect of both elimination steps on entries (y, x) with $\{x, y\} \cap \{a, b, u, w\} = \emptyset$, cf. Figure 14.

Let $M^{ab} = M \rtimes ab$ be M after the elimination step using edge ab. Analogously we let $M^{wu} = M \rtimes wu$, as well as $M^{ab,wu} = M \rtimes ab \rtimes wu$ and $M^{wu,ab} = M \rtimes wu \rtimes ab$. We use small m to denote the entries of these matrices. For instance, $m^{ab,wu}_{yx}$ denotes the entry in row y and column x of $M^{ab,wu}$.

Case "ab first". By Lemma 4.2 we have

$$m^{ab}_{yx} = m_{yx} + m_{ax} \cdot m_{yb} + m_{ya} \cdot m_{bx} + m_{ya} \cdot m_{ax} \cdot m_{bb}.$$

By Lemma 4.2 again, the final value of entry (y, x) is

$$m^{ab,wu}_{yx} = m^{ab}_{yx} + m^{ab}_{wx} \cdot m^{ab}_{yu} + m^{ab}_{yw} \cdot m^{ab}_{ux} + m^{ab}_{wx} \cdot m^{ab}_{yw} \cdot m^{ab}_{uu},$$

where $m^{ab}_{wx} = m_{wx}$ and $m^{ab}_{yw} = m_{yw}$, as the elimination using edge ab does not affect column/row w (cf. Figure 14). Further on, we have

$$\begin{aligned} m^{ab}_{yu} &= m_{yu} + m_{au} \cdot m_{yb} + m_{ya} \cdot m_{bu} + m_{au} \cdot m_{ya} \cdot m_{bb}, \\ m^{ab}_{ux} &= m_{ux} + m_{ax} \cdot m_{ub} + m_{ua} \cdot m_{bx} + m_{ax} \cdot m_{ua} \cdot m_{bb}, \\ m^{ab}_{uu} &= m_{uu} + m_{au} \cdot m_{ub} + m_{ua} \cdot m_{bu} + m_{au} \cdot m_{ua} \cdot m_{bb}, \end{aligned}$$

once more by Lemma 4.2.

Case "wu first". Here we have

$$m^{wu,ab}_{yx} = m^{wu}_{yx} + m^{wu}_{ax} \cdot m^{wu}_{yb} + m^{wu}_{ya} \cdot m^{wu}_{bx} + m^{wu}_{ax} \cdot m^{wu}_{ya} \cdot m^{wu}_{bb},$$

where $m_{bb}^{wu} = m_{bb}$, as the entry (b, b) is not affected by edge elimination using edge wu. For the remaining values we have by Lemma 4.2:

$$
\begin{aligned}
m_{yx}^{wu} &= m_{yx} + m_{wx} \cdot m_{yu} + m_{yw} \cdot m_{ux} + m_{wx} \cdot m_{yw} \cdot m_{uu}, \\
m_{ax}^{wu} &= m_{ax} + m_{wx} \cdot m_{au}, \\
m_{yb}^{wu} &= m_{yb} + m_{yw} \cdot m_{ub}, \\
m_{ya}^{wu} &= m_{ya} + m_{yw} \cdot m_{ua}, \\
m_{bx}^{wu} &= m_{bx} + m_{wx} \cdot m_{bu}.
\end{aligned}
$$

An easy calculation yields that $m_{yx}^{wu,ab} = m_{yx}^{ab,wu}$, which completes the proof. $\square$

6 The Algorithm

Algorithm 2 evaluates the interlace polynomial using a tree decomposition. The input for the algorithm is $G = (V, E)$, the graph of which we want to evaluate the interlace polynomial, and a nice tree decomposition $(\{X_i\}_I, (I, F))$ of G with $O(n)$ nodes, $n = |V|$. In Section 2.1 we discussed how to obtain a nice tree decomposition. Let $k - 1$ be the width of the tree decomposition, i.e. k is the maximum bag size.

6.1 Interlace Polynomial Parts

Algorithm 2 essentially traverses the tree decomposition bottom-up and computes parts $S(i, D, s)$ of the interlace polynomial for each node i. One such part is defined in the following way:

$$
\begin{aligned}
S(i, D, s) = \sum \{ & x_A y_B u^{\mathrm{rk}((G_i \nabla B)[A \cup B])} v^{\mathrm{n}((G_i \nabla B)[A \cup B])} \mid \\
& A, B \subseteq V_i,\ A \cap B = \emptyset,\ \mathrm{scen}(G'[A \cup B, X_i]) = s, \\
& \text{where } G' = G \nabla (B \cup D) \},
\end{aligned} \tag{6.1}
$$

where $D \subseteq X_i$ and s is a scenario of X_i. Recall that we write $\sum \mathcal{S}$ for the sum of all the elements in $\mathcal{S}$ and that V_i is the set of vertices which have been forgotten below node i. Thus, $S(i, D, s)$ is the part of the interlace polynomial of $G[V_i]$ corresponding to D and s.

For every leaf i of the tree decomposition we have $V_i = \emptyset$ and also $X_i = \emptyset$. Thus, in Line 5 of Algorithm 2 we have $D = \emptyset$. Trivially, $\mathrm{scen}(G[\emptyset, \emptyset])$ is the empty scenario. Thus, we have $S(i, \emptyset, ((), ())) = 1$ if i is a leaf.

Algorithm 2 Evaluating the interlace polynomial using a tree decomposition.

Input: Graph G, nice tree decomposition $(\{X_i\}_i, (I, F))$ of G, k such that any bag X_i of the tree decomposition contains at most k vertices

SUPPLYVERTEXORDER ▷ Algorithm 1
for all nodes i of the tree decomposition, in the order they appear in bottom-up traversal **do**
  for all $D \subseteq X_i$ **do**
    if i is a leaf **then**
      $S(i, D, ((), ())) \leftarrow 1$
    else if i is a join node **then**
      JOIN(i, D)
    else if i is an introduce node **then**
      INTRODUCE(i, D)
    else if i is a forget node **then**
      FORGET(i, D)
    end if
  end for
end for
return $S(\text{root}, \emptyset, ((), ()))$ ▷ $X_{\text{root}} = \emptyset$

At the root node r the bag X_r is empty and all vertices have been forgotten, i.e. $V_r = V$. There is only one part left, $S(r, \emptyset, ((), ()))$, and this is just the interlace polynomial of G.

6.2 Join nodes

Join nodes are handled by Algorithm 3. The correctness follows from

Lemma 6.1. *Let i be a join node with children j_1 and j_2, $D \subseteq X_i$ and s a scenario of X_i. Then*

$$\begin{aligned} S(i, D, s) = \sum \{ & S(j_1, D, s_1) S(j_2, D, s_2) \\ & \mid s_1, s_2 \text{ scenarios of } X_i,\ s_{\text{join}}(s_1, s_2, G \nabla D[X_i]) = s \}. \end{aligned} \tag{6.2}$$

Proof. Recall (6.1) for node i. Every admissible A, B give rise to $A_1 = A \cap V_{j_1}$, $A_2 = A \cap V_{j_2}$, $B_1 = B \cap V_{j_1}$, $B_2 = B \cap V_{j_2}$. $G'[A \cup B]$ is the disjoint union of $G'[A_1 \cup B_1]$ and $G'[A_2 \cup B_2]$. (These graphs are subgraphs of the ones depicted in Figure 8.)

We can apply Lemma 5.1 with G' in the role of G, $A_1 \cup B_1$ in the role of V_1 and $A_2 \cup B_2$ in the role of V_2. This implies that $A \cup B$ takes the role of $V_1 \cup V_2$. Using this it is not hard to argue that every admissible (A, B) in (6.1) corresponds to one pair $((A_1, B_1), (A_2, B_2))$ of the expanded version of (6.2). $\square$

Algorithm 3 Computing the parts at a join node.

```
procedure JOIN(i, D)
    for all scenarios s for |X_i| vertices do
            ▷ i.e., enumerate all pairs s = (s^{X_i×V'}, s^{X_i×X_i}) with s^{X_i×V'} being a list
              of linear independent vectors from {0,1}^{X_i} and s^{X_i×X_i} a symmetric
              X_i × X_i matrix with entries from {0,1} – cf. Definition 3.2
        S(i, D, s) ← 0
    end for
    (j_1, j_2) ← (left child of i, right child of i)
    for all scenarios s_1, s_2 for |X_i| vertices do
        s ← s_join(s_1, s_2, G∇D[X_i])                              ▷ Definition 5.2
        S(i, D, s) ← S(i, D, s) + S(j_1, D, s_1) · S(j_2, D, s_2)
    end for
end procedure
```

6.3 Introduce nodes

Introduce nodes are handled by Algorithm 4, which is based on

Lemma 6.2. *Let i be an introduce node with child j and $X_i = X_j \cup \{a\}$. Let $D \subseteq X_i$ and s a scenario of X_i. Let $D' = D \setminus \{a\}$. Then one of the following cases applies:*

- *If there is a scenario s' of X_j with $s_{\text{introduce}}(s', a, G\nabla D[X_i]) = s$, then we have $S(i, D, s) = S(j, D', s')$.*
- *Otherwise, $S(i, D, s) = 0$.*

Proof. Assume there is some (A, B) such that $\text{scen}(G'[A \cup B, X_i]) = s$. Let $s' = \text{scen}(G'[A \cup B, X_j])$. By Lemma 5.3 it follows $s = s_{\text{introduce}}(s', a, G'[X_i])$. Conversely, Lemma 5.3 also guarantees that for all (A, B) with $\text{scen}(G'[A \cup B, X_j]) = s'$ and $s_{\text{introduce}}(s', a, G'[X_i]) = s$ we have $\text{scen}(G'[A \cup B, X_i]) = s$. □

Algorithm 4 Computing the parts at an introduce node.

procedure INTRODUCE(i, D)
    for all scenarios s for $|X_i|$ vertices **do**
        $S(i, D, s) \leftarrow 0$
    end for
    $j \leftarrow$ child of i
    $a \leftarrow$ vertex being introduced in X_i
    for all scenarios s' for $|X_j|$ vertices **do**
        $s \leftarrow s_{\text{introduce}}(s', a, G\nabla D[X_i])$ ▷ Definition 5.4
        $S(i, D, s) \leftarrow S(j, D \setminus \{a\}, s')$
    end for
end procedure

6.4 Forget nodes

Finally, let us consider Algorithm 5, which handles forget nodes. It is based on

Lemma 6.3. *Let i be a forget node with child j and $X_j = X_i \cup \{a\}$. Let $D \subseteq X_i$ and s a scenario of X_i. Then*

$$\begin{aligned} S(i, D, s) = &\sum \big\{S(j, D, s') \mid s' \text{ scenario of } X_j \text{ with } s_{\text{ignore}}(s', a) = s\big\} \\ &+ \sum \big\{x_a u^{\Delta r_{\text{forget}}(s', a, G\nabla D[X_j])} v^{\Delta n_{\text{forget}}(s', a, G\nabla D[X_j])} S(j, D, s') \mid \\ &\qquad s' \text{ scenario of } X_j \text{ with } s_{\text{forget}}(s', a, G\nabla D[X_j]) = s\big\} \\ &+ \sum \big\{y_a u^{\Delta r_{\text{forget}}(s', a, G\nabla D'[X_j])} v^{\Delta n_{\text{forget}}(s', a, G\nabla D'[X_j])} S(j, D', s') \mid \\ &\qquad s' \text{ scenario of } X_j \\ &\qquad \text{with } s_{\text{forget}}(s', a, G\nabla D'[X_j]) = s,\ D' = D \cup \{a\}\big\}. \end{aligned} \tag{6.3}$$

Proof. We use (6.1) again. Let (A, B) admissible. There are three cases: (1) $a \notin A \cup B$, (2) $a \in A$ and (3) $a \in B$. In case (1), the term corresponding to (A, B) is contained in the first sum in (6.3). In case (2) we obtain the term corresponding to (A, B) from the second sum in (6.3), where we use Lemma 5.5 and multiply by x_a to represent the fact that a is in A. We also multiply by some power of u and v depending on the rank (nullity, resp.) difference with vs. without a in the extension. Case (3) is similar, but we also have to use D' instead of D as in this case a belongs to B and thus the self loop at a is toggled. □

6.5 Running time

We start with a nice tree decomposition with $O(n)$ nodes. Recall that k is the maximum bag size of the tree decomposition. To obtain the vertex order (Algorithm 1) $O(n) \cdot \mathsf{poly}(k)$ steps are sufficient.

The running time of Algorithm 2 can be analyzed as follows. The i loop is executed $O(n)$ times, as there are $O(n)$ nodes in the tree decomposition. There are at most 2^k sets $D \subseteq X_i$ for every node i. There are at most $2^{(3k+1)k/2}$ scenarios for k vertices (Lemma 3.3). The join case (Algorithm 3) sums over *pairs* of scenarios and thus dominates the running time of the introduce (Algorithm 4) and forget (Algorithm 5) case. In the join case, we have to sum over at most $(2^{(3k+1)k/2})^2$ pairs (s_1, s_2). Converting the scenarios (Line 10 of Algorithm 3, Line 8 of Algorithm 4, and Lines 8, 11 and 15 of Algorithm 5) takes time polynomial in k, as we have shown in Section 5. Thus, the running time of Algorithm 2 is at most

$$O(n) \cdot 2^k \cdot (2^{(3k+1)k/2})^2 \cdot \mathsf{poly}(k),$$

Algorithm 5 Computing the parts at a forget node.

procedure FORGET(i, D)
    for all scenarios s for $|X_i|$ vertices **do**
        $S(i, D, s) \leftarrow 0$
    end for
    $j \leftarrow$ child of i
    $a \leftarrow$ vertex being forgotten in X_i
    for all scenarios s' for $|X_j|$ vertices **do**
        $s \leftarrow s_{\text{ignore}}(s', a)$ ▷ Definition 5.7
        $S(i, D, s) \leftarrow S(i, D, s) + S(j, D, s')$
        $G' \leftarrow G \nabla D[X_j]$
        $s \leftarrow s_{\text{forget}}(s', a, G')$ ▷ Definition 5.6
        $S(i, D, s) \leftarrow S(i, D, s) + x_a u^{\Delta r_{\text{forget}}(s', a, G')} v^{\Delta n_{\text{forget}}(s', a, G')} S(j, D, s')$
        $D' \leftarrow D \cup \{a\}$
        $G' \leftarrow G \nabla D'[X_j]$
        $s \leftarrow s_{\text{forget}}(s', a, G')$
        $S(i, D, s) \leftarrow S(i, D, s) + y_a u^{\Delta r_{\text{forget}}(s', a, G')} v^{\Delta n_{\text{forget}}(s', a, G')} S(j, D', s')$
    end for
end procedure

if we assume that arithmetic operations such as addition and multiplication (of numbers) can be performed in one time step. The degree of the interlace polynomial is at most n in every variable (cf. Definition 2.1). This leads to the following result.

Theorem 6.4. *Let $G = (V, E)$ be a graph with n vertices. Let a nice tree decomposition of G with $O(n)$ nodes and width k be given, as well as numbers u, v and, for each $a \in V$, x_a, y_a. Then Algorithm 2 evaluates the multivariate interlace polynomial $C(G)$ at $((x_a)_{a\in V}, (y_a)_{a\in V}, u, v)$ using $2^{3k^2+O(k)} \cdot n$ arithmetic operations. If the bit length of u, v, and $x_a, y_a, a \in V$, is at most ℓ, the operands occurring during the computation are of bit length $O(\ell n)$.*

To evaluate the interlace polynomial of Arratia et al. [ABS04b], which does not use self loop toggling in its definition, we do not need parameter D in (6.1) and the D-loop in Algorithm 2. This simplifies the algorithm a bit. The running time is also reduced, but only by a factor $\leq 2^k$ and thus it is still $2^{3k^2+O(k)}n$.

If we consider path decompositions (see, for example, [Bod98]) instead of tree decompositions, we have no join nodes. Thus, for graphs of bounded pathwidth, we get a result similar to Theorem 6.4 but with running time reduced to $2^{1.5k^2+O(k)} \cdot n$.

7 Variants of the algorithm

7.1 Evaluation vs. computation

The main motivation for our algorithm is *evaluation* of the multivariate interlace polynomial: We are given numerical values for the variables x_a, y_a, u, v, an n-vertex graph G and a nice tree decomposition of G. From this, we want to compute the numerical value $C(G; (x_a)_{a\in V}, (y_a)_{a\in V}, u, v)$. Our algorithm solves this problem as described above.

Another problem one might be interested in is the *computation* of the interlace polynomial: Given G, output a description of the polynomial $C(G)$, which is a polynomial over the indeterminates $\{x_a, y_a \mid a \in V\} \cup \{u, v\}$. As the number of monomials of $C(G)$ is exponential in n, there is no algorithm with running time polynomial in n that computes the multivariate interlace polynomial if we represent $C(G)$ as a list of the coefficients of all the monomials. However, there are other ways of representing polynomials, for example arithmetic formulas and arithmetic circuits, which are considered in algebraic complexity theory [BCS97].

An *arithmetic circuit* is a directed graph with nodes of indegree 0 or 2. Nodes with indegree 0 are inputs and labeled by a constant or a variable. They compute the polynomial they are labeled with. Nodes with indegree two are labeled with plus or times and compute the sum (product, resp.) of their children. We say that a circuit computes a polynomial if it computes it at one of its nodes.

If one accepts arithmetic circuits as a compact way to describe polynomials, then our algorithm actually *computes* the multivariate interlace polynomial: Use Algorithm 2 as a procedure to create an arithmetic circuit for the polynomial $C(G)$ in the following way. Start with a circuit with inputs x_a and y_a for each $a \in V$, as well as inputs for u, v, 0, and 1. For each operation of the algorithm of Section 6 using the "parts" $S(i, D, S)$, add gates that implement this operation. In this way, the algorithm creates an arithmetic circuit $\mathcal{C}$ of size $2^{3k^2+O(k)}n$ that computes $C(G)$.

In the following two subsections, we use this point of view for parallel evaluation and for computation of d-truncations of the multivariate interlace polynomial.

7.2 Parallelization

In this subsection we discuss a way to parallelize our algorithm. We do this using two operations on the tree decomposition: (1) removing all leaves and (2) contracting every path with more than one node. Our approach is not new but a variation of standard methods [Lei92, Section 2.6.1], [JaJ92, Section 3.3].

To describe the operations, we need some formalism. We use vectors σ to collect the parts of the interlace polynomial which are computed. For each node i we define the vector $\sigma_i = (S(i, D, s) \mid D \subseteq X_i, s \text{ scenario of } X_i)$, where the order of the components of the vector is fixed appropriately. We call σ_i the "output" of node i. We call nodes with one child 1-nodes and nodes with two children 2-nodes. Nodes without children are leaves. Every 1-node has one input vector σ_j which is the output of its child, every 2-node has two input vectors which are the output vectors of its children. By definition, for leaves the input and the output is identical.

With each 1-node i with child j we associate a matrix A_i. The computation of the 1-node i is $\sigma_i = A_i \sigma_j$. For an introduce node i with child j, by Lemma 6.2 we trivially can write $\sigma_i = A_i \sigma_j$ for some matrix A_i. The entries of A_i are either 0 or 1. Now let i be a forget node with child j. Consider (6.3). Note that in each of the three sums, the question, which $S(j, D, s')$ ($S(j, D', s')$, resp.) are used, i. e. over which (D, s') ((D', s'), resp.) is summed, can be answered considering only $G[X_j]$ and the involved scenarios. Thus, we can compute from this a matrix A_i with $\sigma_i = A_i \sigma_j$, too. The entries of A_i are 0, 1, $x_a u^l v^{2-l}$ or $y_a u^l v^{2-l}$, where $l \in \{0, 1, 2\}$.

Consider a 2-node i with children j_1 and j_2. The computation performed at i is

$$\sigma_i(D, s) = \sum \sigma_{j_1}(D, s_1) \sigma_{j_2}(D, s_2), \tag{7.1}$$

where the sum is taken over the same elements as in (6.2).

The parallel computation of the interlace polynomial works as follows. We start with the nice tree decomposition of the input graph with $O(n)$ nodes and an arithmetic circuit of constant depth which computes σ_i for all leaves i of the tree decomposition and A_i for all matrices associated with any node i of the tree decomposition. Then we reduce the tree underlying the tree decomposition step by step. Every time we reduce the tree, we extend the arithmetic circuit such that the above invariant is preserved.

We initialize the arithmetic circuit as follows: We insert the constants 0 and 1, u, v and for every vertex a of G we insert x_a and y_a. Then we produce all entries of all matrices associated with any node of the tree decomposition in parallel. This takes constant depth.

We repeat the following operations on the tree decomposition until it consists only of one leave: (1) contract all paths of 1-nodes and (2) remove all leaves.

Path contraction works as follows. For a sequence $i_1, i_2, \dots, i_\ell$ of 1-nodes we have $\sigma_{i_\ell} = A_{i_\ell} \cdot \ldots \cdot A_{i_1} \sigma_j$, where σ_j is the input of node i_1. Thus, we can substitute the sequence by one node which has $\tilde{A} = A_{i_\ell} \cdot \ldots \cdot A_{i_1}$ associated with it and gets σ_j as input. The depth of computing the matrix product in parallel is $\Theta(\log \ell)$. Thus a step contracting any number of disjoint 1-nodes paths of length $\leq \ell$ increases the depth of the arithmetic circuit by $\Theta(\log \ell)$.

Now we come to *removal of leaves.* By this we mean the following: Let L be the set of all leaves of the tree decomposition. Remove the elements of L distinguishing the following cases: (1) node i has two children j_1 and j_2 which are both leaves, (2) node i has two children j_1 and j_2, one of which is a leaf (j_1, say) whereas the other is not, and (3) node i has one child j which is a leaf. To handle case (1) we introduce a level with multiplications and a level with additions to perform (7.1). This increases the depth by 2. In case (2) node i becomes a 1-node: The $\sigma_{j_1}(D, s)$ in (7.1) become coefficients of a new matrix $\tilde{A}$ associated to i. As by the invariant, the arithmetic circuit already computes the $\sigma_{j_1}(D, s)$, we do not need any new gates and depth is not increased. For case (3) we have to implement the matrix multiplication $A_i \sigma_j$ to compute σ_i. This increases the depth by a constant. Thus, removing all leaves in L increases the depth only by a constant.

After performing all possible path contractions, the number of 1-nodes is at most two times the number of 2-nodes. Thus, at least $1/4$ of the nodes are leaves. This implies that the following removal of leaves decreases the number of nodes of the tree decomposition by a factor of at least $1/4$. Thus, after $O(\log n)$ steps the tree decomposition is reduced to a single leave. In each step the depth increases by at most $O(\log n)$, which gives a $O(\log^2 n)$ bound on the depth of the constructed arithmetic circuit.

7.3 Computation of the coefficients

As discussed in Section 7.1, our algorithm can be used to create an arithmetic circuit $\mathcal{C}$ of size $2^{3k^2+O(k)}n$ that computes $C(G)$ for an n-vertex graph G with appropriate tree decomposition of width k. Now one can apply standard techniques to convert $\mathcal{C}$ into a procedure computing some of the coefficients of $C(G)$.

Let us elaborate this for an example, the computation of the d-truncation of the multivariate interlace polynomial. Courcelle defines the d-truncation [Cou08, Section 5] of a multivariate polynomial as follows. The *quasi-degree* of a monomial is the number of vertices that index its indeterminates. As the G-indexed part of the monomials of the multivariate interlace polynomial are multilinear, the quasi-degree of a monomial of $C(G)$ is the degree of its G-indexed part. For example, the quasi-degree of the monomial $x_A y_B u^r v^s$ is $|A| + |B|$. The *d-truncation* $P(G)|d$ of a polynomial $P(G)$ is the sum of its monomials of quasi-degree at most d. Let $\mathcal{M}$ be a set of monomials. If

$$f = \sum_{m \in \mathcal{M}} a_m m$$

is a polynomial and $\mathcal{M}' \subseteq \mathcal{M}$, we set

$$f|\mathcal{M}' = \sum_{m \in \mathcal{M}'} a_m m.$$

As we want to use a result on fast multivariate polynomial multiplication which uses computation trees [BCS97, Section 4.4] as model of computation, we also formulate our result in this model. In addition to the arithmetic operations (addition, multiplication, division), also comparisons are allowed in this model. Each of these operations is counted as one step.

Theorem 7.1 ([LS03, Theorem 1])**.** *Consider polynomials over the indeterminates $x_1, \ldots, x_n$. Let d be a positive integer, and $\mathcal{D}$ the monomials of degree at most d. Let f, g be two polynomials. Then, assuming the coefficients of $f|\mathcal{D}$ and $g|\mathcal{D}$ are given, the coefficients of $(f \cdot g)|\mathcal{D}$ can be computed using*

$$O(D(\log D)^3 \log(\log D))$$

operations in the computation tree model, where $D = |\mathcal{D}|$.

Corollary 7.2. *Let G be a graph with n vertices. Let a nice tree decomposition of G with width k and $O(n)$ nodes be given. Then the coefficients of all monomials of the d-truncation of $C(G)$ can be computed using*

$$2^{3k^2+O(1)} n^{d(1+o(1))+O(1)}$$

operations in the computation tree model.

Note that the d-truncation of $C(G)$ has more than $\binom{n}{d} \geq n^{d(1-\log d/\log n)}$ monomials.

Proof of Corollary 7.2. Let us fix a d and a graph G with n vertices and treewidth k. We want to compute the coefficients of the d-truncation of $C(G)$. As discussed in Section 7.1, there exists an arithmetic circuit $\mathcal{C}$ of size $2^{k^3+O(k)}n$ computing $C(G)$. We convert every operation $f = g + h$ or $f = g \cdot h$ in $\mathcal{C}$ into a sequence of operations computing the coefficients of each monomial of $f|d$. In this way, we also get the coefficients of $C(G)|d$. To prove the corollary, it is sufficient to show that each operation is converted into at most $n^{d(1+o(1))+O(1)}$ operations.

We start with additions. We convert every addition gate $f = g + h$ in $\mathcal{C}$ into the operations $f_m = g_m + h_m$, $m \in \mathcal{M}$, where $\mathcal{M}$ is an appropriate set of monomials. The monomials of $C(G)|d$ are a subset of $\mathcal{M}$ if $\mathcal{M}$ denotes the set of all monomials over G-indexed variables x and y and ordinary variables u and v such that the quasi-degree is at most d and the degree in u and in v is at most n. We can select a monomial in $\mathcal{M}$ in the following way. First, choose d times either 1 or a variable from $\{x_a, y_a \mid a \in V\}$. Then, choose the exponent of u and v from $\{0, 1, \ldots, n\}$. Thus, we have

$$|\mathcal{M}| \leq (2n+1)^d(n+1)^2 = n^{d\left(1+\frac{O(1)}{\log n}\right)+O(1)}. \tag{7.2}$$

As we convert every addition from $\mathcal{C}$ into $|\mathcal{M}|$ operations, the claimed bound of the corollary is fulfilled.

Now let us consider multiplications, i.e. let $f = g \cdot h$ be a multiplication gate in $\mathcal{C}$. We use fast multivariate polynomial multiplication for the G-indexed variables and the school method for the ordinary variables. To this end, we fix the u- and v-part of the monomial, i.e. we choose d_u and d_v, $0 \leq d_u, d_v \leq n$. We want to compute the coefficients of the monomials m of f with $\deg_u(m) = d_u$ and $\deg_v(m) = d_v$. Choose nonnegative integers $d_{u,g}, d_{u,h}, d_{v,g}, d_{v,h}$ such that $d_{u,g} + d_{u,h} = d_u$ and $d_{v,g} + d_{v,h} = d_v$. Let

$$\mathcal{D} = \{x_A y_B \mid A, B \subseteq V(G), |A| + |B| \leq d\}.$$

We can assume that we have already computed all coefficients of $\tilde{g} := g|u^{d_{u,g}}v^{d_{v,g}}\mathcal{D}$ and $\tilde{h} := h|u^{d_{u,h}}v^{d_{v,h}}\mathcal{D}$. (Here, an expression of the form $u^a v^b \mathcal{D}$ denotes the set $\{u^a v^b m \mid m \in \mathcal{D}\}$.) By Theorem 7.1, we can compute all coefficients of the product $\tilde{g} \cdot \tilde{h}$ using

$$O(|\mathcal{D}|(\log|\mathcal{D}|)^3 \log\log|\mathcal{D}|) = n^{d\left(1+\frac{O(1)}{\log n}\right)+\frac{O(\log\log n)}{\log n}}$$

operations, as $|\mathcal{D}| \leq (2n+1)^d \leq n^{d\left(1+\frac{\log 3}{\log n}\right)}$. We do this for every choice of $d_{u,g}$, $d_{u,h}$, $d_{v,g}$, and $d_{v,h}$. As these are at most $(n+1)^2$ many, this takes $n^{d\left(1+\frac{O(1)}{\log n}\right)+O(1)}$ steps. Adding the results monomial-wise needs at most $|\mathcal{D}|(n+1)^2 = n^{d\left(1+\frac{O(1)}{\log n}\right)+O(1)}$ additions and yields the coefficients of $f|u^{d_u}v^{d_v}\mathcal{D}$. We do this for all $(n+1)^2$ choices of d_u and d_v to obtain the coefficients of all monomials of the d-truncation of f. Thus, each multiplication in $\mathcal{C}$ is converted into $n^{d\left(1+\frac{O(1)}{\log n}\right)+O(1)}$ operations. This, again, is within the claimed bound of the corollary. □

8 Further questions

If we consider graphs of bounded cliquewidth instead of treewidth, so called k-expressions take the role of tree decompositions. Our concept of scenarios is tailor-made for tree decompositions and does not work with k-expressions. Is there a linear algebra approach, possibly similar to the one we presented in this work, to compute the interlace polynomial using k-expressions?

The notion of rankwidth, which is related to cliquewidth [Oum05, OS06], is defined using the $GF(2)$-rank of some matrices derived from a graph. Furthermore, local complementation is studied in the context of the interlace polynomial as well as in the context of rankwidth [Oum05, Section 2]. Thus, it seems to be possible that rank decompositions support the computation of the interlace polynomial very nicely. We have not investigated this question in detail and leave it as a direction for further research.

Acknowledgement

We would like to thank Bruno Courcelle and the anonymous referees for their helpful comments on previous versions of the paper, which led to a reduction of the running time bound.

References

[ABCS00] Richard Arratia, Béla Bollobás, Don Coppersmith, and Gregory B. Sorkin. Euler circuits and DNA sequencing by hybridization. *Discrete Appl. Math.*, 104(1-3):63–96, 2000.

[ABS04a] Richard Arratia, Béla Bollobás, and Gregory B. Sorkin. The interlace polynomial of a graph. *J. Comb. Theory Ser. B*, 92(2):199–233, 2004.

[ABS04b] Richard Arratia, Béla Bollobás, and Gregory B. Sorkin. A two-variable interlace polynomial. *Combinatorica*, 24(4):567–584, 2004.

[AGM08] Ilia Averbouch, Benny Godlin, and Johann A. Makowsky. A most general edge elimination polynomial. In Hajo Broersma, Thomas Erlebach, Tom Friedetzky, and Daniël Paulusma, editors, *WG*, volume 5344 of *Lecture Notes in Computer Science*, pages 31–42, 2008.

[And98] Artur Andrzejak. An algorithm for the Tutte polynomials of graphs of bounded treewidth. *Discrete Mathematics*, 190(1-3):39–54, 1998.

[AvdH04] Martin Aigner and Hein van der Holst. Interlace polynomials. *Linear Algebra and its Applications*, 377:11–30, 2004.

[BBD97] D. Bénard, A. Bouchet, and A. Duchamp. On the Martin and Tutte polynomials. Technical report, Département d'Infornmatique, Université du Maine, Le Mans, France, 1997.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften / A series of comprehensive studies in mathematics*. Springer, 1997.

[BH08] Markus Bläser and Christian Hoffmann. On the complexity of the interlace polynomial. In Susanne Albers and Pascal Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, pages 97–108, Dagstuhl, Germany, 2008. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[BK08] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1 – 45, 1998.

[Bol02] Béla Bollobás. Evaluations of the circuit partition polynomial. *J. Comb. Theory Ser. B*, 85(2):261–268, 2002.

[Bou87] André Bouchet. Isotropic systems. *Eur. J. Comb.*, 8(3):231–244, 1987.

[Bou88] André Bouchet. Graphic presentations of isotropic systems. *J. Comb. Theory Ser. B*, 45(1):58–76, 1988.

[Bou91] André Bouchet. Tutte Martin polynomials and orienting vectors of isotropic systems. *Graphs Combin.*, 7:235–252, 1991.

[Bou05] André Bouchet. Graph polynomials derived from Tutte–Martin polynomials. *Discrete Mathematics*, 302(1-3):32–38, 2005.

[BR99] Béla Bollobás and Oliver Riordan. A Tutte polynomial for coloured graphs. *Comb. Probab. Comput.*, 8(1-2):45–93, 1999.

[CiO07] Bruno Courcelle and Sang il Oum. Vertex-minors, monadic second-order logic, and a conjecture by seese. *J. Comb. Theory, Ser. B*, 97(1):91–126, 2007.

[CMR01] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[Cou08] Bruno Courcelle. A multivariate interlace polynomial and its computation for graphs of bounded clique-width. *The Electronic Journal of Combinatorics*, 15(1), 2008.

[DP08] Lars Eirik Danielsen and Matthew G. Parker. Interlace polynomials: Enumeration, unimodality, and connections to codes, 2008. Preprint, arXiv:0804.2576v1.

[EM98] Joanna A. Ellis-Monaghan. New results for the Martin polynomial. *J. Comb. Theory Ser. B*, 74(2):326–352, 1998.

[EM99] Joanna A. Ellis-Monaghan. Martin polynomial miscellanea. In *Proceedings of the 30th Southeastern International Conference on Combinatorics, Graph Theory, and Computing*, pages 19–31, Boca Raton, FL, 1999.

[EMS06] Joanna A. Ellis-Monaghan and Irasema Sarmiento. Isotropic systems and the interlace polynomial, 2006. Preprint, arXiv:math/0606641v2.

[EMS07] Joanna A. Ellis-Monaghan and Irasema Sarmiento. Distance hereditary graphs and the interlace polynomial. *Comb. Probab. Comput.*, 16(6):947–973, 2007.

[Jae88] François Jaeger. On Tutte polynomials and cycles of plane graphs. *J. Comb. Theory Ser. B*, 44(2):127–146, 1988.

[JaJ92] Joseph JaJa. *Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[Klo94] T. Kloks. *Treewidth. Computations and Approximations.*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1992.

[LS03] Grégoire Lecerf and Éric Schost. Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal*, 5(1), 2003.

[LV81] M. Las Vergnas. Eulerian circuits of 4-valent graphs imbedded in surfaces. In *Algebraic Methods in Graph Theory, Szeged, Hungary, 1978*, volume 25 of *Colloq. Math. Soc. János Bolyai*, pages 451–477, North-Holland, Amsterdam, 1981.

[LV83] M. Las Vergnas. Le polynôme de martin d'un graphe eulerian. *Ann. Discrete Math*, 17:397–411, 1983.

[LV88] M. Las Vergnas. On the evaluation at (3,3) of the Tutte polynomial of a graph. *J. Comb. Theory Ser. B*, 45(3):367–372, 1988.

[Mar77] P. Martin. *Enumérations Eulériennes dans le multigraphes et invariants de Tutte–Grothendieck*. PhD thesis, Grenoble, France, 1977.

[Neg87] S. Negami. Polynomial invariants of graphs. *Trans. Am. Math. Soc.*, 299:601–622, 1987.

[Nob98] S. D. Noble. Evaluating the Tutte polynomial for graphs of bounded tree-width. *Combinatorics, Probability & Computing*, 7(3):307–321, 1998.

[OS06] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

[Oum05] Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005.

[RP06] Constanza Riera and Matthew G. Parker. One and two-variable interlace polynomials: A spectral interpretation. In *Coding and Cryptography. International Workshop, WCC 2005, Bergen, Norway, March 14-18, 2005*, volume 3969 of *Lecture Notes in Computer Science*, pages 397–411, Berlin / Heidelberg, 2006. Springer.

[Tra08] Lorenzo Traldi. Weighted interlace polynomials, 2008. Preprint, arXiv:0808.1888v1.

[Tra09] Lorenzo Traldi. Binary nullity, Euler circuits and interlace polynomials, 2009. Preprint, arXiv:0903.4405v1.