

Fast Evaluation of Interlace Polynomials on Graphs of Bounded Treewidth

Markus Bläser and Christian Hoffmann

Saarland University, Germany

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 [1], Arratia, Bollobás, and Sorkin defined a graph polynomial which they called interlace polynomial [2]. It turned out that the interlace polynomial is related [2, 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 [3] and generalized by Las Vergnas [4]. Further work on the Martin polynomial has been pursued [5, 6, 7, 8, 9, 10], including a generalization to isotropic systems [11, 12, 13, 14]. 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 [15] 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 [2]. This operation is a composition of local complementations to neighbour vertices (see [16], 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 [17].

The interlace polynomial can also be defined by a closed expression using the $GF(2)$ -rank of adjacency matrices [16, 18, 19]. 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 [20] (see Definition 1 below) as it subsumes the two-variable interlace polynomial of Arratia, Bollobás, and Sorkin [21] and the weighted versions of Traldi [22], as well as the interlace polynomials defined by Aigner and van der Holst [16]. Furthermore, the interlace polynomials

$Q(x, y)$ and Q_n^{HN} , which have emerged from a spectral view on the interlace polynomials [23], are also special cases of Courcelle’s multivariate interlace polynomial.

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 [24], it is fixed parameter tractable with cliquewidth as parameter [20, 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 [20, Page 34]. In the case of the interlace polynomial, this formula involves two quantifier alternations [20, Lemma 24], [25]. If a graph has tree width k , its cliquewidth is bounded by 2^{k+1} [26]. 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, trees, can be evaluated in polynomial time, which also has been observed by Traldi [22].)

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 [27]. 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 1). 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 13). The algorithm can be implemented in parallel using depth polylogarithmic in n , see the full version of this work [28, Section 7]. Apart from evaluating the interlace polynomial, our approach can also be used to compute coefficients of the interlace polynomial [28, Section 7], for example so called d -truncations [20, Section 5]. Our approach is not via logic but via the $GF(2)$ -rank of adjacency matrices, which is specific to the interlace polynomial.

Obstacles. It has been noticed that the Tutte polynomial and the interlace polynomial are similar in some respect. These similarities suggest that evaluating the interlace polynomial using tree decompositions might work completely analogously to the respective approaches for the Tutte polynomial [29, 30]. This is not the case because of the following facts that distinguish the interlace polynomial from the Tutte polynomial: First, the graph operation that is used in the recursive definition of the interlace polynomial is not compliant with tree decomposition (i.e. can increase the treewidth). Second, the recurrence for the multivariate interlace polynomial is very complicated. Third, the “behaviour” of the rank involved in the definition of the interlace polynomial is not as be-

nign as the rank used for the Tutte polynomial (cf. also the beginning of Sect. 3). We discuss these issues in more detail in the full version of this work [28].

Outline. 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 Sect. 3 we give the general idea of our approach. Section 4 and 5 provide technical details, and in Sect. 6 we describe our algorithm. Due to space limitations we have to omit some details and most of the proofs. These can be found in the full version of this work [28].

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.

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 $\text{rk}(G)$ for the rank of the adjacency matrix of the graph G . The nullity (or co-rank) of an $n \times n$ matrix M is $\text{n}(M) = n - \text{rk}(M)$. If G is a graph, we write $\text{n}(G)$ 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 S is a set, we write $\sum S$ for the sum of all the elements in the set.

Definition 1 (Courcelle [20]). *Let $G = (V, E)$ be an undirected graph. The multivariate interlace polynomial is defined as*

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

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

Tree Decompositions. We borrow most of our notation from Bodlaender and Koster [31]. 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 , $\text{tw}(G)$, 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 $2^{O(k^3)}n$ [32].

To evaluate the interlace polynomial we will use *nice* tree decompositions. Note that our definition slightly deviates from the usual one¹. 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 $X_i = X_j \cup \{a\}$.
- Forget: node i has exactly one child j , and there is a vertex $a \in V$ with $X_j = X_i \cup \{a\}$.

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 $O(n) \cdot \text{poly}(k)$ [33, Lemma 13.1.2, 13.1.3].

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

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

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

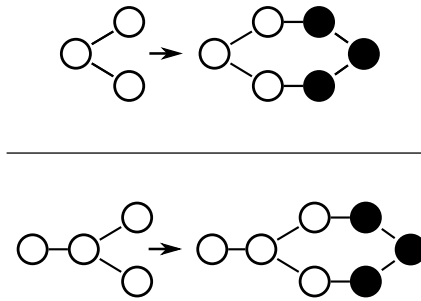
3 Idea

We will now sketch our idea how to evaluate the interlace polynomial. Our approach is dynamic programming. 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

¹ Usually, there is no special restriction on the bag size of the root node, and the leaf nodes contain exactly *one* vertex.

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 situation in the upper half of Fig. 1. If we extend the graph by the black vertices, the rank increases by 2. But in the situation depicted in the lower half of Fig. 1, the *same extension* causes a rank increase by 4.²



	V'								U				
	v ₁	v ₂	v ₃	v ₄	v ₅	v ₆	v ₇	v ₈	v ₉	v ₁₀	v ₁₁	v ₁₂	v ₁₃
v ₁	1												
v ₂		1											
v ₃			1										
v ₄				1							?	?	?
v ₅					1								
v ₆						1							
v ₇							1						
v ₈								1					
v ₉									1				
v ₁₀										1			
v ₁₁											1		
v ₁₂												1	
v ₁₃													1

Fig. 1. Rank over $GF(2)$ of the adjacency matrix may increase by 2 (from 2 to 4, upper half) or by 4 (from 2 to 6, lower half), even if the same extension is used.

Fig. 2. Adjacency matrix of $G[V' \cup U]$ after symmetric Gaussian elimination using V' . Empty entries are zero.

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

Definition 2 (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 $G[V', U]$.

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

² This phenomenon distinguishes the interlace polynomial from the Tutte polynomial. In the case of the Tutte polynomial, the rank increase would be the same in both situations as it only depends on the extension and how it is connected to the graph being extended.

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 5) 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 Sect. 4). The resulting matrix M' is symmetric again and has the same rank as M . Furthermore, M' is of a form as in Fig. 2: 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 $U \times V'$ 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 $\text{poly}(|U|)$ 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 (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 (1) in Sect. 6.

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 4. *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 .*

4 Symmetric Gaussian Elimination

We want to convert adjacency matrices into matrices of a form as in Fig. 2 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' .

Eliminating with a Single Vertex. Assume we are given a graph G , its adjacency matrix M , and a vertex v . A symmetric Gaussian elimination step on M using v is defined in the following way:

- If v is an isolated vertex without a self loop, the result of the elimination step is just M (cf. (1) in Fig. 3).
- If v has a self loop, there is a 1 in the (v, v) -entry of M . We add column v and row v to the column and row of every neighbor of v . This transfers the matrix in a form as depicted as (2) in Fig. 3.
- If v is neither isolated nor has a self loop, there is a neighbor u of v . The elimination step is performed in a similar manner as elimination steps using self loops. The result is of a form as (3) in Fig. 3. (We do not swap columns/rows, as we must keep the vertices in a particular order that is determined by the tree decomposition.)

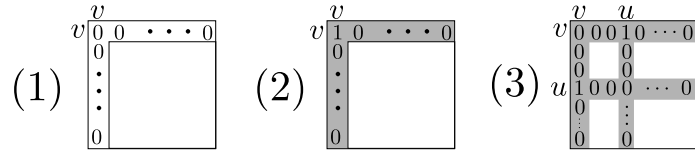


Fig. 3. Effect of a symmetric Gaussian elimination step. Adjacency matrix with isolated unlooped vertex v (1), adjacency matrix after eliminating with a self loop at v (2), adjacency matrix after eliminating with edge vu (3).

Eliminating with a Set of Vertices. Eliminating with a set of vertices V' means that we perform an elimination step as described above for each vertex $v \in V'$. But any edge vu is used for an elimination step only if both, v and u , are in V' . In the setting of Fig. 2, this protects us from using the entries of the $U \times V'$ submatrix for elimination steps.

The order the vertices are processed in is given by a fixed vertex order of the graph computed in the beginning. Also, if an unlooped vertex v has more than one neighbor u in V' , the minimum u with respect to the vertex order is chosen for the elimination step.

The vertex order we use is obtained in the following way. Let a graph G and a nice tree decomposition $(\{X_i\}, T)$ of G be given. We traverse the tree decomposition bottom up. Whenever we reach a forget node that forgets vertex v , we assign the next free position in the vertex order to v . This ensures that for all extended graphs $G[V_i, X_i]$ the vertices in X_i are greater than the vertices in V_i .

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.

Definition 5 (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 $\text{scen}(G[V', U])$ 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 $M'_{U \times V'}$ be the $U \times V'$ submatrix of M' . Consider the column space W of $M'_{U \times V'}$. We can choose a basis of W from the column vectors of $M'_{U \times V'}$. 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 $\text{scen}(G[V', U])$ to be $(s^{U \times V'}, s^{U \times U})$.

5 Scenarios and Nice Tree Decompositions

In this section we will collect lemmas that allow us to compute the parts of a join, forget, and introduce node from the parts of its children nodes (child node, resp.).

Lemma 6 (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 , $\text{scen}(G[V_1, U]) = s_1$, and $\text{scen}(G[V_2, U]) = s_2$, then $\text{scen}(G[V_1 \cup V_2, U]) = s_3$. Moreover, s_3 can be computed from s_1, s_2 and $G[U]$ within $\text{poly}(|U|)$ steps.

Definition 7. In the situation of Lemma 6 we write $s_{\text{join}}(s_1, s_2, G[U])$ for s_3 .

Lemma 8 (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 $\text{scen}(G[V', U]) = s$, then $\text{scen}(G[V', \tilde{U}]) = \tilde{s}$. Moreover, \tilde{s} can be computed from s and $G[\tilde{U}]$ in $\text{poly}(|U|)$ steps.

Definition 9. In the situation of Lemma 8 we write $s_{\text{introduce}}(s, u, G[\tilde{U}])$ for \tilde{s} .

Lemma 10 (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 $\text{scen}(G[V', U]) = s$, then $\text{scen}(G[\tilde{V}, \tilde{U}]) = \tilde{s}$ 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 $\text{poly}(|U|)$ steps, and we have $n = 2 - r$.*

Definition 11. *In the situation of Lemma 10 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 11 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.

Definition 12. *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 $s_{\text{ignore}}(s, u) = (s'_1, s_2)$.*

6 The Algorithm

Algorithm 1 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 Sect. 2 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.

Algorithm 1 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:

$$S(i, D, s) = \sum \{x_A y_B u^{\text{rk}((G_i \nabla B)[A \cup B])} v^{n((G_i \nabla B)[A \cup B])} \mid \begin{array}{l} A, B \subseteq V_i, A \cap B = \emptyset, \text{scen}(G'[A \cup B, X_i]) = s, \\ \text{where } G' = G \nabla (B \cup D) \end{array} \}, \quad (1)$$

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 that 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 . Equation (1) shows that $S(i, \emptyset, ((\cdot), (\cdot))) = 1$ for leaves i . The value $S(r, \emptyset, ((\cdot), (\cdot)))$, where r is the root node, is just the interlace polynomial of G .

The parts of join and forget nodes are computed by Algor. 2 and 3. The parts of an introduce node can be computed similarly.

Algorithm 1 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

- 1: Compute a vertex order as described in Sect. 4
 - 2: **for all** nodes i of the tree decomposition, in the order they appear in bottom-up traversal **do**
 - 3: **for all** $D \subseteq X_i$ **do**
 - 4: **if** i is a leaf **then**
 - 5: $S(i, D, ((), ())) \leftarrow 1$
 - 6: **else if** i is a join node **then**
 - 7: JOIN(i, D)
 - 8: **else if** i is an introduce node **then**
 - 9: INTRODUCE(i, D)
 - 10: **else if** i is a forget node **then**
 - 11: FORGET(i, D)
 - 12: return $S(\text{root}, \emptyset, ((), ()))$ $\triangleright X_{\text{root}} = \emptyset$
-

Algorithm 2 Computing the parts at a join node.

- 1: **procedure** JOIN(i, D)
 - 2: **for all** scenarios s for $|X_i|$ vertices **do**
 - 3: \triangleright i.e., enumerate all pairs $s = (s^{X_i \times V'}, s^{X_i \times X_i})$ with $s^{X_i \times V'}$ being a list
 - 4: of linear independent vectors from $\{0, 1\}^{X_i}$ and $s^{X_i \times X_i}$ a symmetric
 - 5: $X_i \times X_i$ matrix with entries from $\{0, 1\}$ – cf. Def. 3
 - 6: $S(i, D, s) \leftarrow 0$
 - 7: $(j_1, j_2) \leftarrow$ (left child of i , right child of i)
 - 8: **for all** scenarios s_1, s_2 for $|X_i|$ vertices **do**
 - 9: $s \leftarrow s_{\text{join}}(s_1, s_2, G \nabla D[X_i])$ \triangleright Definition 7
 - 10: $S(i, D, s) \leftarrow S(i, D, s) + S(j_1, D, s_1) \cdot S(j_2, D, s_2)$
-

Algorithm 3 Computing the parts at a forget node.

- 1: **procedure** FORGET(i, D)
 - 2: **for all** scenarios s for $|X_i|$ vertices **do**
 - 3: $S(i, D, s) \leftarrow 0$
 - 4: $j \leftarrow$ child of i
 - 5: $a \leftarrow$ vertex being forgotten in X_i
 - 6: **for all** scenarios s' for $|X_j|$ vertices **do**
 - 7: $s \leftarrow s_{\text{ignore}}(s', a)$ \triangleright Definition 12
 - 8: $S(i, D, s) \leftarrow S(i, D, s) + S(j, D, s')$
 - 9: $G' \leftarrow G \nabla D[X_j]$
 - 10: $s \leftarrow s_{\text{forget}}(s', a, G')$ \triangleright Definition 11
 - 11: $S(i, D, s) \leftarrow S(i, D, s) + x_a u^{\Delta r_{\text{forget}}(s', a, G')} v^{\Delta n_{\text{forget}}(s', a, G')} S(j, D, s')$
 - 12: $D' \leftarrow D \cup \{a\}$
 - 13: $G' \leftarrow G \nabla D'[X_j]$
 - 14: $s \leftarrow s_{\text{forget}}(s', a, G')$
 - 15: $S(i, D, s) \leftarrow S(i, D, s) + y_a u^{\Delta r_{\text{forget}}(s', a, G')} v^{\Delta n_{\text{forget}}(s', a, G')} S(j, D', s')$
-

Running Time. There are $O(n)$ nodes i in the tree decomposition and for each of it at most 2^k subsets D of X_i . The number of scenarios (Line 2 of Algor. 2, Lines 2 and 6 of Algor. 3) is dominated by the number of *pairs* of scenarios (Line 8 of Algor. 2). By Lemma 4, there are at most $(2^{(3k+1)k/2})^2$ such pairs. Converting the scenarios (Line 9 of Algorithm 2 and Lines 7, 10 and 14 of Algorithm 3) takes time polynomial in k (Sect. 5). Thus, the running time is at most

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

Theorem 13. *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 1 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 $O(\ell n)$.*

Proof. The time bound stems from the discussion above. The statement about operand length follows as the degree of the interlace polynomial is at most n in each variable.

References

- [1] Arratia, R., Bollobás, B., Coppersmith, D., Sorkin, G.B.: Euler circuits and DNA sequencing by hybridization. *Discrete Appl. Math.* **104**(1-3) (2000) 63–96
- [2] Arratia, R., Bollobás, B., Sorkin, G.B.: The interlace polynomial of a graph. *J. Comb. Theory Ser. B* **92**(2) (2004) 199–233
- [3] Martin, P.: *Énumérations Eulériennes dans le multigraphes et invariants de Tutte–Grothendieck*. PhD thesis, Grenoble, France (1977)
- [4] Las Vergnas, M.: Le polynôme de martin d’un graphe eulerian. *Ann. Discrete Math* **17** (1983) 397–411
- [5] Las Vergnas, M.: 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.*, North-Holland, Amsterdam (1981) 451–477
- [6] Las Vergnas, M.: On the evaluation at (3,3) of the Tutte polynomial of a graph. *J. Comb. Theory Ser. B* **45**(3) (1988) 367–372
- [7] Jaeger, F.: On Tutte polynomials and cycles of plane graphs. *J. Comb. Theory Ser. B* **44**(2) (1988) 127–146
- [8] Ellis-Monaghan, J.A.: New results for the Martin polynomial. *J. Comb. Theory Ser. B* **74**(2) (1998) 326–352
- [9] Ellis-Monaghan, J.A.: Martin polynomial miscellanea. In: *Proceedings of the 30th South-eastern International Conference on Combinatorics, Graph Theory, and Computing*, Boca Raton, FL (1999) 19–31
- [10] Bollobás, B.: Evaluations of the circuit partition polynomial. *J. Comb. Theory Ser. B* **85**(2) (2002) 261–268
- [11] Bouchet, A.: Isotropic systems. *Eur. J. Comb.* **8**(3) (1987) 231–244
- [12] Bouchet, A.: Graphic presentations of isotropic systems. *J. Comb. Theory Ser. B* **45**(1) (1988) 58–76
- [13] Bouchet, A.: Tutte Martin polynomials and orienting vectors of isotropic systems. *Graphs Combin.* **7** (1991) 235–252

- [14] Bénard, D., Bouchet, A., Duchamp, A.: On the Martin and Tutte polynomials. Technical report, Département d'Informatique, Université du Maine, Le Mans, France (1997)
- [15] Ellis-Monaghan, J.A., Sarmiento, I.: Distance hereditary graphs and the interlace polynomial. *Comb. Probab. Comput.* **16**(6) (2007) 947–973
- [16] Aigner, M., van der Holst, H.: Interlace polynomials. *Linear Algebra and its Applications* **377** (2004) 11–30
- [17] Danielsen, L.E., Parker, M.G.: Interlace polynomials: Enumeration, unimodality, and connections to codes (2008) Preprint, arXiv:0804.2576v1.
- [18] Bouchet, A.: Graph polynomials derived from Tutte–Martin polynomials. *Discrete Mathematics* **302**(1-3) (2005) 32–38
- [19] Ellis-Monaghan, J.A., Sarmiento, I.: Isotropic systems and the interlace polynomial (2006) Preprint, arXiv:math/0606641v2.
- [20] Courcelle, B.: A multivariate interlace polynomial and its computation for graphs of bounded clique-width. *The Electronic Journal of Combinatorics* **15**(1) (2008)
- [21] Arratia, R., Bollobás, B., Sorkin, G.B.: A two-variable interlace polynomial. *Combinatorica* **24**(4) (2004) 567–584
- [22] Traldi, L.: Weighted interlace polynomials (2008) Preprint, arXiv:0808.1888v1.
- [23] Riera, C., Parker, M.G.: 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.*, Berlin / Heidelberg, Springer (2006) 397–411
- [24] Bläser, M., Hoffmann, C.: On the complexity of the interlace polynomial. In Albers, S., Weil, P., eds.: *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2008) 97–108
- [25] Courcelle, B., Oum, S.: Vertex-minors, monadic second-order logic, and a conjecture by seese. *J. Comb. Theory, Ser. B* **97**(1) (2007) 91–126
- [26] Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. *Discrete Applied Mathematics* **101**(1-3) (2000) 77–114
- [27] Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics* **108**(1-2) (2001) 23–52
- [28] Bläser, M., Hoffmann, C.: Fast computation of interlace polynomials on graphs of bounded treewidth (2009) Preprint, arXiv:0902.1693.
- [29] Andrzejak, A.: An algorithm for the Tutte polynomials of graphs of bounded treewidth. *Discrete Mathematics* **190**(1-3) (1998) 39–54
- [30] Noble, S.D.: Evaluating the Tutte polynomial for graphs of bounded tree-width. *Combinatorics, Probability & Computing* **7**(3) (1998) 307–321
- [31] Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. *Comput. J.* **51**(3) (2008) 255–269
- [32] Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing* **25**(6) (1996) 1305–1317
- [33] Kloks, T.: Treewidth. Computations and Approximations. Volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin (1994)