Computational Complexity of Graph Polynomials

Christian Hoffmann

Saarbrücken, 2010

Dissertation zur Erlangung des Grades des
Doktors der Naturwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten der
Universität des Saarlandes

Tag des Kolloquiums: 20. August 2010

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät I:
Prof. Dr. Holger Hermanns

Mitglieder des Prüfungsausschusses:
Prof. Dr. Matthias Hein (Vorsitzender)
Prof. Dr. Markus Bläser (Betreuer und Berichterstatter)
Prof. Dr. Johann A. Makowsky (Berichterstatter)
Dr. Alexey Pospelov

Summary

The thesis provides hardness and algorithmic results for graph polynomials.

We observe VNP-completeness of the interlace polynomial, and we prove VNP-completeness of almost all q-restrictions of $Z(G; q, \mathbf{x})$, the multivariate Tutte polynomial.

Using graph transformations, we obtain point-to-point reductions for graph polynomials. We develop two general methods: Vertex/edge cloning and, more general, uniform local graph transformations. These methods unify known and new hardness-of-evaluation results for graph polynomials. We apply both methods to several examples. We show that, almost everywhere, it is #P-hard to evaluate the two-variable interlace polynomial and the (normal as well as extended) bivariate chromatic polynomial. "Almost everywhere" means that the dimension of the set of exceptional points is strictly less than the dimension of the domain of the graph polynomial. We also give an inapproximability result for evaluation of the independent set polynomial. Providing a new family of reductions for the interlace polynomial that increases the instance size only polylogarithmically, we obtain an $\exp(\Omega(n/\log^3 n))$ time lower bound for evaluation of the independent set polynomial under a counting version of the exponential time hypothesis.

We observe that the extended bivariate chromatic polynomial can be computed in vertex-exponential time.

We devise a means to compute the interlace polynomial using tree decompositions. This enables a parameterized algorithm to evaluate the interlace polynomial in time linear in the size of the graph and single-exponential in the treewidth. We give several versions of the algorithm, including a parallel one and a faster way to compute the interlace polynomial of *any* graph. Finally, we propose two faster algorithms to compute/evaluate the interlace polynomial in special cases.

Zusammenfassung

Diese Arbeit beinhaltet Härteresultate und Algorithmen für Graphpolynome.

Wir stellen zunächst fest, dass das Interlacepolynom VNP-vollständig ist, und wir zeigen die VNP-Vollständigkeit fast aller q-Restriktionen des multivariaten Tutte-Polynoms $Z(G; q, \mathbf{x})$.

Unter Verwendung von Graphtransformationen erhalten wir Punkt-zu-Punkt-Reduktionen für Graphpolynome. Dabei entwickeln wir auch zwei allgemeine Methoden: Das Klonen von Knoten bzw. Kanten und, allgemeiner, uniforme lokale Graphtransformationen. Beide Methoden vereinheitlichen bekannte und neue Härteresultate für das Auswerten von Graphpolynomen. Wir wenden beide Methoden auf verschiedene Beispiele an. Wir zeigen, dass es fast überall #P-schwer ist, das Interlacepolynom in zwei Variablen bzw. das (normale oder erweiterte) bivariate chromatische Polynom auszuwerten. „Fast überall“ heißt hier: überall, außer auf einer Ausnahmemenge, deren Dimension um mindestens eins kleiner ist als der Definitionsbereich des Graphpolynoms. Wir zeigen auch, dass näherungsweises Auswerten des Independent-Set-Polynoms schwer ist. Wir entwickeln eine neue Familie von Reduktionen für das Interlacepolynom, die die Instanz nur polylogarithmisch vergrößert. Damit zeigen wir, unter Annahme einer Variante der Exponentialzeit-Hypothese, dass das Auswerten des Independent-Set-Polynoms fast überall Zeit $\exp(\Omega(n/\log^3 n))$ benötigt.

Wir stellen fest, dass das erweiterte bivariate chromatische Polynom in Zeit exponentiell in der Knotenzahl berechnet werden kann.

Wir entwickeln ein Mittel, um das Interlacepolynom mit Hilfe von Baumzerlegungen zu berechnen. Das führt zu einem parametrisierten Algorithmus zum Auswerten des Interlacepolynoms mit Laufzeit linear in der Anzahl der Knoten und einfach exponentiell in der Weite der gegebenen Baumzerlegung. Wir diskutieren verschiedene Varianten dieses Algorithmus, einschließlich Parallelisierung und einer Möglichkeit, das Interlacepolynom *jedes* Graphen asymptotisch schneller zu berechnen. Schließlich geben wir zwei schnellere Algorithmen an, die das Interlacepolynom in speziellen Situationen berechnen.

Acknowledgments

I would like to thank my advisor Markus Bläser very much for his guidance and advice throughout my time as a PhD student. In particular, I would like to thank him for his comments on previous versions of this work, which improved it a lot.

I had motivating and helpful discussions with Johann A. Makowsky. In particular, he drew my attention to the bivariate chromatic polynomial. Bruno Courcelle and Lorenzo Traldi sent me valuable comments during different stages of my research. Holger Dell provided me with a preliminary version of his work on the exponential time complexity of the Tutte polynomial. Sascha Parduhn listened to many of my ideas and discussed them with me. I would like to thank all of them for their support. To the same extent, I would like to thank everyone who has been supporting me in non-scientific matters during the last five years, most of all, my family, friends, and colleagues.

Finally, I would like to express my gratitude for what has been and will always be the foundation of my life: I would like to thank God for making peace with me by his son Jesus Christ and for his presence in very dark just as in very joyful hours of my life.

Contents

1 Introduction

Graph polynomials are functions that map graphs to polynomials in such a way that isomorphic graphs are mapped to the same[1] polynomial. Many graph polynomials encode interesting properties of the graph. This information can be the coefficients of the actual polynomial or the values of the polynomial at a particular point. In this way, every evaluation point gives rise to a computational problem: given a graph, evaluate the graph polynomial at this point. Such evaluation problems often correspond to computational problems on graphs, for instance counting independent sets or colorings. As in these examples, these problems are often well-known hard problems. An ultimate goal in the study of graph polynomials is to determine for every point of the domain the complexity of evaluating the graph polynomial at this point.

As all evaluation points and the associated computational problems belong to one and the same graph polynomial, the polynomial can provide a means to find reductions between these problems. Thus, another goal in the study of graph polynomials is to find such reductions. A fruitful approach are graph transformations that yield point-to-point reductions. This means the following: We have an efficiently computable procedure to transform every graph G into a transformed graph G'. Additionally, we know that, for some evaluation points x, x', the original and transformed graph fulfill an equation such as $P(G'; x) = P(G; x')$. This enables us to reduce the evaluation at x' to the evaluation at x: Transform the input G into G' and evaluate $P(G'; x)$.

The first part of this thesis (Chapter 2–5) is devoted to finding such reductions and proving several hardness results for evaluation of graph polynomials. The main contributions are to the interlace polynomial and the extended bivariate chromatic polynomial. We show that both polynomials are #P-hard to evaluate almost everywhere. Additionally, we give hardness results for the interlace polynomial and the Tutte polynomial in Valiant's algebraic model of computation. Furthermore, we obtain an inapproximability result for the independent set polynomial. Finally, we show that, unless the counting version of the exponential time hypothesis fails, evaluating the independent set polynomial of an n-vertex graph requires time $\exp(\Omega(n/\log^3 n))$ at almost every point. This is achieved by a reduction for the interlace polynomial that preserves exponential time hardness up to an $\Theta(\log^3 n)$ factor in the exponent.

Concerning point-to-point reductions for graph polynomials, we also develop two

[1] In the sense that polynomials that involve vertex/edge variables are considered to be the same if they can be obtained from each other by renaming the vertex/edge variable as induced by the graph isomorphism, see Definition 1.3.

general tools. First, we define multivariate vertex/edge induced subgraph polynomials (VSPs/ESPs). We show that if the generating function of such a graph polynomial fulfills a simple property ("supports cloning"), this leads to a powerful family of point-to-point reductions, which finally enables interpolation of the whole polynomial. This technique can be applied to the independent set, interlace, Tutte, and extended bivariate chromatic polynomial.

As a second tool, we consider a class of graph transformations which we call local transformations. We exhibit an important property that these transformation can fulfill[2]. This property is crucial in many proofs of point-to-point reductions, including well-known ones as well as new ones devised in this work. We call local transformations that fulfill this property "uniform", and we prove that if a local graph transformation is uniform, it immediately yields a point-to-point reduction. This method works for a class of multivariate graph polynomials that can be defined as weighted sums over vertex/edge based substructures of a graph, which we call vertex/edge function polynomials (VFPs/EFPs). This concept is more general than the notion of vertex/edge induced subgraph polynomials. Using this approach, we give a new point-to-point reduction for a vertex cover polynomial and a domination polynomial.

The second part of the thesis (Chapter 6 and 7) is concerned with algorithms to compute graph polynomials. We start by observing that a recent method developed for the Tutte polynomial can be applied to the extended bivariate chromatic polynomial as well: Although being characterized by a sum over 3-partitions of the *edges*, the polynomial can be computed in *vertex*-exponential time.

Our main algorithmic contribution is with respect to the interlace polynomial. We develop a technique that enables us to evaluate the interlace polynomial using tree decompositions. Our main result is a parameterized algorithm to evaluate the interlace polynomial in time linear in the size of the graph and single exponential in the treewidth. We discuss several variants of the algorithm, including efficient parallelization. Our techniques also allow us to evaluate the interlace polynomial of *any* graph asymptotically faster than by the trivial exponential time algorithm. We conclude with two different algorithms that are substantially faster if we restrict ourselves to simple graphs of bounded maximum degree or to low-degree coefficients of the polynomial.

In the rest of this chapter, we first introduce notation (Section 1.1). Then we give a short introduction to the graph polynomials that we will consider in this work (Section 1.2). After that, we explain what computational models we use (Section 1.3) and define computational problems on graph polynomials formally (Section 1.4). Finally, in Section 1.5, we give a detailed discussion of our results and relate them to previous work.

[2] with respect to a particular vertex/edge function

1.1 Terminology and Basic Definitions

We write $\mathbb{N}$ for the set of natural numbers, $\mathbb{N} = \{0, 1, 2, \ldots\}$, $\mathbb{Z}$ for the set of integers, $\mathbb{Q}$ for the rationals, $\mathbb{R}$ for the reals, and $\mathbb{C}$ for the complex numbers. As $\sqrt{2}$ will play a special role in the computational complexity of the interlace polynomial, we define $\widetilde{\mathbb{Q}} = \mathbb{Q}(\sqrt{2})$, i.e. the smallest field extension of $\mathbb{Q}$ that contains $\sqrt{2}$.

For a positive integer q, we let $[q] = \{1, 2, \ldots, q\}$. We write $q^{\underline{k}}$ for the *falling factorial*, which is defined as $q^{\underline{k}} = q(q-1)(q-2)\ldots(q-k+1)$.

The power set of a set A is denoted by 2^A. We write $A \triangle B$ to denote the symmetric difference of two sets, $A \triangle B = (A \setminus B) \cup (B \setminus A)$. If a set A is naturally contained in a ground set S, we write $\bar{A}$ to denote $S \setminus A$. A typical example is $A \subseteq V$, where A is a subset of V, the vertices of a graph.

If $a = a_1 \ldots a_\ell$ is a string of length ℓ, $a^R = a_\ell a_{\ell-1} \ldots a_1$ is the *reverse* of s.

We will consider different types of graphs in this work. The term "graph" always refers to undirected graphs; we use the expression "digraph" for directed graphs. Depending on the context, a graph may or may not have multiple edges and self loops. We use the term "simple graph" to denote an undirected graph without multiple edges and without self loops. The expression "simple graph with self loops" denotes an undirected graph without multiple edges that may have self loops. If we use just the term "graph", it is supposed to denote a graph that may have multiple edges and self loops. For a formal definition of graphs, we refer to Diestel [Die05, Section 1.1 and 1.10]. We assume that, for every graph, the set of its vertices and the set of its edges are disjoint.

If a and b are vertices of an undirected graph without multiple edges, we write $\{a, b\}$ or ab to denote the edge between a and b. If G is a graph, we denote its vertices by $V(G)$ and its edges by $E(G)$; if D is a digraph, we write $E(D)$ for the set of its directed edges. Sometimes we use the term "arc" for a directed edge. The *size of a graph* G, $|G|$, is the number of its vertices.

Let $G = (V, E)$ be a graph. If $U \subseteq V$ is a subset of the vertices, $G[U]$ denotes the subgraph of G that is induced by vertex subset U, i.e.

$$G[U] = (U, \{e \in E \mid \text{both end vertices of } e \text{ are in } U\}.$$

If $A \subseteq E$ is a subset of the edges of graph $G = (V, E)$, $G[A]$ denotes (V, A), the subgraph of G that is induced by edge subset A. Furthermore, we write $V(A)$ to denote the set of vertices of G that are end vertices of an edge in A.

$\mathcal{C}G$ denotes the connected components of a graph G, $k(G)$ the number of connected components of G. If $G = (V, E)$ is a graph and $A \subseteq E$ is a subset of the edges, the *covered components* of A are the connected components of $(V(A), A)$. We denote the number of covered components of an edge set A by $k_{\text{cov}}(A)$.

If $G = (V, E)$ is a simple graph with self loops and $U \subseteq V$ is a subset of the vertices, $G\nabla U$ is obtained from G by "toggling" the self loops of the vertices in U, i.e.

$$G\nabla U = (V, E \triangle \{uu \mid u \in U\}).$$

For a graph G and a vertex a in G, $N(a) := N_G(a)$ denotes the neighbors (neighborhood) of a, i.e. all vertices b of G, $b \neq a$, such that a and b are connected by an edge in G. Note that a is never a neighbor of itself, even if it has a self loop. However, if a has a self loop, we say that a is adjacent to itself.

For a matrix $A = (a_{ij})$ over $\{0, 1\}$, A^C, the complement of A, is defined as $(a_{ij} \oplus 1)$, where $\oplus$ is the XOR operation (addition in the field with two elements).

The rank of a matrix A will be denoted by $\operatorname{rk} A$. The nullity of a matrix A of dimension $n \times n$ is $n - \operatorname{rk} A$. We will abbreviate it by $\operatorname{n} A$.

The rank of the adjacency matrix of a graph will always be the rank over $GF(2)$, the field with two elements. Unless explicitly stated otherwise, the rank (nullity) of a graph is the rank (nullity, resp.) of its adjacency matrix. We denote it by $\operatorname{rk} G$ ($\operatorname{n} G$, resp.).

Definition 1.1 (Graph Isomorphism)**.** *A* graph isomorphism *from a graph $G_1 = (V_1, E_1)$ onto a graph $G_2 = (V_2, E_2)$ is a bijection $\varphi : V_1 \cup E_1 \to V_2 \cup E_2$ such that*

1. *φ maps vertices to vertices and edges to edges and*

2. *for all $v \in V_1$, $e \in E_1$, v is an end vertex of edge e in G_1 iff $\varphi(v)$ is an end vertex of edge $\varphi(e)$ in G_2.*

A *vertex-indexed* variable $\mathbf{x}$ is a set of variables the elements of which are indexed by vertices. For a graph $G = (V, E)$, a G-vertex indexed (or, shorter, V-indexed) variable $\mathbf{x}$ includes the set $\{x_a \mid a \in V\}$ of independent variables. *Edge-indexed* variables are defined analogously. Vertex-indexed and edge-indexed variables of a particular graph G are also called *G-indexed* variables; vertex- and edge-indexed variables in general (i.e. without restriction to one particular graph) are called *$\mathcal{G}$-indexed* variables. If $\mathbf{x}$ is a G-vertex (G-edge) indexed variable and A is a subset of the vertices (edges, resp.) of G, we define

$$x_A := \prod_{a \in A} x_a.$$

If $P(G; \mathbf{x})$ is a polynomial with G-indexed $\mathbf{x}$, we write $P(G; x)$ to denote $P(G; \mathbf{x})$, where $x_a := x$ for every $x_a \in \mathbf{x}$.

Variables that are not indexed by a vertex or edge are called *ordinary variables.*

Definition 1.2. *Let φ a graph isomorphism from G_1 onto G_2, R a ring, $\mathcal{X}$ a set of G_1-indexed variables, and $P \in R[\mathcal{X}]$ a polynomial. Then φ induces a polynomial*

$$P_\varphi$$

in G_2-indexed variables, which is obtained from P in the following way: Substitute x_a by $x_{\varphi(a)}$ for every G_1-indexed variable $\mathbf{x}$ and every vertex (edge, resp.) a of G_1.

1.2 Graph Polynomials

Definition 1.3. *Let R be a ring, $\mathcal{X}$ a set of $\mathcal{G}$-indexed variables and $\mathcal{Y}$ a set of ordinary variables. A* graph polynomial *p maps graphs into $R[\mathcal{X} \cup \mathcal{Y}]$ such that*

1. *$p(G)$ is a polynomial in G-indexed and ordinary variables for every graph G and*
2. *$p(G_1)_\varphi = p(G_2)$ whenever φ is a graph isomorphism from G_1 onto G_2.*

Remark 1.4. *If a graph polynomial maps into a polynomial ring over ordinary variables only, isomorphic graphs are mapped to the same polynomial.*

Some of the graph polynomials we are considering in this work, such as the interlace polynomial, will be defined on simple graphs with or without self loops. Others are defined on graphs with multiple edges, such as the Tutte polynomial and the extended bivariate chromatic polynomial. The set of graphs on which a particular graph polynomial is defined will be denoted by $\mathcal{G}$.

While a graph polynomial p formally is not just a polynomial but a mapping from graphs to polynomials, often $p(G)$ is called graph polynomial, too: If it is clear from the context that we are considering a particular graph polynomial p, we may use the expression "the graph polynomial of G" for $p(G)$. For instance, we will talk about "the interlace polynomial of the empty graph" or "the chromatic polynomial of the edgeless graph on n vertices".

The coefficients of a graph polynomial $p(G)$ typically count some properties of the graph G. Consequently, most natural arising graph polynomials are polynomials over the integers. But our considerations are not limited to this coefficient ring. In fact, applications from physics suggest to evaluate graph polynomials at non-integer points. For this reason, we consider most graph polynomials as polynomials over the rationals, the reals, or the complex numbers. When computational aspects come into play, we restrict ourselves to a finite dimensional extension of $\mathbb{Q}$.[3]

Many graph polynomials are polynomials in one, two, or are another fixed number of variables. Even though these polynomials may have more than one variable, we use the term *multivariate graph polynomial* only for graph polynomials that have $\mathcal{G}$-indexed variables.

1.2.1 Some Examples

The Independent Set Polynomial

As a first example, let us consider the independent set polynomial [HL94], which sometimes is called independent polynomial [GH83]. We define the independent set polynomial on simple graphs. A common definition of the independent set polynomial is

$$I(G;x) = \sum_{0 \le k \le n} i(G;k)x^k, \tag{1.1}$$

[3] Alternatively, we could use an algebraic model of computation.

where $G = (V, E)$ is a graph and $i(G; k)$ is the number of independent sets of G of size k. (An independent set of a graph $G = (V, E)$ is a vertex set $A \subseteq V$ such that no two different vertices of A are connected by an edge.)

$I(G, 1)$ counts independent sets, i.e. $I(G, 1)$ equals the number of independent sets of the graph G. If we evaluate I at other points than 1, independent sets of different size are weighted differently. Besides being an interesting mathematical object, the independent set polynomial is also important in physics as it is the partition function of the lattice gas with hard-core self-repulsion and hard-core pair interaction [SS05].

To obtain a multivariate independent set polynomial, we can write the independent set polynomial as

$$I(G; x) = \sum_{\substack{A \subseteq V \\ A \text{ independent}}} x^{|A|}. \tag{1.2}$$

This inspires the following multivariate version of the independent set polynomial [SS05]:

$$I(G; \mathbf{x}) = \sum_{\substack{A \subseteq V \\ A \text{ independent}}} x_A, \quad \text{where} \quad x_A = \prod_{a \in A} x_a. \tag{1.3}$$

Here we have a different variable x_v for every vertex v of G, i.e. $\mathbf{x}$ is a vertex-indexed variable.

The Chromatic Polynomial

The chromatic polynomial [Bir12] is another important graph polynomial. Sokal lists more than 50 papers that study the chromatic polynomial [Sok04].

For a simple graph $G = (V, E)$ and a natural number λ, a λ-coloring is a mapping from the vertices of G to a set of λ elements, which are called colors, such that the two end vertices of every edge receive different colors. The number of λ-colorings of G is denoted by $\chi(G; \lambda)$. It is easy to see that $\chi(G; \lambda)$ is a polynomial in λ, the chromatic polynomial. For the proof we need the concept of edge deletion and edge contraction.

Definition 1.5 (Edge Deletion and Edge Contraction)**.** *Let $G = (V, E)$ be a graph and e be an edge that connects the vertices u and v.*

1. *$G \setminus e$ is the graph G with edge e deleted. This means $G \setminus e = (V, E \setminus \{e\})$.*
2. *G/e is the graph G with edge e contracted, i.e. edge e is removed and its two end vertices u, v are unified. (If G has, besides e, other edges between u and v, these become self loops in G/e.)*

Proposition 1.6 (Deletion-Contraction Recursion)**.** *$\chi(G; \lambda)$ fulfills the following recursion:*

$$\chi(G; \lambda) = \begin{cases} \lambda^n & \text{if } G \text{ is the edgeless graph with } n \text{ vertices} \\ \chi(G \setminus e; \lambda) - \chi(G/e; \lambda) & \text{if } G \text{ has an edge } e \end{cases}$$

Thus, $\chi(G;\lambda)$ is a polynomial in λ, the chromatic polynomial.

Proof. The statement for the edgeless graph is obvious from the definition.

Let e be an edge of G connecting vertices a and b. We argue that the λ-colorings of $G \setminus e$ are the "disjoint union" of the λ-colorings of G and of G/e: The λ-colorings of $G \setminus e$ where a and b receive different colors are exactly the λ-colorings of G. The λ-colorings of $G \setminus e$ where a and b receive the same color correspond to the λ-colorings of G/e. □

The Tutte Polynomial

The Tutte polynomial is a generalization of the chromatic polynomial. It is universal in the sense that every graph polynomial that fulfills a deletion-contraction recursion can be derived from the Tutte polynomial [OW79], [Bol98, Theorem X.2]. The Tutte polynomial contains other important graph invariants such as the flow polynomial, the reliability polynomial, the Jones polynomial, and Potts model [Bol98, Chapter X], [Sok05]. It counts spanning trees, forests, connected spanning subgraphs, and spanning subgraphs. We will use the multivariate Tutte polynomial [Sok05]. For a graph $G = (V, E)$, which may have self loops and multiple edges, the multivariate Tutte polynomial is[4]

$$Z(G; q, \mathbf{x}) = \sum_{A \subseteq E} x_A q^{k(G[A])}. \tag{1.4}$$

The Tutte polynomial fulfills the following recursion [Sok05, (4.16)]

$$Z(G; q, \mathbf{x}) = \begin{cases} q^n & \text{if } G \text{ is the edgeless graph} \\ & \text{with } n \text{ vertices} \\ Z(G \setminus e; q, \mathbf{x}) + v_e Z(G/e; q, \mathbf{x}) & \text{if } G \text{ has an edge } e \end{cases} \tag{1.5}$$

Proposition 1.6 and (1.5) show that the relation between the Tutte polynomial and the chromatic polynomial is

$$\chi(G; \lambda) = Z(G; \lambda, -1). \tag{1.6}$$

Classically (see, for example, Tutte [Tut84], Brylawski and Oxley [BO92], Welsh [Wel93], or Bollobás [Bol98]), the Tutte polynomial is defined as

$$T(G; x, y) = \sum_{A \subseteq E} (x-1)^{\operatorname{rk} G[E] - \operatorname{rk} G[A]} (y-1)^{|A| - \operatorname{rk} G[A]}, \tag{1.7}$$

where the rank of a graph with n vertices and k connected components is defined as $n-k$. In fact, this is the $GF(2)$-rank of the incidence matrix of the graph [ABS04b]. $T(G; x, y)$ can easily be converted into $Z(G; q, x)$ and vice versa.

[4] We use $\mathbf{x}$ as edge-indexed variable as $\mathbf{v}$, what Sokal uses, might mistakenly be understood as referring to vertices.

The Bivariate Chromatic Polynomial

Another generalization of the chromatic polynomial has been introduced by Dohmen, Pönitz, and Tittmann [DPT03]. They define $P(G; x, y)$ to be the number of generalized proper (x, y)-colorings of a simple graph $G = (V, E)$ with x colors, y of which are proper. A generalized (x, y)-coloring is a mapping from the vertices to a set X, the colors, $|X| = x$. These colors are split into Y, the set of proper colors, $|Y| = y$, and $X \setminus Y$, the improper colors. A generalized coloring is proper if no two adjacent vertices receive the same *proper* color. Dohmen et al. prove that $P(G; x, y)$ is a polynomial in x and y. In fact, they show

$$P(G; x, y) = \sum_{A \subseteq V} (x - y)^{|A|} \chi(G[V \setminus A]; y). \tag{1.8}$$

They also observe that the univariate independent set polynomial can be obtained from P as follows [DPT03, Corollary 2][5]:

$$I(G; x) = x^{|V(G)|} P(G; \frac{1}{x} + 1, 1). \tag{1.9}$$

The bivariate chromatic polynomial fulfills a recursion, see (1.12) and (1.15).

Matching Polynomials

Several versions of matching polynomials have been considered. The matching generating polynomial is very similar to the independent set polynomial: Let $m_G(k)$ be the number of matchings of size k of the simple graph G. (A matching of a graph $G = (V, E)$ is an edge subset $A \subseteq E$ such that no two edges in A have a vertex in common.) The matching generating polynomial is

$$g(G; x) = \sum_{0 \leq k} m_G(k) x^k = \sum_{\substack{A \subseteq E \\ A \text{ matching}}} x^{|A|}.$$

Another version is the matching defect polynomial

$$\mu(G; \lambda) = \sum_{0 \leq k} m_G(k) (-1)^k \lambda^{|V| - 2k}.$$

The term in the exponent, $|V| - 2k$, equals the number of vertices in $G = (V, E)$ that are not matched if the matching has size k. The following bivariate matching polynomial generalizes g and μ.

$$M(G; x, y) = \sum_{0 \leq k} m_G(k) x^{|V| - 2k} y^k.$$

[5] Dohmen et al. do not use use $I(G; x)$, but $\sum_k i_G(k) x^{n-k}$, a slightly different version of the independent set polynomial. Thus, (1.9) differs a bit from their Corollary 2.

Multivariate matching polynomials have been defined, too. Averbouch and Makowsky suggest the following version [AM07]. It has vertex-indexed variables $\mathbf{x}$ and edge-indexed variables $\mathbf{y}$.

$$U(G;\mathbf{x},\mathbf{y}) = \sum_{\substack{A\subseteq E\\ A \text{ matching}}} x_{V\setminus V(A)} y_A.$$

The matching generating polynomial g, the matching defect polynomial μ, the bivariate matching polynomial M, the rook polynomial and the multivariate matching polynomial of Heilmann and Lieb can be obtained from U by substitution and multiplication with a prefactor [AM07, Theorem 1].

Recursions for matching polynomials use the following operation.

Definition 1.7 (Edge Extraction)**.** *Let $G = (V, E)$ be a graph and e be an edge that connects the vertices u and v. Then $G \dagger e$ denotes the graph G with edge e* extracted. *This means $G \dagger e = G[V \setminus \{u, v\}]$.*

U fulfills the following recursion [AM07]:

$$U(G;\mathbf{x},\mathbf{y}) = \begin{cases} x_V & \text{if } G \text{ is the edgeless graph} \\ U(G\setminus e;\mathbf{x},\mathbf{y}) + y_e U(G\dagger e;\mathbf{x},\mathbf{y}) & \text{if } G \text{ has an edge } e. \end{cases}$$

In Section 3.1, we will come back the the following special case, from which we can obtain U by substitution and multiplication with an appropriate prefactor:

$$M(G;\mathbf{x}) := U(G;1,\mathbf{x}) = \sum_{\substack{A\subseteq E\\ A \text{ matching}}} x_A. \tag{1.10}$$

The Extended Bivariate Chromatic Polynomial

All polynomials[6] that have been mentioned so far are subsumed by the extended bivariate chromatic polynomial by Averbouch, Godlin, and Makowsky [AGM10]. The extended bivariate chromatic polynomial of a graph G, which may have multiple edges and self loops, equals

$$\xi(G;x,y,z) = \sum_{\substack{A,B\subseteq E\\ V(A)\cap V(B)=\emptyset}} x^{k(G[A\cup B])-k_{\text{cov}}(B)} y^{|A|+|B|-k_{\text{cov}}(B)} z^{k_{\text{cov}}(B)}. \tag{1.11}$$

ξ fulfills the following recursion: $\xi(G;x,y,z) = x^{|V|}$ if G has no edges and

$$\xi(G;x,y,z) = \xi(G\setminus e;x,y,z) + y\xi(G/e;x,y,z) + z\xi(G\dagger e;x,y,z) \tag{1.12}$$

if G has an edge e. Other graph polynomials can be derived from ξ as follows [AGM10]:

$$Z(G;q,x) = \xi(G;q,x,0), \tag{1.13}$$
$$M(G;x,y) = \xi(G;x,0,y), \tag{1.14}$$
$$P(G;x,y) = \xi(G;x,-1,x-y). \tag{1.15}$$

[6] At least in their non-multivariate versions.

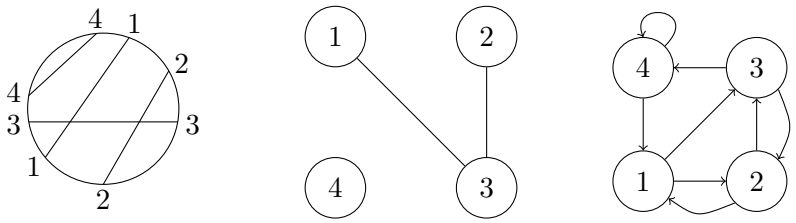

Figure 1.1: A chord diagram with corresponding interlace graph and 2-in, 2-out digraph.

1.2.2 Interlace Polynomials

The interlace polynomial is a graph polynomial that, if evaluated on interlace graphs, counts Euler circuits. It is a main object of study of this work. We give an introduction to the interlace polynomial in this section. The material of this section is taken from the literature, except for the side remark on Page 23.

Remark. After submission of this thesis, the author became aware of a new interesting discussion of interlace polynomials. In particular, an alternative multivariate interlace polynomial is suggested [Tra10a].

Counting Euler Circuits and Edge Partitions via the Interlace Relation

The graph polynomials that we have discussed so far "count" objects that have a straightforward connection to the actual graph. For instance, the independent set polynomial of a graph G counts the independent sets of G, the chromatic polynomial of G counts colorings of G. But the interlace polynomial of a graph G counts objects that are only indirectly connected to G. These objects have been described in three different ways.

The first way uses collections of chords: Consider a circle with several chords as in Figure 1.1, left part. Some of the chords intersect, while others do not intersect. We can represent the intersection relation by an interlace graph (also known as circle graph [Gol91, GSH89, Spi94]), see middle part of Figure 1.1. The vertices of the graphs are the chords. Two vertices are connected iff the corresponding chords intersect. The interlace polynomial of the interlace graph contains information that is related to the collection of chords. (We will make this precise soon.)

Instead of collection of chords, we can also consider a sequences of characters (i.e. a string s) where each character appears exactly twice. (For instance, the chord collection of Figure 1.1 corresponds to the string 12321344, which is obtained by walking clockwise along the circumference of the circle.) This is inspired from computational biology, in particular DNA sequencing by hybridization [ABCS00]. If we consider two different characters a and b of s, the order in which a and b appear may be either

- $\ldots a \ldots a \ldots b \ldots b \ldots$ (or any rotation of it), or

- $\ldots a \ldots b \ldots a \ldots b \ldots$ (or a rotation of it).

In the last case we say that a and b are *interlaced*. This is equivalent to chords a and b intersecting. In 12321344, for instance, 1 and 2 are not interlaced but 1 and 3 are. Each sequence induces an interlace graph that represents the interlace relation between the characters of the sequence.

Finally, every collection of chords (or, equivalently, sequence of characters where each character occurs exactly twice) corresponds to an Euler circuit in a 2-in, 2-out digraph D (i.e. a digraph in which every vertex as indegree exactly 2 and outdegree exactly 2) as follows. The vertices of D are the chords (characters, resp.). The arcs of D are obtained by walking along the circumference of the circle of the chords. Whenever we move from one endpoint of chord a to an endpoint of chord b, we insert a directed edge from a to b in D. The digraph corresponding to the chord collection in the left of Figure 1.1 is drawn in the right part of this figure.

Definition 1.8 (Interlace graph / circle graph)**.** *Let C be an Euler circuit in a 2-in, 2-out digraph (or, equivalently, a collection of chords of circle, or a sequence of characters where each character appears exactly twice). The* interlace graph *(or* circle graph*) of C is the graph that has as vertices the vertices of C and an edge between two vertices a, b iff a and b are interlaced in C.*

Now we can give a precise statement of what the interlace polynomial counts. The vertex-nullity interlace polynomial is defined on undirected graphs H without multiple edges but with self loops allows. It is a polynomial in one variable, written $q_N(H; y)$. The definition is given in Definition 1.13. Assume that we are given an Euler circuit C in a 2-in, 2-out digraph D. Let H be the interlace graph of C. Then we have [ABS04a, Theorem 9, Theorem 12]

$$q_N(H; 1) = \#\text{Euler circuits in } D. \tag{1.16}$$

Note that $q_N(H; 1)$ evaluates the polynomial of the interlace graph H, not of D. Thus, it is enough to know the interlace relation of the vertices with respect to some Euler circuit in D to deduce the number of Euler circuits of D. Any other information such as information about the actual arcs of D is not needed. This also holds for the following, more general statement, which obviously includes (1.16). If D is a 2-in, 2-out digraph, C is an Euler circuit in D, and H is the interlace graph of C, then

$$q_N(H; y) = \sum_P (y-1)^{|P|-1}, \tag{1.17}$$

where the sum is over all partitions P of $E(D)$ into directed circuits. This follows from a theorem in the work of Arratia et al. [ABS04a, Theorem 24]. Note that Traldi gives an enlightening proof of this fact [Tra09].

The interlace polynomial counts *undirected* circuit partitions, too. Following the exposition of Traldi [Tra09], we describe this in the rest of this subsection. We need some definitions. Let D be a 2-in, 2-out digraph. Let G be the undirected

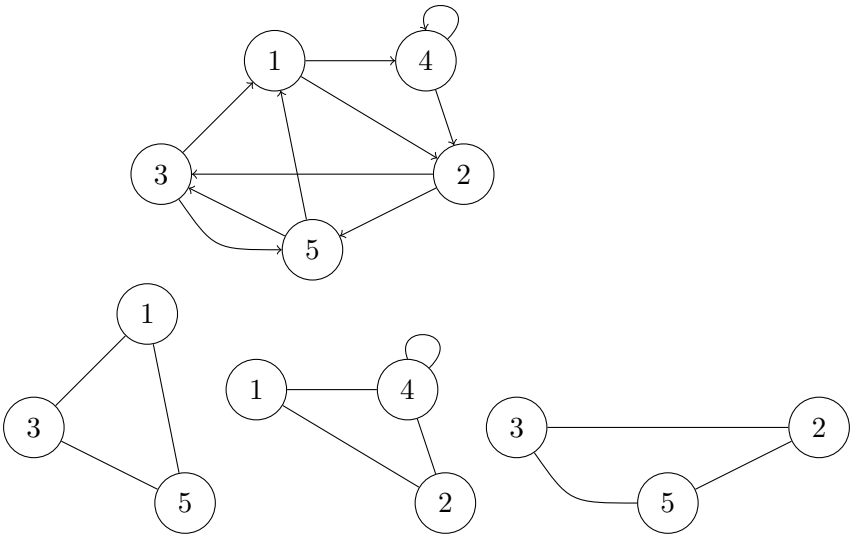

Figure 1.2: A 2-in, 2-out digraph and a partition of the undirected edges.

graph that we obtain when we remove the orientation of the edges of D. Let P be a partition of the edges of G into undirected circuits and let C be a (directed) Euler circuit in D. For example, see Figure 1.2, where D is drawn in the upper part and P below. Let v be a vertex of D (or, equivalently, G). When following C, v is visited twice. Let u_1vw_1 and u_2vw_2 denote these two visits. There are three possibilities how P can be related to C in v:

- P follows C through v. This means that $\{u_1, v\}$ and $\{v, w_1\}$ are neighbor edges on some circuit of P. It follows that also $\{u_2, v\}$ and $\{v, w_2\}$ are neighbor edges on some (possibly different) circuit of P.
- P is orientation-consistent at v but does not follow C. This means that $\{u_1, v\}$ and $\{v, w_2\}$ are neighbor edges on some circuit of P. It follows that also $\{u_2, v\}$ and $\{v, w_1\}$ are neighbor edges on some circuit of P.
- P is orientation-inconsistent at v. This means that $\{u_1, v\}$ and $\{v, u_2\}$ are neighbor edges on some circuit of P. If follows that also $\{w_1, v\}$ and $\{v, w_2\}$ are neighbor edges on some circuit of P.

For an example, let us consider the digraph D and the undirected edge partition P in Figure 1.2. With respect to Euler circuit $C = 1231442535$, partition P follows[7] C through vertex 4, is orientation consistent but does not follow C in vertex 3, and is orientation inconsistent in vertices 1, 2 and 5.

[7] In fact, a drawing as in Figure 1.2 does not show the difference between P following C through 4 and being orientation-inconsistent at 4. To handle self loops in a well-defined way, the definition of P following C in v etc. should be given in terms of half-edges [Tra09].

Theorem 1.9 ([Tra09, Corollary 5]). *Let D be a 2-in, 2-out digraph and G be obtained from D by removing the edge orientations. Let C be a collection of circuits in D that induces an Euler circuit in each component of G. Let H be the interlace graph of C with self loops added at some of the vertices. Then*

$$q_N(H;y) = \sum_{A \subseteq V(H)} (y-1)^{|P_A|-k(G)},$$

where P_A is the undirected circuit partition that follows C at each vertex $v \notin A$, is orientation-inconsistent at each looped vertex $v \in A$, and is orientation-consistent but does not follow C at each unlooped vertex $v \in A$.

Edge Pivot, Local Complement, and Recursive Definitions

Assume that we have an Euler circuit C in a 2-in-2-out digraph with a and b interlaced. A natural operation on C is to swap the two paths from a to b:

Definition 1.10 ([ABS04a, Definition 2]). *Given an Euler circuit C with a and b interlaced, a transposition on the pair ab is the circuit C^{ab} resulting from exchanging one of the edge sequences from a to b with the other.*

A transposition of an interlaced pair on Euler circuits corresponds to a graph operation on the *interlace* graph, the edge pivot operation.

Definition 1.11 (Edge Pivot, [ABS04a, Definition 5]). *Let ab be an edge in a graph G and let V_a be the vertices in G which are neighbors of a but not of b, V_b the vertices in G which are neighbors of b but not of a, and V_{ab} the vertices in G which are neighbors of a and of b (cf. Figure 1.3). Then G^{ab} is the graph G with all edges and non-edges toggled between all vertices u and v where u and v belong to different sets from $\{V_a, V_b, V_{ab}\}$. Here, toggling means that an edge in G becomes a non-edge in G^{ab}, a non-edge in G becomes an edge in G^{ab}.*

Edge pivot on interlace graphs and transpositions on Euler circuits are connected in the following way [ABS04a, Lemma 7]: If H is the interlace graph of C and $H(C^{ab})$ is the interlace graph of C^{ab}, we have

$$H^{ab} = H(C^{ab})_{ab}.$$

Here, G_{ab} denotes the graph obtained from G swapping the labels of vertices a and b.[8]

As a side remark, let us consider the following question: Let C_1, C_2 be two Euler circuits in the same 2-in, 2-out digraph D and their interlace graphs be $H(C_1)$ and $H(C_2)$. From (1.17) we know that $q_N(H(C_1);y) = q_N(H(C_2);y)$. Could this

[8] It would also be possible to define the edge pivot operation such that it includes the swapping of the labels of the end vertices of the edge which is used for pivot. This would simplify the notation a bit. However, we prefer to follow the notation that has been introduced in the literature.

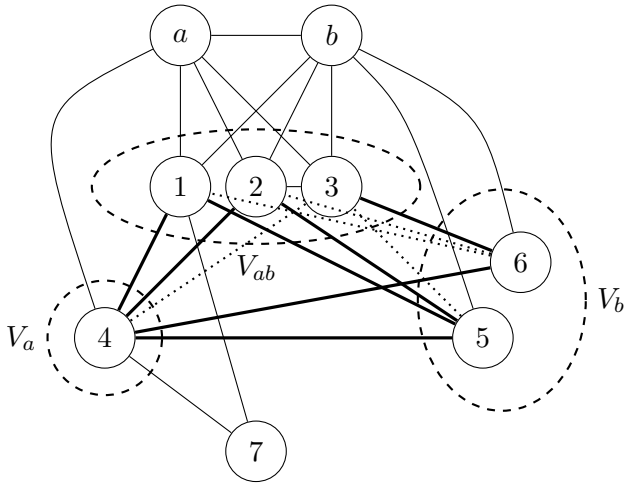

Figure 1.3: Edge pivot on edge ab. In G, thick edges are present, dotted edges are absent. In G^{ab}, thick edges are absent, dotted edges are present.

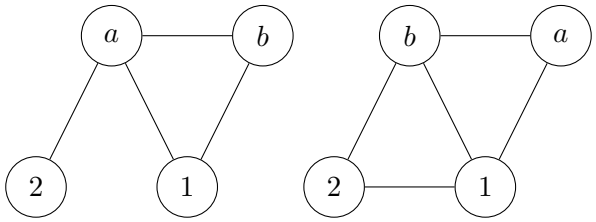

Figure 1.4: G (left graph) and $(G^{ab})_{ab}$ (right graph) are not isomorphic, but interlace graphs of $C = ab12ab21$ and C^{ab}, which are Euler circuits in one and the same digraph.

be just because interlace graphs of Euler circuits in the same digraph are always isomorphic? The answer is negative, and the connection between transposition and edge pivot yields an example: Choose $H(C_1)$ "complicated enough" such that the edge pivot with ab adds an edge. Thus, the pivoted graph $H(C_1)^{ab}$ is not isomorphic to $H(C_1)$. Nevertheless, with $C_2 = {C_1}^{ab}$ we have C_1 and C_2 both in the same 2-in-2-out digraph. See Figure 1.4.

Let us come back to the topic of this subsection: recursive definitions of the interlace polynomial. Arratia et al. observe [ABS04a, Theorem 12] that, for loopless graphs G,

$$q_N(G;y) = \begin{cases} q_N(G-a;y) + q_N(G^{ab}-b;y) & \text{if } G \text{ contains an edge } ab \\ y^n & \text{if } G \text{ is the edgless graph} \\ & \text{with } n \text{ vertices.} \end{cases} \tag{1.18}$$

In fact, (1.18) can also be used as a definition for the vertex-nullity interlace polynomial as it defines a unique polynomial [ABS04a, Theorem 12].

Local complementation is another graph operation that is important for the interlace polynomial.

Definition 1.12. *The local complement of a graph G at vertex a, G^a, is the graph G where the subgraph induced by the neighborhood of a is replaced by its complement, i.e. edges (including self loops) are replaced by non-edges and vice versa.*

We have [ABS04b, Theorem 6]

$$q_N(G) = q_N(G - a) + q_N(G^a - a) \quad \text{if } G \text{ has a self loop at vertex } a. \tag{1.19}$$

Local complementation on interlace graphs corresponds to the following operation on Euler circuits in 2-in, 2-out digraphs. Let a be a vertex and $s = s_1as_2as_3$ be a sequence of vertices where every vertex appears exactly twice. The two occurrences of a are separated by the substrings s_1, s_2, and s_3. Then we define s^a by $s_1as_2{}^Ras_3$. Note that D', the digraph of s^a is, in general, different from D, the digraph of s: we obtain D' from D by reversing the arcs corresponding to the circuit as_2a. (As this is a circuit, the reversion of the arcs retains the 2-in, 2-out property.) Even though the underlying digraph may change, for the interlace graph H of an Euler circuit C containing a vertex a and the interlace graph $H(C^a)$ of the Euler circuit C^a we have

$$H(C^a) = (H^a)\nabla N(a).$$

This can be verified by a simple case distinction and has already been observed by Bouchet [Bou94]. Note that $(H^a)\nabla N(a)$ is just local complementation "modulo self loops". This means that we do not add/remove self loops during local complementation. The reason why local complementation is defined including self loops will become clear in the next subsection.

Local complementation and edge pivot establish a connection between the interlace polynomial and codes: Danielsen and Parker mention that "the local complement orbit of a graph corresponds to the equivalence class of a self-dual quantum code" and a similar connection between edge pivot and binary linear codes [DP08]. But note also that Bouchet observed that "the theory of isotropic systems is the theory of simple graphs up to local complementation" [Bou88]. Isotropic systems are a notion developed by Bouchet, that unifies "some properties common to 4-regular graphs and pairs of dual binary matroids". Some properties of the interlace polynomial have been proved using isotropic systems [Bou05].

Linear Algebra Characterizations

The number of circuit partitions as well as the graph operations can be expressed by means of Linear Algebra.

Traldi notes the following correspondence between partitions into circuits, a special kind of permutations, and vertex subsets. Let D be a 2-in, 2-out digraph and

C an Euler circuit in D. Consider a partition P of the edges of D into directed circuits. P corresponds to a permutation π_P of a special form that can be described by a subset A of vertices of D. A is the set of vertices in which P does not follow C [Tra09, Kot68]. Traldi notes further that the cardinality of P equals the number of orbits of π_P on the edge set. Using the Cohn-Lempel equality [CL72], [Tra09, Theorem 1], this number can be expressed as the $GF(2)$-nullity of the A-induced submatrix of the adjacency matrix of the interlace graph of C. Thus, (1.17) can be written as [Tra09, Corollary 2]

$$q_N(H;y) = \sum_P (y-1)^{|P|-1} = \sum_{A \subseteq V} (y-1)^{\mathrm{n}(H[A])}, \tag{1.20}$$

where $V = V(H) = V(D)$. Theorem 1.9 is based on similar arguments in a different, slightly more general setting, including an extension of the Cohn-Lempel equation [Tra09].

In fact, (1.20) is the definition of the vertex-nullity interlace polynomial for arbitrary simple graph with self loops [ABS04a, ABS04b].

Definition 1.13. *Let $G = (V, E)$ be a simple graph with self loops. The* vertex-nullity interlace polynomial *of G is defined as*

$$q_N(G;y) = \sum_{A \subseteq V} (y-1)^{\mathrm{n}(G[A])}.$$

Local complementation and edge pivot can be expressed by means of linear algebra, too. They are special cases of principal pivot transform [Tsa00]. We follow the discussion of Brijder and Hoogeboom [BH09a, BH09b]. Let us define principal pivot transform for the special case of symmetric matrices over $GF(2)$. If a matrix M can be written as

$$M = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$$

and A is invertible, the principal pivot transform with respect to A is defined as

$$\begin{pmatrix} A^{-1} & A^{-1}B \\ B^T A^{-1} & C - B^T A^{-1} B \end{pmatrix}.$$

Let a be a self looped vertex of a graph G with adjacency matrix

$$M = \begin{pmatrix} 1 & \mathbf{1} & \mathbf{0} \\ \mathbf{1}^T & A & B \\ \mathbf{0} & B^T & C \end{pmatrix}.$$

(We ordered the vertices as a, neighbors of a, the other vertices.) The adjacency matrix of G^a is the principal pivot transform of M with respect to the entry of M corresponding to the self loop at a, the upper left 1:

$$\begin{pmatrix} 1 & \mathbf{1} & \mathbf{0} \\ \mathbf{1}^T & A^C & B \\ \mathbf{0}^T & B^T & C \end{pmatrix}.$$

Edge pivot can be expressed by principal pivot transform on the adjacency matrix in the following way: Let G be a graph and vertices a and b be neighbors in G, where each of a and b does not have a self loop. We can write the adjacency matrix of G as

$$M = \begin{pmatrix} 0 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1}^T & \mathbf{1}^T & A_1 & A_2 & A_3 & A_4 \\ \mathbf{1}^T & \mathbf{0}^T & A_2^T & A_5 & A_6 & A_7 \\ \mathbf{0}^T & \mathbf{1}^T & A_3^T & A_6^T & A_8 & A_9 \\ \mathbf{0}^T & \mathbf{0}^T & A_4^T & A_7^T & A_9^T & A_{10} \end{pmatrix}.$$

(We ordered the vertices as a, b, common neighbors of a and b, neighbors only of a, neighbors only of b, other vertices.) Principal pivot transform on M with respect to the submatrix corresponding to $\{a, b\}$, the submatrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ in the upper left corner, is

$$\begin{pmatrix} 0 & 1 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1}^T & \mathbf{1}^T & A_1 & A_2^C & A_3^C & A_4 \\ \mathbf{0}^T & \mathbf{1}^T & (A_2^T)^C & A_5 & A_6^C & A_7 \\ \mathbf{1}^T & \mathbf{0}^T & (A_3^T)^C & (A_6^T)^C & A_8 & A_9 \\ \mathbf{0}^T & \mathbf{0}^T & A_4^T & A_7^T & A_9^T & A_{10} \end{pmatrix}.$$

This is the adjacency matrix of ${G^{ab}}_{ab}$ with the vertices ordered as a, b, neighbors of a and b, neighbors of only b (where b refers to the vertex b of ${G^{ab}}_{ab}$), neighbors of only a, other vertices.

Various Versions of the Interlace Polynomial

Early work on polynomials that count circuits has been performed by Martin [Mar77], including a polynomial that is equivalent to the vertex-nullity interlace polynomial but defined on the digraph instead of the interlace graph. Las Vergnas gave a generalization [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]. Arratia, Bollobás, and Sorkin coined the term "interlace polynomial" and defined the vertex-nullity interlace polynomial via (1.18) [ABS04a]. They also gave the following generalization in two variables that looks very similar to the classical Tutte polynomial (1.7).

Definition 1.14 ([ABS04b])**.** *Let $G = (V, E)$ be a simple graph with self loops. The two-variable interlace polynomial of G is*

$$q(G; x, y) = \sum_{A \subseteq V} (x-1)^{\mathrm{rk}\, G[A]} (y-1)^{\mathrm{n}\, G[A]}. \tag{1.21}$$

The vertex-rank interlace polynomial is $q_R(G; x) = q(G; x, 2)$.

Note that the vertex-nullity interlace polynomial (Definition 1.13) is $q_N(G;y) = q(G;2,y)$.

Aigner and van der Holst characterized the vertex-nullity polynomial in terms of the nullity of a matrix and introduced a version of the interlace polynomial that is defined by a recursion using local complementation [AvdH04]. All these versions of interlace polynomials are subsumed in the following multivariate interlace polynomial by Courcelle.

Definition 1.15 ([Cou08])**.** *Let $G = (V,E)$ be a simple graph with self loops. The* multivariate interlace polynomial *is defined as*

$$C(G;u,v,\mathbf{x},\mathbf{y}) = \sum_{\substack{A,B\subseteq V,\\ A\cap B=\emptyset}} x_A y_B u^{\mathrm{rk}((G\nabla B)[A\cup B])} v^{\mathrm{n}\,(G\nabla B)[A\cup B]}. \tag{1.22}$$

Traldi defined a weighted interlace polynomial [Tra10b], which can be obtained from C via substitution and multiplication with an easy to compute prefactor. Riera and Parker "provide spectral interpretations" of the interlace polynomial and define, among others, two interlace polynomials $Q(x,y)$ and Q_n^{HN} [RP06], which are also special cases of $C(G)$.

Glantz and Pelillo use the linear algebra approach to give a generalization of the interlace polynomial where, loosely speaking, adjacency matrices over arbitrary fields are considered [GP06].

We will use the following multivariate interlace polynomial, which relates to $q(G;x,y)$ as Sokal's multivariate Tutte polynomial relates to the classical $T(G;x,y)$.

Definition 1.16. *Let $G = (V,E)$ be a simple graph with self loops. We define*

$$\bar{q}(G;u,\mathbf{x}) = \sum_{A\subseteq V} x_A u^{\mathrm{rk}\,G[A]}.$$

This polynomial is a special case ($y = 0$ and $v = 1$) of Courcelle's and via multiplication with a prefactor and substitution closely related to Traldi's weighted interlace polynomial. The relation between $\bar{q}$ and q is as follows:

Lemma 1.17. *Let G be a graph. Then we have the polynomial identities $q(G;x,y) = \bar{q}(G;\frac{x-1}{y-1}, y-1)$ and $\bar{q}(G;u,x) = q(G;ux+1,x+1)$.* □

Independent Sets and Tutte-Martin Connection

Finally, let us mention two further properties of the interlace polynomial. An easy, but interesting observation is that $\mathrm{rk}\,G = 0$ if and only if G has no edges. This proves the following observation.

Proposition 1.18. *$\bar{q}(G;0,\mathbf{x})$ is the multivariate independent set polynomial $I(G;\mathbf{x})$.* □

As Arratia et al. observe, $q(G;2,1) = q_N(G;1)$ counts full-rank induced subgraphs of G [ABS04b, Section 5].

The "Tutte-Martin connection" states that 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. The connection is established via medial graphs. For every planar graph G one can build the oriented medial graph $\vec{G}_m$, find an Euler circuit C in $\vec{G}_m$ and obtain the interlace graph H of C. The whole procedure can be performed in polynomial time. For details we refer to Ellis-Monaghan and Sarmiento [EMS07].

Theorem 1.19 ([ABS04a, End of Section 7]; [EMS07, Theorem 3.1])**.** *Let G be a planar graph, $\vec{G}_m$ be the oriented medial graph of G, and H be the circle graph of some Euler circuit C of $\vec{G}_m$. Then $q_N(H;y) = T(G;y,y)$.*

1.3 Models for Computation

We state our results in several computational models. Unless stated otherwise, we use the Turing machine model.

1.3.1 Turing Machines

The Turing machine model is one model we use to state our results on computational complexity of graph polynomials. We are mainly interested in the running time of algorithms but we occasionally also consider space usage. Furthermore, complexity classes such as P, RP, and $\#\mathsf{P}$ refer to this model. The necessary definitions and basic results can be found in the literature [Pap94].

As our algorithms handle numbers and polynomials, let us discuss how these are represented on Turing machines. Non-negative integers are represented by their binary encoding. Non-negative rational numbers are encoded as the pair of their coprime numerator and denominator. Integers and rationals in general are represented by the encoding of their absolute value plus a bit for the sign. We also consider algebraic extensions of $\mathbb{Q}$. Elements of such a field can be represented as polynomials with coefficients in $\mathbb{Q}$. Unless stated otherwise, a polynomial will be represented as the sequence of its coefficients; a rational function will be represented as the pair of its numerator and denominator polynomial. Let us define a name for fields the elements of which we can represent by Turing machines.

Definition 1.20. *By a* Turing representable field *we mean Q or any field of rational functions over Q, where $Q = \mathbb{Q}$ or any algebraic extension of $\mathbb{Q}$.*

In particular, $\mathbb{Q}$ itself is a Turing representable field.

Definition 1.21. *Let $\mathbb{F}$ be a Turing representable field and $a \in \mathbb{F}$. We write $\|a\|$ to denote the length of the representation (binary encoding) of the number/rational function a.*

The length *of a number/rational function is the length of its representation (binary encoding).*

Computation on numbers/rational functions can be performed in time polynomial in the length of the operands. Of course, much more is known about this [GG03]. But we are mostly concerned with hardness results via polynomial Turing reductions and with exponential time algorithms. In this setting, the precise complexity of arithmetic operations is not an issue.

Graphs can be encoded as adjacency list or adjacency matrix. It is not relevant for our purposes which representation is used. Unless stated otherwise, we measure the running time of graph algorithms in the size of the input graph, i.e. in the number of its vertices.

In the Turing machine model, an algorithm is considered to be efficient if its running time (i.e. the number of steps the Turing machine performs) is bounded by a polynomial in the input size. Problems that allow for such an algorithm belong to the class P (languages that are decidable by a polynomial time bounded deterministic Turing machine) or FP (functions that are computable by a polynomial time bounded deterministic Turing machine).

Many computational problems that arise in the context of graph polynomials are not known to belong to FP but to a supposably larger class of functions, $\#\mathsf{P}$. The class $\#\mathsf{P}$ denotes the class of functions f such that there exists a polynomial time bounded nondeterministic Turing machine M such that on input x the number of accepting computations of M equals $f(x)$. It has been introduced by Valiant [Val79a, Val79b].

Even more famous is the class NP, which denotes the class of languages L such that there exists a polynomial time bounded nondeterministic Turing machine M such that, on input x, machine M has an accepting computations iff $x \in L$.

To give evidence that a problem P can not be solved efficiently (i.e. in polynomial time in the size of the input), we show that it is $\#\mathsf{P}$-hard or NP-hard. This means that a polynomial time algorithm for P would imply a polynomial time algorithm for *every* problem in $\#\mathsf{P}$ (NP, resp.). It is widely believed that $\#\mathsf{P}$-hard (NP-hard) problems can not be solved in polynomial time:

Conjecture 1.22. *(P vs. NP problem)*

$$\mathsf{P} \neq \mathsf{NP} \qquad \textit{and} \qquad \mathsf{FP} \neq \#\mathsf{P}.$$

To show that an algorithm for one problem implies an algorithm for another problem, we need a notion of reduction. The reductions we will use are defined as follows.

Definition 1.23. *Let f, g be two functions $\{0,1\}^* \to \mathbb{N}$.*

1. *f is* many-one reducible *to g if there are a polynomial time computable functions $r : \{0,1\}^* \to \{0,1\}^*$ and $s : \mathbb{N} \to \mathbb{N}$ such that $f(x) = s(g(r(x)))$ for all $x \in \{0,1\}^*$. We write $f \preceq^{\mathtt{m}} g$ if f is many-one reducible to g.*

2. *f is* Turing reducible *to g if there is polynomial time bounded Turing machine M that has oracle access to g and computes f. We write $f \preceq_T g$ if f is Turing reducible to g.*

Definition 1.24. *A function $g : \{0,1\}^* \to \mathbb{N}$ is* #P-hard *if every function $f \in \#\mathsf{P}$ can be reduced to g.*

Unless stated otherwise, statements about reducibility or hardness are meant with respect to Turing reductions.

1.3.2 Random Access Machines

For the analysis of the main algorithm in Chapter 7, we use the random access machine (RAM) model. These machines have a finite program and an infinite array of registers that can store natural numbers. A RAM can perform the basic arithmetic operations, comparisons, and conditional and unconditional jumps within the program. The registers can be addressed directly or indirectly (i.e. via other registers). The running time is measured in the number of operations the machine performs. This implies that every arithmetic operation is counted as one step. We will take care that the numbers that are processed are of polynomial length in the input size. A description of the RAM model can be found in the introductory parts of most books on design and analysis of algorithms, see, for instance, Cormen et al. [CLRS01] or Mehlhorn and Sanders [MS08].

In Chapter 6, we will state running time bounds and space bounds for algorithms computing on "ring elements". The underlying ring is not specified. But it is understood that we are working in the RAM model and there are registers that can store ring elements. Furthermore, the machine is able to perform ring operations on ring elements in the natural way, at unit costs.

Finally, our reductions and algorithms can be understood as algorithms in the BSS model as well [BCSS98]. In this model, the registers of the machine can store real or complex numbers, and the machine can perform arithmetic operations on such numbers in constant time. Many results that we state here over $\mathbb{Q}$ or some algebraic extension of it, immediately yield an analogous result in the BSS model that holds over $\mathbb{R}$ or $\mathbb{C}$.

1.3.3 Arithmetic Circuits

We will also consider a non-uniform, algebraic model of computation without branchings or loops, Valiant's VP and VNP. The definitions in this subsection are mostly taken from Bürgisser [Bür00].

Definition 1.25. *An* arithmetic circuit *is an acyclic 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. The size of an arithmetic circuit is the number of its nodes, the depth is the length of the longest directed path starting at an input node.

The complexity $L(p)$ *of a set of polynomials* $p \subseteq \mathbb{F}[X_1, \ldots, X_n]$, $\mathbb{F}$ *some field, is the minimum size of an arithmetic circuit computing* p *from inputs* $X_1, \ldots, X_n$ *and constants from* $\mathbb{F}$.

If g *is a polynomial in* k *variables and we extend the definition of an arithmetic circuit such that additionally nodes labeled* g *with indegree* k *are allowed, the circuit is called a* g*-oracle arithmetic circuit. The* oracle complexity $L^g(f)$ *of a polynomial* f *with respect to oracle polynomial* g *is the minimum size of a* g*-oracle arithmetic circuit computing* f.

Definition 1.26. *1. A function* $t : \mathbb{N} \to \mathbb{N}$ *is called* p-bounded *iff there exists some* $c > 0$ *such that* $t(n) \leq n^c + c$ *for all* n. *If we moreover have* $n^{1/c} - c \leq t(n) \leq n^c + c$ *for all* n, *the function* t *is called* p-bounded from above and below.

2. A sequence $f = (f_n)$ *of multivariate polynomials over a field* $\mathbb{F}$ *is called a* p-family, *if the number of variables and the degree are* p*-bounded functions of* n.

Note that, for all G-indexed graph polynomials that we have mentioned so far, we obtain a p-family if we consider the polynomials of $(G_n)_{n\in\mathbb{N}}$, where (G_n) is an infinite sequence of graphs such that G_n has n vertices.

Definition 1.27. *A* p*-family* $f = (f_n)$ *is said to be* p-computable *if the complexity* $L(f_n)$ *is a* p*-bounded function in* n. *The complexity class* $\mathsf{VP} = \mathsf{VP}_{\mathbb{F}}$ *consists of all* p*-computable families over* $\mathbb{F}$.

VP captures the notion of "efficient computation".

Definition 1.28. *A* p*-family* $f = (f_n)$ *is called* p-definable *iff there exists a* p*-computable family* $g = (g_n)$, $g_n \in \mathbb{F}[X_1, \ldots, X_{u(n)}]$, *such that for all* n

$$f_n(X_1, \ldots, X_{v(n)}) = \sum_{e\in\{0,1\}^{u(n)-v(n)}} g_n(X_1, \ldots, X_{v(n)}, e_{v(n)+1}, \ldots, e_{u(n)}).$$

The set of all p*-definable families form the complexity class* $\mathsf{VNP} = \mathsf{VNP}_{\mathbb{F}}$.

VNP is a class that captures many interesting arithmetic problems, similar to NP, which captures many interesting general computational problems. To identify problems that are hard for VNP, the following notions of reducibility have been introduced.

Definition 1.29. *1. A polynomial* f *is called a* projection *of a polynomial* g *iff* $f(X_1, \ldots, X_n) = g(a_1, \ldots, a_n)$ *for some* $a_i \in \mathbb{F} \cup \{X_1, \ldots, X_n\}$.

2. A p*-family* $f = (f_n)$ *is a* p-projection *of* $g = (g_m)$ *iff there exists a function* $t : \mathbb{N} \to \mathbb{N}$ *that is* p*-bounded from above and below such that* $\exists n_0 \forall n \geq n_0$: f_n *is a projection of* $g_{t(n)}$.

3. *A p-family $f = (f_n)$ is* c-reducible *(or* polynomial oracle reducible*) to a p-family $g = (g_n)$ iff there is a p-bounded function $t : \mathbb{N} \to \mathbb{N}$ such that $n \mapsto L^{g_{t(n)}}(f_n)$ is p-bounded.*

Definition 1.30. *A p-definable p-family f is called* VNP*-complete (with respect to a certain kind of reduction) iff every $g \in \mathsf{VNP}$ is reducible to f (with respect to this kind of reduction).*

We will use the following statement to argue that the multivariate interlace polynomial is p-definable.

Proposition 1.31 (Valiant's criterion [Bür00, Proposition 2.20])**.** *Let $\varphi : \{0,1\}^* \to \mathbb{N}$ be a function in $\#\mathsf{P}/\mathsf{poly}$. Then the family (f_n) of polynomials defined by*

$$f_n = \sum_{e \in \{0,1\}^n} \varphi(e) X_1^{e_1} \cdots X_n^{e_n}$$

is p-definable.

The permanent family is the classical example for a VNP-complete family with respect to p-projections. We will use the following modified version:

$$\mathrm{PER}_n^* = \sum_{\pi} \prod_{i} X_{i,\pi(i)}, \tag{1.23}$$

where the sum is over all injective partial maps $\{1, \ldots, n\} \to \{1, \ldots, n\}$ and the product is over all i for which $\pi(i)$ is defined.

Theorem 1.32 (Jerrum, cf. Bürgisser [Bür00, Theorem 3.7])**.** *The family* PER^* *is* VNP*-complete via p-projections.*

Another important p-family is the following, where a graph is *closed* if every vertex has even degree:

$$C(G; \mathbf{x}) = \sum_{\substack{A \subseteq E, \\ (V,A) \text{ closed}}} x_A. \tag{1.24}$$

The cubic lattice graph of dimension $n \times n \times 2$, C_n, is defined as follows: The vertex set is $\{(i,j,k) \mid 1 \le i,j \le n, 1 \le k \le 2\}$. There is an edge between two vertices a, b iff the (Euclidean) distance between a and b equals 1.

Theorem 1.33 (Jerrum [Jer81], cf. Bürgisser [Bür00, Theorem 3.9])**.** *The p-family $C(C_n; \mathbf{x})$ is $\mathsf{VNP}_{\mathbb{R}}$-complete via p-projections, where C_n denotes the cubic lattice graph of dimension $n \times n \times 2$.*

1.4 Computational Complexity of Graph Polynomials

A graph polynomials $P(G)$ of a graph G contains information about G. This information can be given by some or all coefficient(s) of $P(G)$. Or it can be the value of $P(G)$ at a particular point. For instance, the coefficients of $I(G;x)$, the independent set polynomial, are the number of independent sets of G of a particular size. $I(G;1)$ is the number of all independent sets of G.

One and the same graph polynomial $P(G)$ can contain different kinds information about the graph that apriori seems not to be related: By its definition, the chromatic polynomial evaluated at 3 equals the number of 3-colorings of a graph G. Interestingly, it also makes sense to evaluate it at -1: $\chi(G;-1)$ equals $(-1)^{|V|}$ times the number of acyclic orientations of G [Sta73]. As we have mentioned in Subsection 1.2.1, the Tutte polynomial contains even more information about a graph. Thus, we can say that a graph polynomial provides a means to capture many properties of a graph in a uniform way. Computing any of these properties constitutes a computational problem on graphs. If we are able to understand the computational complexity of a graph polynomial, we have characterized the computational complexity of all computational graph problems contained in this polynomial. This is a fruitful approach as the polynomial provides a way to connect the different graph problems, i.e. it provides reductions between computational problems.

The main problems we are concerned with in this work are evaluation and computation of a graph polynomial. Evaluation asks for the value of a graph's graph polynomial at some particular point, computation asks for a description of the graphs's graph polynomial, which typically is a list of its coefficients.

Definition 1.34 (Evaluation of a graph polynomial)**.** *Let $P : \mathcal{G} \to R[x]$ be a graph polynomial that maps G, a graph from $\mathcal{G}$, to $P(G;x)$, a polynomial in x over the ring R. Let $\xi \in R$. Then* evaluation of P at ξ *denotes the following computational problem:*

Name: $P(-;\xi)$
Input: G
Output: $P(G;\xi)$

Definition 1.35 (Computation of a graph polynomial)**.** *Let $P : \mathcal{G} \to R[x]$ be a graph polynomial that maps G, a graph from $\mathcal{G}$, to $P(G;x)$, a polynomial in x over the ring R. Then* computation of P *denotes the following computational problem:*

Name: $P(-;x)$
Input: G
Output: the x-coefficients of $P(G;x)$

Both definitions can be easily generalized to polynomials in more than one variable. It would also be possible to define a computational problem that asks for a particular coefficient of a graph polynomial. In a broader sense, we refer to all problems of this kind as "computing" a graph polynomial. However, our algorithms and

hardness results will only concern evaluation and computation as defined in Definition 1.34 and Definition 1.35.

Example 1.36. *1.* $\chi(-;3) = Z(-;3,-1) = \xi(-;3,-1,0)$ *is the problem of counting the* 3*-colorings of a graph.*

2. $\chi(-;-1) = Z(-;-1,-1) = \xi(-;-1,-1,0)$ *is the problem of counting acyclic orientations of a graph.*

3. $I(-;1) = \bar{q}(-;0,1) = P(-;2,1) = \xi(-;2,-1,1)$ *is the problem of counting the independent vertex sets of a graph.*

1.5 Our Contribution

The contribution of this thesis has two aspect: First, we provide new hardness results for several graph polynomials. This means that we identify large areas of the domains of the graph polynomials as areas where it is, at every single point, hard to evaluate or approximate the graph polynomial. Depending on the model, "hard" means either #P-hard, NP-hard, VNP-complete, or (almost) exponential-time hard under the exponential time hypothesis. Our main tool are graph transformations that allow for point-to-point reductions and finally interpolation. We devise new graph transformations, and we also apply known ones in a different context. We also develop a general framework to unify transformation based reductions which are used in different contexts but show the same internal structure. Using our theory, one can obtain a reduction for some graph polynomials rather quickly. Almost all of our new results fit into this framework.

On the other hand, we give new algorithmic results, i.e. algorithms to compute graph polynomials. Here, our main contribution concerns the interlace polynomial. In particular, we devise a notion that helps to compute the interlace polynomial using tree decompositions.

1.5.1 Algorithmic Results

If one considers the Tutte polynomial, say in its multivariate version (1.4), it is clear the it can be computed using $2^m\mathsf{poly}(n)$ operations on a graph with n vertices and m edges. Björklund, Husfeldt, Kaski, and Koivisto improved this to vertex exponential running time, i.e. $3^n\mathsf{poly}(n)$ operations and polynomial space or time and space $2^n\mathsf{poly}(n)$ [BHKK08]. In Chapter 6, we will observe that even the more general extended bivariate chromatic polynomial can be computed in vertex-exponential time using essentially the same technique.

Apart from this observation, our algorithmic results are with respect to the interlace polynomial. We will give improved algorithms for computing the interlace polynomial. By their definitions (Definitions 1.13, 1.14, 1.16), the two-variable interlace polynomial, its specializations and its multivariate generalization can be computed using $2^n\mathsf{poly}(n)$ operations. In general, this time bound also follows if one

uses recursions, such as (1.18), in a naive way. Courcelle's interlace polynomial can obviously be computed using $3^n\mathsf{poly}(n)$ operations. We will improve this slightly to $o(2^n)$ operations for the two-variable interlace polynomial and its direct relatives (Theorem 7.24), and to $o(3^n)$ operations for Courcelle's interlace polynomial (Theorem 7.25).

For simple graphs of bounded maximum degree, we are able to reduce the base of the exponentiation in the running time bound (Theorem 7.26). Furthermore, the polynomial $\bar{q}(G; u, \mathbf{x}) \bmod u^k$ can be evaluated using $(\sqrt{3})^n O(n^k)$ operations for arbitrary graphs with n vertices (Theorem 7.28). The technical ingredient is essentially the same in both cases: We observe that, due to restriction of the problem, not all branches of the obvious 2^n branching algorithm are needed.

Computation of the Interlace Polynomial Using Tree Decompositions

In fact, our $o(2^n)$ ($o(3^n)$, resp.) time algorithms for the interlace polynomial are a by-product of a new technique to compute the interlace polynomial using tree decompositions. Developing this new technique constitutes a major part of Chapter 7. Our aim is to obtain a parameterized algorithm[9] for evaluating Courcelle's multivariate interlace polynomial C on graphs of bounded treewidth.

Previously it has already been known that evaluation of the interlace polynomial parameterized by the treewidth of the input graph is fixed parameter tractable. This follows as the interlace polynomial is fixed parameter tractable with cliquewidth as parameter [Cou08, Theorem 23, Corollary 33], which, in turn, is a consequence of the fact that the interlace polynomial is monadic second order logic definable (MS_1 definable as defined by Courcelle, Makowsky, and Rotics [CMR01]; see also Courcelle [Cou08, Section 5]).[10] 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 treewidth

[9] Whereas classically one measures the running time of an algorithm only as a function of the input size, parameterized algorithms are measured depending on the input size and an additional parameter [DF99, FG06, Nie06]. In this way, one can identify algorithms for hard problems that are efficient for large instances if the parameter is small.

[10] Note the following crucial difference with respect to monadic second order logic definability: MS_1 definable evaluation problems are fixed parameter tractable with *cliquewidth* as parameter [CMR01, Theorem 31]. MS_1 is a logic that allows one-sorted structures, the universe of which consists of the vertices of the graph. Set variables range over vertex subsets. On the contrary, MS_2 is a logic that allows two-sorted structures, the universe of which consists of the vertices *and edges* of the graph. Set variables range over vertex subsets or edge subsets, which, for instance, enables the definition of the Tutte polynomial in MS_2. MS_2 definable evaluation problems are known to be fixed parameter tractable with *treewidth* as parameter [CMR01, Theorem 32]. We can not expect that this generalizes to cliquewidth, see Fomin, Golovach, Lokshtanov, and Saurabh [FGLS10].

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 treewidth 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 [Tra10b].)

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 Chapter 7, 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 $2^{3k^2+O(k)}n$ RAM-operations to evaluate Courcelle's multivariate interlace polynomial (and thus any other version of the interlace polynomial mentioned in Section 1.2.2) on an n-vertex graph given a tree decomposition of width k (Theorem 7.19). The algorithm can also be understood as a procedure to construct an arithmetic circuit that describes (i.e., in the sense of algebraic complexity theory, computes) the interlace polynomial (Section 7.6.1), and it can be implemented in parallel using depth polylogarithmic in n (Section 7.6.2, Theorem 7.21). 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 7.6.3, Corollary 7.23).

Techniques

As we have seen, the Tutte polynomial and the interlace polynomial are similar in some respect: Both can be defined by a recursion using a graph operation ((1.5) and (1.18)), both can be defined as closed sums over edge/vertex subsets involving some kind of rank ((1.7) and (1.21)). 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 edge pivot on a graph, it is not clear how to obtain a tree decomposition of the modified graph. In particular, a single edge pivot operation can turn a tree (treewidth 1) into a cycle (treewidth 2), see Figure 1.5.

Another problem is that in the Tutte case the recursion formula naturally generalizes from the simplest versions (chromatic polynomial) to the most general ones

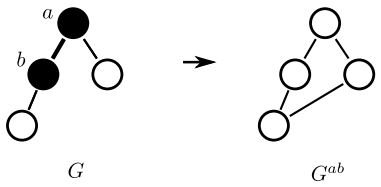

Figure 1.5: Edge pivoting, a central graph operation for the interlace polynomial, increases treewidth.

(it is the defining recursion of the Bollobás-Riordan graph invariant [BR99]; cf. also the recurrence for the extended bivariate chromatic polynomial (1.12)). The interlace polynomial, in contrast, needs more and more complicated recursions when generalizing the vertex-nullity interlace polynomial to the multivariate interlace polynomial[11] (see Courcelle [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, if a graph is extended by a set of vertices and some edges between the old and the new vertices, the set of all partitions of the new vertices captures all possible types of "behavior" of the rank (i.e. number of connected components) when the new vertices and some or all of the new edges are added. – 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 technical contribution of Chapter 7 is to devise such a tool and to prove that it works well with tree decompositions. Our approach is not via logic but via the $GF(2)$-rank of adjacency matrices, which is specific to the interlace polynomial. The main idea is explained in Section 7.2.

1.5.2 Hardness Results

To give nontrivial lower bounds for any interesting computational problem in any general model of computation seems to be out of reach of current research. Thus, the usual approach is as follows: 1. Define a class that contains many interesting problems. 2. Define a notion of reduction in this class such that the class is closed under this reduction[12]. 3. For a particular problem, show that it is "hard" in

[11] But note that Traldi reduced a three-term recursion to a two-term recursion [Tra10b, Corollary 2.4].

[12] This is the ideal case. In practice, there are several subtleties. For example, the class #P, which we will use in the following, is not closed under subtraction. However, providing a polynomial time algorithm for an arbitrary single #P-hard problem immediately yields a polynomial time

the sense that every problem of the class can be reduced to it. Thus, an efficient algorithm for the hard problem would immediately yield an efficient algorithm for every problem of the class. In this way, hardness results are given as a substitute for lower bounds. We will give hardness results for the evaluation (computation, resp.) of graph polynomials in various settings.

Exact Evaluation in Polynomial Time?

Almost all graph polynomials P discussed in this work have some evaluation point ξ where $P(-;\xi)$ is equivalent to some well-known counting problem, which is already known as #P-hard.

Theorem 1.37. *1. Counting 3-colorings in a graph, i.e. $\chi(-;3)$, is #P-hard. Note that the Tutte polynomial, the bivariate chromatic polynomial, and the extended bivariate chromatic polynomial all contain the chromatic polynomial.*

2. Counting independent sets in a graph, i.e. $I(-;1)$, is #P-hard. The independent set polynomial is contained in the bivariate chromatic polynomial, in the bivariate interlace polynomial q, in the multivariate interlace polynomial $\bar{q}$, and in Courcelle's multivariate interlace polynomial C.

Proof. Garey, Johnson, and Stockmeyer proved that deciding 3-coloring is NP-hard [GJS76]. It is usually claimed without giving the details that (a modification of) the techniques working for the decision version do also work for the counting problem [Wel93, Page 7, (1.3.5)], [Pap94, Chapter 18], [JVW90, (5.12)]. For an actual construction, we refer to Dell et al. [DHW10, Lemma 7 and 8].

#P-hardness of counting independent sets is generally known since Valiant introduced #P in 1979 [Val79b]. For an actual construction, we refer to the proofs of Lemma 5.1 and 5.2. □

Jaeger, Vertigan, and Welsh completely characterized the complexity of evaluating the Tutte polynomial:

Theorem 1.38 ([JVW90])**.** *Let $(x,y) \in \mathbb{Q}(i,j)$, where $i^2 = -1$ and $j = e^{2\pi i/3}$. Unless $(x-1)(y-1) = 1$ or (x,y) is one of $(1,1)$, $(0,0)$, $(0,-1)$, $(-1,0)$, $(i,-i)$, $(-i,i)$, (j,j^2), (j^2,j), the computational problem $T(-;x,y)$ is #P-hard with respect to Turing reductions.*

This means that evaluating the Tutte polynomial is #P-hard "almost everywhere", where "almost everywhere" means that, while the Tutte polynomial is defined on a two-dimensional set (all $(x,y) \in \mathbb{Q}(i,j)^2$), the exceptional set is only of dimension one, consisting of a curve and nine points.

Vertigan proved a result which basically states that the Tutte polynomial is #P-hard to evaluate almost everywhere even if we restrict the input to planar graphs [Ver05]. We use the following special case.

algorithm for every problem in #P.

Theorem 1.39 ([Ver05]). *Evaluating the Tutte polynomial T of planar graphs at (α, α) is #P-hard for all $\alpha \in \widetilde{\mathbb{Q}}$, except for $\alpha \in \{0, 1, 2, 1 \pm \sqrt{2}\}$.*

One contribution of this work is to generalize Theorem 1.38 to the extended bivariate chromatic polynomial: ξ is defined on a set of dimension 3, but we prove that only on a set of dimension at most 2, ξ is not #P-hard to evaluate (Theorem 4.22). In this context, we also prove that the bivariate chromatic polynomial is #P-hard to evaluate everywhere except on a set of dimension at most 1 (Theorem 4.19).

We also give a similar statement for the two-variable interlace polynomial $q(G; x, y)$. Before our work, the following has been known: The two-variable interlace polynomial $q(G; x, y)$ contains counting independent sets, and, because of the Tutte-Martin connection and Theorem 1.39, the vertex-nullity interlace polynomial $q_N(G; y)$ must be #P-hard to evaluate almost everywhere.[13] This means that there is a set of dimension 1 on which the two-variable interlace polynomial $q(G; x, y)$ is #P-hard to evaluate. We show (Theorem 4.9, see also Figure 4.1) that, in fact, the two-variable interlace polynomial is #P-hard to evaluate everywhere on the two-dimensional plane, except on 5 lines (where the complexity is unknown or it is polynomial time computable). This answers a question of Arratia, Bollobás, and Sorkin [ABS04b, Page 579].

As mentioned before, our main technique are graph transformations, point-to-point reductions, and interpolation. For the bivariate chromatic polynomial, we show that Linial's construction can be used, see the discussion at the beginning of Section 4.3. Then we add edge thickening to obtain the final result for the extended bivariate chromatic polynomial. For earlier uses of edge thickening, see the beginning of Section 3.1. The new #P-hardness results concerning the interlace-polynomial are based on vertex cloning, which is discussed more detailed at the beginning of Section 3.1, too. Additionally, we use new graph transformations: adding "combs" or cycles to vertices, see Section 3.2.2 and Section 3.2.3.

Exact Computation in Subexponential Time?

The theory of NP- or #P-hard problems serves as a substitute for superpolynomial lower bounds for decision problems (NP) or counting problems (#P). For many problems, the best known algorithms take exponential time. To argue that these algorithms can not be improved, the exponential time hypothesis (ETH) has been introduced, as well as a notion of reduction that preserves subexponential complexity [IPZ01]. Dell, Husfeldt, and Wahlén recently introduced a counting version of the exponential time hypothesis, which is more appropriate for the evaluation of graph polynomials [DHW10]. We state this hypothesis as "#ETH" below. It claims a lower bound for the following problem.

[13] It seems, however, that no one has been interested in stating this clearly, i.e. considering individual evaluation points of the vertex-nullity interlace polynomial. Ellis-Monaghan and Sarmiento only state that "the vertex-nullity interlace polynomial is #P-hard in general" [EMS07, Corollary 3.2].

Name: #3SAT
Input: Boolean formula φ in 3CNF with m clauses in n variables
Output: Number of satisfying assignments for φ

#ETH (Dell, Husfeldt, Wahlén [DHW10]). *There is a constant $c > 0$ such that no deterministic algorithm can compute #3SAT in time* $\exp(c \cdot n)$.

Using this hypothesis, Dell et al. show that, at almost every point, evaluating the Tutte polynomial of a graph with m edges needs time $\exp(\Omega(m/\log^3 m))$. (In fact, for some points they give even the stronger bound $\exp(\Omega(m))$, which matches the upper bound.)

As a basis, they show that exponential time in the number of variables, as the #ETH is concerned with, is equivalent to exponential time in the number of clauses. Impagliazzo, Paturi, and Zane proved this for the decision version first [IPZ01].

Theorem 1.40 ([DHW10, Theorem 1]). *For all $d \geq 3$, #ETH holds if and only if #d-SAT requires time $exp(\Omega(m))$.*

As one would expect, #d-SAT, d a positive integer, is the following problem:

Name: #d-SAT
Input: Boolean formula φ in d-CNF with m clauses in n variables
Output: Number of satisfying assignments for φ

In Chapter 5, we transfer the technique of Dell et al. to the interlace polynomial. We give a reduction along the x-axis for $\bar{q}(G; u, x)$ that loses only a factor $\log^3 n$ (Theorem 5.3). As we also give a subexponential time preserving reduction from #3SAT to counting independent sets (Section 5.1), we can conclude that evaluation of the independent set polynomial needs time $\exp(\Omega(n/\log^3 n))$ at every point $x \in \mathbb{Q} \setminus \{0\}$ (unless #ETH fails), see Corollary 5.4.

Our result is based on a combination of vertex cloning and path addition to vertices.

Approximation in Randomized Polynomial Time?

Besides exact counting, approximation algorithms for various graph problems have been considered.

Definition 1.41 (FPRAS). *Let $f : \mathcal{G} \to R$ be a mapping from graphs to the rationals or some finite-dimensional, totally ordered extensions of the rationals. A* fully polynomial randomized approximation scheme *(FPRAS) for f is a randomized algorithm that, given a graph G with n vertices and an error tolerance $\varepsilon, 0 < \varepsilon < 1$, runs in time polynomial in n and $1/\varepsilon$ and returns $\tilde{f}(G) \in R$ such that*

$$\Pr[2^{-\varepsilon} f(G) \leq \tilde{f}(G) \leq 2^{\varepsilon} f(G)] \geq \frac{3}{4}.$$

Definition 1.42. *Let $f : \mathcal{G} \to R$ be a mapping from graphs to the rationals or some finite-dimensional, totally ordered extensions of the rationals. A randomized $2^{n^{1-\varepsilon}}$-*approximation *algorithm for f is a randomized algorithm that, given a graph G with n vertices, runs in time polynomial in n and returns $\tilde{f}(G) \in R$ such that*

$$\Pr[2^{-n^{1-\varepsilon}} f(G) \leq \tilde{f}(G) \leq 2^{n^{1-\varepsilon}} f(G)] \geq \frac{3}{4}.$$

It is known that approximately counting independent sets is hard in a very strong sense, for example:

Theorem 1.43 ([Rot96, Lemma A.3])**.** *For every $\varepsilon > 0$, approximating the number of independent sets of a graph on n vertices within $2^{n^{1-\varepsilon}}$ is* NP*-hard.*

The proof is based on the blow-up technique by Jerrum, Valiant, and Vazirani [JVV86]. A similar inapproximability result appears in Sinclair's Thesis [Sin88].

Vadhan proves that this inapproximability result holds even if the input is restricted to graph of maximum degree 3 [Vad95].

Luby and Vigoda give an inapproximability result for evaluation of the independent set polynomial for graphs of bounded degree [LV97]. The range where the inapproximability applies depends on the maximum degree of the graph and is not explicitly given:

Theorem 1.44 ([LV97, Theorem 4])**.** *Unless* RP = NP*, there is no algorithm to approximately compute $I(G;\xi)$ within any polynomial factor when $\xi > \frac{c}{\Delta}$ for some constant $c > 0$ (Δ is the maximum degree of the graph).*

Finally, let us mention the following result of Dyer, Frieze, and Jerrum:

Theorem 1.45 ([DFJ02])**.** *Unless* NP = RP*, there is no polynomial time algorithm which estimates the logarithm of the number of independent sets in a Δ-regular graph ($\Delta \geq 25$) within relative error at most $\varepsilon = 10^{-6}$.*

As a by-product of the graph transformations we develop for the interlace polynomial, we give an inapproximability result for the independent set polynomial that is not completely covered by these known results: For general graphs, we obtain the same strong inapproximability result as Roth, but for almost every point of the plane (Theorem 4.12).

Jerrum and Goldberg give an inapproximability result for the Tutte polynomial [GJ07].

Arithmetic Circuits of Polynomial Size?

We observe in Theorem 2.2 that the multivariate version of the interlace polynomial of Arratia, Bollobás, and Sorkin is VNP-complete via p-projections.

For the multivariate Tutte polynomial, we give a hardness result with respect to c-reductions. Lotz and Makowsky observe that the multivariate Tutte polynomial

$Z(G; q, \mathbf{x})$ is VNP-complete [LM04]. In fact, they argue that already $Z(G; 2, \mathbf{x})$ is VNP-complete. We show that this also holds for the other values of q, $q \notin \{0, 1\}$ (Theorem 2.4). To our knowledge, this is the first result in the VNP model that reduces from one specialization of a graph polynomial to another specialization of the same polynomial. Technically, we use a strengthened version of Linial's construction, see the discussion on Page 50, in particular compare (2.3) and (2.4).

1.5.3 General Results

Our hardness results are not just an accumulation of theorems about individual graph polynomials. We extract important ideas and put them into a general framework.

Johann A. Makowsky describes a research program to develop a general theory of graph polynomials [Mak08]. One class of graph polynomials are the ones "counting the number of (induced) subgraphs of a certain kind" [Mak08, Page 545]. In this work, we observe that a particular graph transformation yields a powerful family of point-to-point reduction for many graph polynomials of this class. Via interpolation, these reductions finally lead to the following result: While we can count the induced subgraphs of the desired property with respect to different weights, these weights do not influence the polynomial time computability of the counting problem (i.e. graph polynomial evaluation problem): Using (almost) any weight leads to the same complexity as evaluation with all weights set to 1 (Theorem 3.19).

As a formal framework, we define vertex/edge induced subgraph polynomials (VSPs / ESPs) in Section 3.1. We define them as multivariate polynomials, where every vertex/edge has its own weight variable. This is inspired by Alan D. Sokal, who pointed out that considering multivariate versions of graph polynomials helps greatly to simplify the technical derivations [Sok04, Sok05, SS05]. In fact, physicist are using graph models where vertices and edges are weighted individually for many years [FK72]. Individual weights are also important in knot theory.

As we discuss at the end of Section 3.1, our framework can be directly applied to the independent set polynomial, the Tutte polynomial, the interlace polynomial, and the extended bivariate chromatic polynomial. If we consider graphs with multiple edges, it can be applied to matching polynomials, too.

Our second general contribution to the complexity of graph polynomials is in Section 3.3. There, we point out a general property ("local uniformity", Definition 3.40) that a graph transformation can have with respect to a certain graph polynomial generating function. For several graph transformations (vertex/edge cloning, edge stretching, adding "combs", cycles, or paths), it is this property that ensures to obtain a point-to-point reduction for the respective graph polynomial. To state our observations formally, we introduce vertex/edge function polynomials (VFPs/EFPs). These generalize VSPs/ESPs as they do not just count induced subgraphs with certain properties, but any "part" of a graph that is determined by a vertex/edge subset. We prove that every graph transformation that is local uniform with respect to a VFP/EFP generating function immediately yields a

point-to-point reduction for the respective VFP/EFP (Theorems 3.45 and 3.49). To demonstrate our framework, we give a new point-to-point reduction for the vertex cover polynomial (Example 3.51) and a polynomial counting dominating sets (Example 3.52).

Let us discuss our results in the context of the following difficult point conjecture for graph polynomials [Mak08, Conjecture 1]:

Conjecture 1.46 (Makowsky)**.** *Let $P(G; x_1, \ldots, x_r)$ be an extended SOL-definable graph polynomial in r indeterminates. The complexity of $P(-; \xi_1, \ldots, \xi_r)$ of the polynomial $P(G; x_1, \ldots, x_r)$ is described as follows: There is a set $B \subseteq \mathbb{C}^r$ such that*

1. *for all $\xi^{(0)} \in B$, the evaluation problem $P(-; \xi^{(0)})$ is solvable in polynomial time (in the BSS model),*
2. *for all $\xi^{(0)}, \xi^{(1)} \in \mathbb{C}^r \setminus B$, the evaluation problem $P(-; \xi^{(0)})$ is polynomial time oracle reducible to $P(-; \xi^{(1)})$ (in the BSS model), and*
3. *B is a finite union of algebraic sets in $\mathbb{C}^r$ of dimension strictly less than r.*

The conjecture implies: If there is one difficult (i.e. hard to evaluate) point outside B, all points outside of B are difficult, as they are mutually reducible.

Our results on individual graph polynomials support the difficult point conjecture in the following way:

- For the (extended) bivariate chromatic polynomial, almost all the points fulfilling the precondition of Theorem 4.22 or Theorem 4.19 are mutually reducible. Still, if we only use the reductions as stated in the "moreover" part of the theorems, we obtain exception sets B that are not a *finite* union of sets of lower dimension. However, we can exploit the following "dirty trick": It is not hard to see that, for every triple (x, y, z) of natural numbers, $\xi(-; x, y, z) \in \#\mathsf{P}$. Thus, computing the the coefficients of $\xi(G; X, Y, Z)$ can be performed using a polynomial (in the size of G) number of oracle calls to a $\#\mathsf{P}$-hard problem. This $\#\mathsf{P}$-hard problem can be evaluation at any point $(x_0, y_0, z_0) \in \mathbb{Q}^3$ that has been identified as a $\#\mathsf{P}$-hard evaluation point of ξ. Thus, every two $\#\mathsf{P}$-hard evaluation points of Theorem 4.19 or Theorem 4.22 are mutually reducible. This yields an exception set of dimension 1 (Theorem 4.19) or 2 (Theorem 4.22). This, in turn, partially supports the difficult point conjecture. (We can not be sure that all points outside of B are polynomial time computable.)
- For the independent set polynomial, our results support the difficult point conjecture with the exceptional set $B = \{0\}$ (use the proof of Corollary 4.8).
- For the interlace polynomial $\bar{q}(G; u, x)$, we have reducibility along the x-axis, see Section 3.2. If we could also prove reducibility along the u-axis for an appropriate x, and we could also handle the three lines of unknown complexity, it would be shown that $\bar{q}$ fulfills the statement of the difficult point conjecture.

In general, we can say that Theorem 3.19 shows the following about VSPs/ESPs that are generated by a graph polynomial that supports cloning[14]: To verify the difficult point conjecture for a VSP/ESP P, we, basically, have to consider only the specialization of P with all weights set to 1. Or, in other words, identifying a graph polynomial as VSP/ESP of a cloning supporting generating graph polynomial means to reduce its complexity by one dimension.

Finally, let us mention that all polynomial time Turing reductions for evaluation / computation of graph polynomials that we establish in this work are actually uniform algebraic reductions in the sense of Bläser, Dell, and Makowsky [BDM10]. In particular, the proof of Theorem 3.19 yields:

Theorem 1.47. *Let $f : \mathcal{G} \to \mathbb{Q}$ be a graph polynomial that supports vertex cloning and $F = \mathrm{VSP}\, f$. Then, for every $\xi_0 \in \mathbb{Q} \setminus \{0, -1, -2\}$, we have*

$$F \preccurlyeq^{\mathrm{p}}_{\mathrm{A}U} F(-; \xi_0),$$

which means that the parameterized numeric graph invariant $F : \mathcal{G} \times \mathbb{N} \to \mathbb{Q}$, $(G, k) \mapsto F(G; k)$ algebraically reduces to the numeric graph invariant $G \mapsto F(G; \xi_0)$ uniformly in polynomial time.

A completely analogous statement holds for edge cloning and ESPs.

[14] To express the idea clearly, we ignore the special cases of weights -2, -1, and 0 for a moment.

2 VNP-complete Graph Polynomials

In this chapter, we give hardness results for graph polynomials with respect to an algebraic model. We show that the interlace polynomial is VNP-complete via p-reductions. To establish p-definability (Lemma 2.1), we use Valiant's criterion. The hardness results follows from a simple reduction of the partial permanent to independent sets (Theorem 2.2).

In Section 2.2, we prove that almost every restriction $Z(G; q, \mathbf{x})$, $q \in \mathbb{C}$, of the Tutte polynomial is VNP-complete via c-reductions (Theorem 2.4). We use a graph transformation that enables a reduction. This is based on Linial's construction [Lin86], but we have to prove a stronger statement than the usual reduction for the chromatic polynomial.

2.1 VNP-Completeness of the Interlace Polynomial

Lemma 2.1. *Let $(G_k)_{k\in\mathbb{N}}$ be a sequence of graphs such that $k \mapsto |V(G_k)|$ is injective and p-bounded. Then the sequences $\big(\bar{q}(G_k; u, \mathbf{x})\big)_{k\in\mathbb{N}}$ and $\big(C(G_k; u, v, \mathbf{x}, \mathbf{y})\big)_{k\in\mathbb{N}}$ are p-definable p-families of multivariate polynomials.*

Proof. Let us prove the statement for $\bar{q}$ using Valiant's criterion (Proposition 1.31).

For every $n \in \mathbb{N}$ such that there is a $k \in \mathbb{N}$ with $|V(G_k)| = n/2$, define $\varphi(e_1, \ldots, e_n)$ as follows. Fix an order on $V(G_k)$, and let $v_1, \ldots, v_{n/2}$ be the elements of $V(G_k)$ in increasing order. For $(e_1, \ldots, e_n) \in \{0,1\}^n$, define

$$A(e_1, \ldots, e_{n/2}) = \{v_i \mid e_i = 1\}$$

and

$$\varphi(e_1, \ldots, e_n) = \begin{cases} 1 & \text{if } \operatorname{rk} G_k[A(e_1, \ldots, e_{n/2})] = \ell \text{ and } e_{n/2+1} \ldots e_n = 1^\ell 0^{n/2-\ell} \\ 0 & \text{otherwise.} \end{cases}$$

For all other $n \in \mathbb{N}$, let $\varphi(e_1, \ldots, e_n) = 0$ for all $(e_1, \ldots, e_n) \in \{0,1\}^n$. Using an encoding of G_k as advice, we see that $\varphi \in \mathsf{FP/poly}$. By Valiant's criterion,

$$f(X_1, \ldots, X_n) = \sum_{e\in\{0,1\}^n} \varphi(e) X_1^{e_1} \cdots X_n^{e_n}$$

is in VNP. Furthermore, $\tilde{f}(X_1, \ldots, X_n, U) := f(X_1, \ldots, X_n, \underbrace{U, \ldots, U}_{n \text{ times } U})$, is p-definable as well. For $k \in \mathbb{N}$ and $n = |V(G_k)|$, we have $\tilde{f}(x_1, \ldots, x_n, u) = \bar{q}(G_k; u, \mathbf{x})$. As

$k \mapsto |V(G_k)|$ is p-bounded and injective, this implies that $\big(\bar{q}(G_k; u, \mathbf{x})\big)_k$ is a p-family and p-definable.

The statement for C can be proved similarly. □

As the interlace polynomial is a polynomial over vertex subsets, we can not obtain *every* graph from the complete graph by just setting some variables to zero. For this reason, we really need the particular family of graphs in the following theorem and can not just use the K_n family. The proof of the following Theorem follows closely Bürgisser's proof that the (edge based) clique polynomial is VNP-complete [Bür00, Theorem 3.10].

Theorem 2.2. *There exists a polynomial time constructible family G_n of graphs such that the family $\bar{q}(G_n; u, \mathbf{x})$ is VNP-complete via p-projections. This also holds for C, the multivariate interlace polynomial of Courcelle.*

Proof. We let $G_n = ([n] \times [n], E_n)$, where E_n is such that two vertices are neighbors if and only if they belong to the same column or row (i.e. $\{(i,j),(i',j')\} \in E_n$ iff $i = i'$ or $j = j'$, but $(i,j) \neq (i',j')$). G_n can be computed in polynomial time in n. The independent sets of G_n correspond to the injective partial maps $[n] \to [n]$. Thus, we can write the family of partial permanents, which is VNP-complete, as a p-projection of the interlace polynomials of G_n:

$$\bar{q}(G_n; u = 0, \mathbf{x}) = \sum \{x_A \mid A \text{ independent set in } G_n\} = \mathrm{PER}_n^* .$$

Thus, PER* is a p-projection of $(\bar{q}(G_n))$ and also of $(C(G_n))$. The families $(\bar{q}(G_n))$ and $(C(G_n))$ are p-definable by Lemma 2.1. □

2.2 On the VNP-Completeness of the Tutte Polynomial

Lotz and Makowsky [LM04] note that the Tutte polynomial

$$Z(G; q, \mathbf{x}) = \sum_{A \subseteq E} q^{k(A)} \prod_{e \in A} x_e$$

is VNP-complete via c-reductions. Their statement is based on the following relation.

Lemma 2.3 ([LM04, Lemma 15])**.** *Let $G = (V, E)$ be an edge-weighted graph and $C(G)$ be the generating function of closed[1] subgraphs with equal weight, i.e.*

$$C(G; \mathbf{x}) = \sum_{\substack{A \subseteq E, \\ (V,A) \text{ closed}}} x_A .$$

Then the following identity holds:

$$C(G; \mathbf{w}) = 2^{-|V|} \Big(\prod_{e \in E} (1 - w_e) \Big) Z\Big(G; 2, \frac{2w_e}{1 - w_e}\Big). \tag{2.1}$$

[1] A graph is called *closed* iff all vertices have even degree.

Note that the term $2^{-|V|}$ of (2.1) is missing their paper. It stems from the fact that the number of mappings $\sigma : V \to \{-1, +1\}$ is $2^{|V|}$. Therefore, a factor of $2^{|V|}$ should be inserted in [LM04, Page 342] on lines 4 and 5 from below.

The p-family $C(C_n; \mathbf{x})$ is VNP-complete (via p-projections), where C_n denotes the cubic lattice graph of dimension $n \times n \times 2$ (Theorem 1.33). This fact and (2.1) imply that the Tutte polynomial Z is VNP-complete via c-reductions. Let us mention some details of the reduction. First, we observe that we can produce the factors $2^{-|V|}$ and $\prod_{e \in E}(1 - w_e)$ easily. We only have to add a certain number of graphs H_i to G. Each H_i consists of an isolated edge, the weight w_i of which is chosen such that $Z(H_i; 2, w_i)$ equals $\frac{1}{2}$ or $1 - w_e$. Now, given a graph G and its edge variables, we have to build an arithmetic circuit computing $Z(G; 2, \frac{2w_e}{1-w_e})$. The circuit may use additions, multiplications, and oracle calls to $Z(G; q, \mathbf{x})$. If divisions are allowed, too, we can compute $\frac{2w_e}{1-w_e}$ directly and obtain $Z(G; 2, \frac{2w_e}{1-w_e})$. However, in the usual definition of c-reductions (oracle reductions, Definition 1.29), divisions are not allowed. The standard method to avoid divisions [Str73] is by splitting the polynomial into its homogeneous parts. Then each division is simulated by a multiplication by a truncation of the inverse power series of the divisor. Unfortunately the usual definition of c-reductions allows oracle access only to Z and not the homogeneous parts of Z. Thus, to formally justify our statement that the Tutte polynomial is VNP-complete, we either have to allow divisions in the reduction or we must assume access to the homogeneous parts of the function to which is being reduced to. However, this is only a formal issue: If we are given an arithmetic circuit A_n for the Tutte polynomial of a graph with n vertices and A_n has polynomial size in n, then we do have access to its homogeneous components. We only need to maintain the homogeneous parts of each gate of the circuit, which increases the size of the circuit only polynomially in n. Thus, (2.1) immediately gives us a polynomial size (in n) arithmetic circuit A'_n for the generating function of the closed subgraphs. If there are any divisions in A'_n, we can avoid them in the usual way and still maintain the polynomial size of A'_n.

In the rest of this work we assume that divisions are allowed in c-reductions.

The multivariate Tutte polynomial is VNP-complete via c-reductions. In fact, (2.1) gives a stronger statement: the specialization of the multivariate Tutte polynomial with $q = 2$ is VNP-complete. What about other specializations? For $q = 1$, of course, $Z(G; 1, \mathbf{x})$ is p-computable as $Z(G; 1, \mathbf{v}) = \prod_{e \in E}(1 + x_e)$. Similarly, $Z(G; 0, \mathbf{x}) = 0$. For all other values of q, we have the following statement.

Theorem 2.4. *Let K_n be the complete graph with n vertices. For every $q \in \mathbb{C} \setminus \{0, 1\}$, the family $Z(K_n; q, \mathbf{x})$ is VNP-complete via c-reductions.*

Note that a similar statement holds for the classical Tutte polynomial in the Boolean model of computation: Jaeger et al. prove that at every point of the plane, the Tutte polynomial is #P-hard to evaluate, except at a hyperbola and a set of nine special points [JVW90]. Their proof works by evaluating the Tutte polynomial at many points and interpolation. In this way, many evaluations of the polynomial eventually yield the *coefficients* of the polynomial with respect to one variable.

Our proof uses the same idea. We want to obtain the coefficients of Z considered as a polynomial in q over $\mathbb{C}[\mathbf{x}]$. The construction we use is a variation of a nice observation for the chromatic polynomial:

Lemma 2.5 ([Lin86, JVW90])**.** *Let G be a graph and G' be the graph obtained from G by inserting a new vertex v' and connecting all vertices of G to v'. Then we have the polynomial identity*

$$\chi(G';q) = q \cdot \chi(G;q-1). \tag{2.2}$$

As it is short and beautiful, we present the well-known proof by Linial.

Proof of Lemma 2.5. Vertex v' must obtain a color different from all colors of the other vertices. Thus, every q-coloring of G' implies an $(q-1)$-coloring of G. On the other hand, every $(q-1)$-coloring of G corresponds to exactly q q-colorings of G'.

This show that (2.2) holds for every $q \in \mathbb{N} \setminus \{0\}$, which are infinitely many. Thus, (2.2) also holds as an identity for polynomials in q. □

Let us write the chromatic polynomial as a sum over all edge induced subsets (cf. (1.4) and (1.6)). An edge subset $A' \subseteq E(G')$ corresponds to an edge subset $A \subseteq E$ and an edge subset $\hat{B}$, where $\hat{B}$ is a subset of the edges connecting the vertices in V to v'. Thus, (2.2) can be written as

$$\sum_{A \subseteq E(G)} \sum_{B \subseteq V(G)} q^{|\mathcal{C}G'[A \cup \hat{B}]|}(-1)^{|A \cup \hat{B}|} = \sum_{A \subseteq E(G)} q(q-1)^{|\mathcal{C}G[A]|}(-1)^{|A|}, \tag{2.3}$$

where $\hat{B}$ denotes the set of edges $\subseteq E(G')$ that connect the vertices in B to v'. We can prove a stronger statement than (2.3): for every subset A of the edges of G, we have

$$\sum_{B \subseteq V(G)} q^{|\mathcal{C}G'[A \cup \hat{B}]|}(-1)^{|B|} = q(q-1)^{|\mathcal{C}G[A]|}. \tag{2.4}$$

This is the statement of the following lemma if we set $x_e = -1$ for every edge e connecting a vertex in V to v'.

Lemma 2.6. *Let $G = (V, E)$ be a graph and G' be obtained from G by adding a new vertex v' and connecting vertex v' to all vertices in V. Let $\mathbf{x}$ be an $E(G')$-indexed variable. Then, for every $A \subseteq E$, we have*

$$\sum_{B \subseteq V} q^{|\mathcal{C}G[A \cup \hat{B}]|} x_{\hat{B}} = q \prod_{C \in \mathcal{C}G[A]} (q - 1 + \prod_{u \in V(C)} (1 + x_{\hat{u}})), \tag{2.5}$$

where $\hat{u}$ denotes the edge connecting $u \in V$ and v'.

Proof. We write every $B \subseteq V$ as disjoint union of $B_1, \ldots, B_\ell$, where each B_i contains the vertices of B that belong to the same connected component of $G[A]$:

$$\begin{aligned}
&\sum_{B \subseteq V} q^{|\mathcal{C}G[A \cup \hat{B}]|} x_{\hat{B}} \\
=&\sum_{\substack{\emptyset \subseteq \mathcal{C} \subseteq \mathcal{C}G[A] \\ \mathcal{C}=\{C_1,\ldots,C_\ell\}}} \sum_{\emptyset \subsetneq B_1 \subseteq V(C_1)} \cdots \sum_{\emptyset \subsetneq B_\ell \subseteq V(C_\ell)} q^{|\mathcal{C}G[A \cup \hat{B}_1 \cup \ldots \cup \hat{B}_\ell]|} x_{\hat{B}_1 \cup \ldots \cup \hat{B}_\ell} \\
=&\sum_{\{C_1,\ldots,C_\ell\}} q^{|\mathcal{C}G[A]|-\ell+1} \sum_{\emptyset \subsetneq B_1 \subseteq V(C_1)} x_{\hat{B}_1} \cdots \sum_{\emptyset \subsetneq B_\ell \subseteq V(C_\ell)} x_{\hat{B}_\ell} \\
=&q \sum_{\{C_1,\ldots,C_\ell\}} q^{|\mathcal{C}G[A]|-\ell} \prod_{1 \leq i \leq \ell} \Big(\big(\prod_{u \in V(C_i)} 1 + x_{\hat{u}} \big) - 1 \Big) \\
=&q \prod_{C \in \mathcal{C}G[A]} \Big(q + \big(\big(\prod_{u \in V(C)} 1 + x_{\hat{u}} \big) - 1 \big) \Big).
\end{aligned}$$

□

Corollary 2.7. *Let G be a graph and $G^{(k)}$ obtained from G by performing k times the following operation: Add a new vertex $v^{(i)}$ and connect all old vertices to $v^{(i)}$. Let $\mathbf{x}$ be a labeling of the edges of $G^{(k)}$ with the following property: For all edges e in $E(G^{(k)}) \setminus E(G)$, we have $x_e = -1$. For all edges $e \in E(G)$, the labels x_e are pairwise independent indeterminates. Then we have*

$$Z(G^{(k)}; q, \mathbf{x}) = q^{\underline{k}} Z(G; q - k, \mathbf{x}).$$

Proof. Let G be obtained from G by adding v' and connecting v' to all vertices of G. As the weights of the new edges are -1, we obtain from Lemma 2.6

$$Z(G'; q, \mathbf{x}) = qZ(G; q - 1, \mathbf{x}).$$

We apply this step k times to obtain the statement of the corollary. □

Proof of Theorem 2.4. Let $q \in \mathbb{C} \setminus \{0, 1\}$ and $n \in \mathbb{N}$. We describe how to construct an arithmetic circuit computing $Z(K_n; 2, \mathbf{x})$ given oracle access to a circuit computing $Z(K_{2n}; q, \mathbf{x})$. This proves the theorem.

For the first case, let us assume that q is an integer and $n > q - 2$. Then, by Corollary 2.7, we can compute $Z(K_n; 2, \mathbf{x})$ from $Z((K_n)^{(q-2)}; q, \mathbf{x})$. As $n > q - 2$, $(K_n)^{(q-2)}$ has less than $2n$ vertices. Thus, $Z((K_n)^{(q-2)}; q, \mathbf{x})$ can be computed from $Z(K_{2n}; q, \mathbf{x}')$ where we set $x'_e = 0$ for all edges in K_{2n} that are not edges of $(K_n)^{(q-1)}$ and $x'_e = x_e$ otherwise.

Let us the discuss the second case: q is not an integer or $n \leq q - 2$. We we consider $P := Z(K_n; y, \mathbf{x})$ as polynomial in y. Its degree is at most n. Once we have circuits for $p_0 := Z(K_n; q, \mathbf{x})$, $p_1 := Z(K_n; q-1, \mathbf{x})$, …, $p_n := Z(K_n; q-n, \mathbf{x})$, we can build a polynomial size (in n) circuit that interpolates the coefficients of P. From this we can build a polynomial size (in n) circuit computing $Z(K_n; y, \mathbf{x})$, where y is an indeterminate. This finishes the reduction as we can substitute $y = 2$.

We can build a circuit for $p_1, \ldots, p_n$ as follows. As $n \leq q-2$ or q is not an integer, $q^{\underline{n}}$ is not zero and we can use Corollary 2.7 to compute $p_1, \ldots, p_n$ by computing $Z((K_n)^{(i)}; q, \mathbf{x})$ for $1 \leq i \leq n$. As $(K_n)^{(i)}$, $1 \leq i \leq n$, has at most $2n$ vertices, we can compute $Z((K_n)^{(i)}; q, \mathbf{x})$ from a circuit computing $Z(K_{2n}; q, \mathbf{x}')$ by setting some of the x'_e to zero. □

Let us close this section with the following comparison: Regarding #P-hardness, 2 is an "easy" point for the chromatic polynomial as $\chi(-; 2)$ is polynomial time computable. But regarding VNP-hardness of specializations of the Tutte polynomial, $q = 2$ is a hard point. As we have seen, this is the only point $q \in \mathbb{Q}$ where the situation of $\chi(-; q)$ regarding #P-hardness is not the same as the situation of $(Z(K_n; q, \mathbf{x}))$ regarding VNP-hardness.

2.3 Open Problems

An open problem that arose—and, in fact, has been posed before [LM04, Problem 1]—is: Do the hardness results for the Tutte polynomial also hold with respect to p-projections?

3 Graph Transformations and Reductions

The aim of this chapter it to provide graph transformations that enable point-to-point reductions for graph polynomials. Basically, such a reduction is just an equation

$$P(G;\xi') = p(G) \cdot P(G';\xi),$$

where G' is the graph obtained from G via the graph transformation and $p(G)$ is an easy to compute prefactor. This reduces $P(-;\xi')$ to $P(-;\xi)$.

The first section of this chapter presents a powerful technique that yields hardness results for many graph polynomials. We introduce the notions of vertex/edge induced subgraph polynomials (VSPs/ESPs) and of vertex/edge cloning. Then we focus on situations in which cloning is "supported", i.e. fulfills a simple condition. We show that this immediately leads to a point-to-point reduction. Our main result is then that, if cloning is supported, computing the coefficients of the VSP/ESP reduces to evaluation of the VSP/ESP at almost any point. This is a crucial ingredient in the hardness proofs for the independent set, interlace, Tutte, and extended bivariate chromatic polynomial.

In Section 3.2, we present more graph transformations (adding "combs", cycles, or paths) that enable point-to-point reductions for the interlace polynomial. They will be used in Chapters 4 and 5 to further investigate the computational complexity of the interlace polynomial .

All these graph transformations and the associated proofs of a point-to-point reduction follow a general scheme. In Section 3.3, we point out what conditions have to be met for this scheme to work. We demonstrate this scheme with a vertex cover and a domination polynomial and discuss limits of the method.

3.1 Subgraph Induced Graph Polynomials, Clones, and Interpolation

As Johann A. Makowsky observes, many graph polynomials are or can be defined as graph polynomials "counting the number of (induced) subgraphs of a certain kind" [Mak08, Page 545] or, slightly more general, sum over certain substructures of a graph and weight each substructure according to its size. This observation is one motivation for our definition of vertex/edge induced subgraph polynomials (Definition 3.1).

Alan D. Sokal has pointed out the advantages of using "multivariate weights" [Sok04, Sok05, SS05] in this setting. In fact, individual weights for vertices and

edges have been used in physical models for a long time [FK72] and are also important in knot theory. The following feature of individual weights is very useful in many proofs of point-to-point reductions for graph polynomials: During the computation of a graph polynomial, a part of the graph may be replaced by a single vertex or edge if the weight of this vertex/edge is set to an appropriate value. This is the same concept as in the well-known formulas for the equivalent resistance for multiple resistances connected in parallel or series in electrical networks. We incorporate multivariate weights in our definition of vertex/edge induced subgraph polynomials.

This section is centered around only one particular graph transformation: cloning of vertices or edges. Edge cloning is mainly known as edge thickening or edge fattening. It means that every edge of a graph is replaced by k parallel copies of the edge. Thickening has been used to to find reductions for the Tutte polynomial [JVW90, Sok05, BM06, BDM10], counting problems for graph homomorphisms [DG00], and the extended bivariate chromatic polynomial [Hof10]. Note that edge cloning is, in some respect, the inverse of parallel reduction for resistances in electrical networks.

Vertex cloning replaces every vertex v of a graph by k vertices that have exactly the same set of neighbors as v. This technique has been employed for a long time, for example by Valiant [Val79b, Theorem 1, Reduction 3.], Jerrum et al. [JVV86], Roth [Rot96, Lemma A.3], and Bläser and Hoffmann [BH08].

A main reason to apply cloning is to get a family of point-to-point reductions that finally enables interpolation. Most of the times, it is this very same strategy, which one just has to follow almost blindly, and which eventually gives the desired interpolation result. This is particularly true for several graph polynomials and the associated proofs that they are #P-hard to evaluate almost everywhere. It turns out that these graph polynomials are all vertex/edge induced subgraph polynomials and that it is just one simple precondition ((3.1) in Definition 3.12) that ensures that the whole procedure works.

Definition 3.1. *Let f be a graph polynomial and $\mathbf{x}$ be a vertex indexed variable that does not appear in f. The vertex induced subgraph polynomial (VSP) of f,* $\mathrm{VSP}\, f := \mathrm{VSP}_{\mathbf{x}} f$*, is the following function:*

$$(\mathrm{VSP}_{\mathbf{x}} f)(G; \mathbf{x}) = \sum_{A \subseteq V(G)} x_A f(G[A]), \quad x_A = \prod_{a \in A} x_a.$$

The edge induced subgraph polynomial (ESP) of f, $\mathrm{ESP}\, f := \mathrm{ESP}_{\mathbf{x}} f$*, is defined analogously if $\mathbf{x}$ is an edge indexed variable:*

$$(\mathrm{ESP}_{\mathbf{x}} f)(G; \mathbf{x}) = \sum_{A \subseteq E(G)} x_A f(G[A]), \quad x_A = \prod_{a \in A} x_a.$$

We call $\mathbf{x}$ the weight variable *of* $\mathrm{VSP}_{\mathbf{x}} f$ *(*$\mathrm{ESP}_{\mathbf{x}} f$*, resp.).*

Note that f itself can depend on a set of indeterminates, cf. Examples 3.4 and 3.5 below. In this case, of course, the VSP (or ESP) of f depends on these indeterminates, too. We do not mention them explicitly as they do not play a significant role in the theory we are going to develop and carrying them along all the time would distract the reader.

Remark 3.2. *Every VSP and every ESP is a graph polynomial.*

Proof. We prove the statement for ESPs, VSPs are analogous. Let $\varphi : G_1 \to G_2$ be a graph isomorphism, $G_i = (V_i, E_i)$ for $i = 1, 2$, $\mathbf{x}$ an edge-indexed variable, and f a graph polynomial such that $\mathbf{x}$ does not appear in f. We have

$$\begin{aligned}\operatorname{ESP} f(G_1)_\varphi &= \sum_{A \subseteq E_1} x_{\varphi(A)} f(G_1[A])_\varphi \\ &= \sum_{A \subseteq E_1} x_{\varphi(A)} f(G_2[\varphi(A)]) \\ &= \sum_{B \subseteq E_2} x_B f(G_2[B]) = \operatorname{ESP} f(G_2).\end{aligned}$$

The second equality holds as f is a graph polynomial and φ induces an isomorphism from $G_1[A]$ onto $G_2[\varphi(A)]$. □

Example 3.3. *For a graph G, let $i(G) = 1$ if G has no edges and $i(G) = 0$ otherwise. The multivariate independent set polynomial I,* (1.3)*, is the VSP of i:*

$$I(G; \mathbf{x}) = \operatorname{VSP} i(G; \mathbf{x}).$$

Example 3.4. *For a graph G, let $U(G; u) = u^{\operatorname{rk} G}$, where u is a variable and $\operatorname{rk} G$ is the $GF(2)$-rank of the adjacency matrix of G. The multivariate interlace polynomial $\bar{q}$ (Definition 1.16) is the VSP of U:*

$$\bar{q}(G; u, \mathbf{x}) = \operatorname{VSP} U(G; u, \mathbf{x}).$$

Example 3.5. *For a graph G, let $Q(G; q) = q^{k(G)}$, where q is a variable and $k(G)$ is the number of connected components of G. The multivariate Tutte polynomial* (1.4) *is the ESP of Q:*

$$Z(G; q, \mathbf{x}) = \operatorname{ESP} Q(G; q, \mathbf{x}).$$

Example 3.6.

1. *For a graph G, let $m(G) = 1$ if G is a matching and 0 otherwise. The multivariate matching polynomial* (1.10) *is the ESP of m:*

$$M(G; \mathbf{x}) = \operatorname{ESP} m(G; \mathbf{x}).$$

2. *Let us define an* mmatching *(modified matching) to be a set of edges such that the end vertices of every two edges are either disjoint or equal. Let $m'(G) = 1$ if G is an mmatching and 0 otherwise. Then*

$$M'(G; \mathbf{x}) = \operatorname{ESP} m'(G; \mathbf{x}) = \sum_{A \subseteq E(G)} x_A,$$

where the sum is over all mmatchings, is a graph polynomial that coincides with the multivariate matching polynomial on simple graphs.

Remark 3.7. *The product $G \mapsto f(G) \cdot g(G)$ of two graph polynomials $f, g : \mathcal{G} \to R$ is a graph polynomial, too.*

Example 3.8. *A slightly modified version of the extended bivariate chromatic polynomial, ψ in Section 4.3.1, is an ESP (Definition 4.13, Proposition 4.16, Lemma 4.17).*

Definition 3.9 (Cloning one vertex/edge)**.** *Let $G = (V, E)$ be a graph.*

1. *Let $v \in V$ be a vertex of G. The* vertex v clone *of G, $\kappa_v G$, is obtained from G by the following procedure:*
   a) *Start with $\kappa_v G = G$.*
   b) *Add a new vertex v' to $\kappa_v G$.*
   c) *For every edge in G that connects v and some vertex u, add a new edge to $\kappa_v G$ that connects v' and u. (Note that, in this way, multiple edges from v to some vertex u in G enforce multiple edges from v' to u in $\kappa_v G$. Furthermore, as we have not excluded $u = v$, self loops at v enforce edges between v' and v in $\kappa_v G$.)*
   d) *Let k be the number of self loops at v in G. Add k self loops at v' to $\kappa_v G$.*
2. *Let $e \in E$ be an edge of G. The* edge e clone *of G, $\kappa_e G$, is obtained from G by adding a new edge e' that connects the same vertices as e.*

Remark 3.10. *Cloning is commutative, i.e. $\kappa_a \kappa_b = \kappa_b \kappa_a$ for every two vertices (edges) a and b.*

Proof. For vertex cloning, consider a graph $G = (V, E)$ and let a' and b' be the clones of a and b, resp. It is easy to see that in both, $G_1 = \kappa_a \kappa_b G$ and $G_2 = \kappa_b \kappa_a G$, the following holds:

- Vertices a and a' are connected to all vertices in $N_G(a) \setminus \{b\}$. The multiplicity of the respective edges equals the multiplicity of the corresponding edges in G. A similar statement holds for b and b'.
- The number of edges between a and a' equals the number of self loops at a in G, similarly for b and b'.
- There are (multi)edges $ab, ab', a'b, a'b'$ if ab is an edge in G (with the respective multiplicity). Otherwise, there is no edge between $\{a, a'\}$ and $\{b, b'\}$.

For edge cloning, the statement is obvious. □

Remark 3.10 ensures that the following is well-defined.

Definition 3.11 (Cloning multiple vertices/edges)**.** *Let $G = (V, E)$ be a graph. We write $\kappa_A G$ for $\kappa_{a_r} \dots \kappa_{a_1} G$, where $A = \{a_1, \dots, a_r\}$ is a set of vertices (edges). We write $\kappa_V^k G$ ($\kappa_E^k G$) for*

$$\underbrace{\kappa_A \dots \kappa_A}_{k \text{ applications of } \kappa_A} G,$$

where $A = V$ ($A = E$, resp.).

Now we come to the most crucial definition of this section. It is just the following condition that ensures that cloning yields a powerful point-to-point reduction (Theorem 3.15), which leads to an interpolation result (Theorem 3.19).

Definition 3.12. *Let f be a graph polynomial. We say that f* supports vertex (edge) cloning*, if for every graph G and every vertex (edge) a of G*

$$f(\kappa_a G) = f(G). \tag{3.1}$$

Remark 3.13. *If two graph polynomials support vertex (edge) cloning, their product also supports vertex (edge, resp.) cloning.*

Example 3.14.

1. *i from example 3.3 supports vertex cloning. This is easy to see.*
2. *$G \mapsto u^{\operatorname{rk} G}$ supports vertex cloning. This will be proven in Lemma 3.20.*
3. *$G \mapsto q^{k(G)}$ supports edge cloning. This will be proven in Proposition 4.14.*
4. *Recall the definition of a mmatching from Example 3.6. The function*

   $$m' : G \mapsto \text{the edges of } G \text{ form a mmatching}$$

   supports edge cloning. This is easy to see.
5. *The function h defined by (4.6), $x^{k(G)}$, and $h \cdot x^{k(G)}$ from Definition 4.13 support edge cloning. This will be proven in Proposition 4.15.*

To recognize the full power of the following theorem, please recall that, as f may depend on indeterminates that are not mentioned explicitly, the theorem can be applied to graph polynomials that have more variables than just x.

Theorem 3.15. *Let f be a graph polynomial that supports vertex (edge) cloning and let $F = \operatorname{VSP} f$ ($\operatorname{ESP} f$, resp). Let $G = (V, E)$ be a graph and $A = V$ ($A = E$, resp.). Then*

$$F(\kappa_A^k G; x) = F(G; (1 + x)^{k+1} - 1). \tag{3.2}$$

Proof. We prove the statement for edge cloning. Vertex cloning is completely analogous.

Let e be any edge of G, $E' = E \setminus \{e\}$, and e' the clone of e in $\kappa_e G$ (cf. Definition 3.9).

$$\begin{aligned}
\operatorname{ESP} f(\kappa_e G; \mathbf{x}) &= \sum_{A \subseteq E(\kappa_e G)} x_A f((\kappa_e G)[A]) \\
&= \sum_{A \subseteq E'} x_A \Big(f((\kappa_e G)[A]) + x_e f((\kappa_e G)[A \cup \{e\}]) \\
&\qquad + x_{e'} f((\kappa_e G)[A \cup \{e'\}]) + x_e x_{e'} f((\kappa_e G)[A \cup \{e, e'\}]) \Big) \\
&= \sum_{A \subseteq E'} x_A \Big(f(G[A]) + x_e f(G[A \cup \{e\}]) + x_{e'} f(G[A \cup \{e\}]) \\
&\qquad + x_e x_{e'} f(G[A \cup \{e\}]) \Big).
\end{aligned}$$

The last equality is by the fact that $(\kappa_e G)[A] = G[A]$, $(\kappa_e G)[A \cup \{e\}] = G[A \cup \{e\}]$, $(\kappa_e G)[A \cup \{e'\}]$ and $G[A \cup \{e\}]$ are isomorphic, and f supports cloning.

We have just proven

$$\operatorname{ESP} f(\kappa_e G; \mathbf{x}) = \operatorname{ESP} f(G; \mathbf{X}) \tag{3.3}$$

with $X_a = x_a$ for $a \in E'$ and $X_e = x_e + x_{e'} + x_e x_{e'} = (1 + x_e)(1 + x_{e'}) - 1$. Thus, cloning every edge k times yields the statement of the theorem. □

Remark 3.16. *If we replace* (3.1) *by*

$$f(\kappa_a G) = 0,$$

we can prove a similar statement, namely $F(\kappa_A^k G; x) = F(G; (k+1)x)$. *This is the case for the matching polynomial.*

Now we define a formalism for "a family of graph transformations that yield point-to-point reductions which enable interpolation".

Definition 3.17. *Let* $\mathbb{F}$ *be a Turing representable field (Definition 1.20 on Page 29),* $x \notin \mathbb{F}$ *an indeterminate, and* $P : \mathcal{G} \to \mathbb{F}[x]$ *a graph polynomial. We say that* P *has an* interpolating family *of graph transformations at* $\xi \in \mathbb{F}$ *if the following holds: There exists a polynomial time algorithm that on input* (k, G), k *being a positive integer and* G *a graph, constructs graphs* $G_1, \ldots, G_k$ *and pairwise distinct* $\xi_1, \ldots, \xi_k \in \mathbb{F}$ *such that for every* i, $1 \leq i \leq k$,

1. $|G_i|$ *is bounded by a function* $s(|G|, i)$ *that is monotonously increasing in* i *and a polynomial in both parameters,*
2. $\|\xi_i\|$ *is bounded by a polynomial in* $\|\xi_0\|$ *and* $|G|$, *and*

3. $P(G_i;\xi_0) = P(G;\xi_i)$.

The constraint that s is monotone in i is not essential but only for convenient formulation of Theorem 3.18.

Theorem 3.18. *Let $\mathbb{F}$ be a Turing representable field, $x \notin \mathbb{F}$ an indeterminate, and $P : \mathcal{G} \to \mathbb{F}[x]$ a graph polynomial that has an interpolating family at $\xi_0 \in \mathbb{F}$. Let the x-degree of $P(G;x)$ be bounded by a polynomial function $d(|G|)$. Then $P(-;x)$ is Turing reducible to $P(-;\xi_0)$. The reduction has the following properties:*

1. *the size of the input instances for the oracle are bounded by $s(|G|, d(|G|))$,*
2. *besides the oracle calls, the reduction works in time polynomial in $|G|$ and $\|\xi_0\|$.*

In particular, the reduction is a polynomial time Turing reduction, i.e. $P(-;x) \preceq_T P(-;\xi_0)$.

Let us remark that the same result holds for a more general notion of interpolating families:

- Instead of 3. in Definition 3.17 it would be sufficient to require only $P(G_i;\xi_0) = p \cdot P(G;\xi_i)$ where $p \neq 0$ is polynomial time computable. In fact, we will encounter such equations in Theorem 3.24, 3.27, and 3.31. But we will use them for interpolation only once (Theorem 5.3). Thus, there is no need to generalize (and, thus, complicate) our definition of interpolating families.
- We could use more than one transformed graph for every evaluation point: It might be possible to compute $P(G;\xi_i)$ using $P(G_i^{(1)};\xi_0)$ and $P(G_i^{(2)};\xi_0)$ for two transformed graphs $G_i^{(1)}$ and $G_i^{(2)}$. We will do this in Proposition 4.21, using Lemma 4.20. But again, we do this only once in this work.

Proof of Theorem 3.18. Construct in time $\mathsf{poly}(|G|)$ the graphs $G_1, \ldots, G_{d(|G|)+1}$ as in the definition of interpolating families (Definition 3.17). Evaluate $P(G;\xi_0)$, $P(G_1;\xi_0), \ldots, P(G_{d(|G|)};\xi_0)$. This yields $P(G;\xi_1), \ldots, P(G;\xi_{d(|G|)})$, giving us the value of $P(G;x)$ at at least $\deg P(G;x) + 1$ distinct points. Using Lagrange interpolation, we interpolate the coefficients of $P(G;x)$. Let us argue that the numbers occurring during the interpolation are not too large. By the definition of interpolating family, every $|\xi_i|$ is bounded by $|\xi_0|$ to the power of some polynomial in $|G|$. Thus, as $|P(G;\xi_i)|$ basically is bounded by $|\xi_i|$ to the power of $d(|G|)$, $|P(G;\xi_i)|$ is bounded by $|\xi_0|$ to $\mathsf{poly}(|G|) \cdot d(|G|)$. Thus, the size of the encoding of $|P(G;\xi_i)|$ is polynomial in $|G|$ and $\|\xi_0\|$. □

Let us now state the main result of this section. Recall that a Turing representable field can contain indeterminates. For instance, we can choose $\mathbb{F} = \mathbb{Q}(q)$ to apply the following theorem to the Tutte polynomial $Z(G;q,x)$.

Theorem 3.19. *Let $\mathbb{F}$ be a Turing representable field, $x \notin \mathbb{F}$ an indeterminate, $f : \mathcal{G} \to \mathbb{F}$ a graph polynomial that supports vertex cloning, and $F = \mathrm{VSP}\, f$. Then, for every $\xi_0, \xi_1 \in \mathbb{C} \cap \mathbb{F}$, $|1 + \xi_0| \notin \{0, 1\}$, we have*

- $F(-; x) \preceq_T F(-; \xi_0)$ *and*
- $F(-; \xi_1) \preceq_T F(-; \xi_0)$.

A completely analogous statement holds for edge cloning and ESPs.

Proof. We prove the theorem for VSPs, ESPs are completely analogous.

The following procedure shows that $F = \mathrm{VSP}\, f$ has an interpolating family: On input (G, k) construct $G_i = \kappa_V^i G$ and $\xi_i = (1 + \xi)^i - 1$ for all $i = 1, \ldots, k$. We have $|G_k| = k|G|$ and $\|\xi_i\| = O(i\|\xi\|)$. The ξ_i are pairwise distinct as $|\xi_0 + 1| \notin \{0, 1\}$. By Theorem 3.15, we have $F(G_k; \xi) = F(G; \xi_k)$. The first statement of the theorem now follows from Theorem 3.18. For the second statement, we use the first statement to obtain the reduction $F(-; \xi_1) \preceq^{\mathtt{m}} F(-; x) \preceq_T F(-; \xi_0)$. □

3.1.1 Examples

The machinery we just have defined can be applied to several important graph polynomials.

Independent Set Polynomial

Theorem 3.19, Example 3.3, and 1. of Example 3.14 yield that evaluating the independent set polynomial is #P-hard at almost every point. This is almost the entire statement of Corollary 4.8. The underlying point-to-point reduction follows from Theorem 3.15:

$$I(\kappa_{V(G)}^k G; x) = I(G; (1 + x)^{k+1} - 1). \tag{3.4}$$

Interlace Polynomial

Proposition 3.22, which is the main ingredient for our #P-hardness results on the computational complexity of evaluating the interlace polynomial $\bar{q}$, follows from Theorem 3.19, Example 3.4, and 2. of Example 3.14.

Tutte Polynomial

Let us consider the computational complexity of evaluating the Tutte polynomial. Jaeger, Vertigan, and Welsh showed [JVW90] that at every point of the plane, except for a few exceptional points and on one hyperbola, evaluating the Tutte polynomial is #P-hard (Theorem 1.38). In principle, this result can be deduced from the following two ingredients:

1. Linial's construction [Lin86] (see Lemma 2.5), gives a reduction along the q axis:
$$Z(G'; q, -1) = qZ(G; q - 1, -1).$$

2. Theorem 3.19, Example 3.5, and 3. of Example 3.14 give, for every q, a reduction along the x axis:

$$Z(\kappa_E^{k-1}G;q,x) = Z(G;q,(1+x)^k-1) \quad \text{for graph } G=(V,E).$$

Extended Bivariate Chromatic Polynomial

The extended bivariate chromatic polynomial allows for two different point-to-point reductions that both can be obtained via cloning. The first is via edge cloning and will be given in Theorem 4.18:

$$\xi(\kappa_{E(G)}^{k-1}G;x,y,z) = \xi\Big(G;x,(1+y)^k-1,z\frac{(1+y)^k-1}{y}\Big). \tag{3.5}$$

Writing ξ as an ESP is clearly inspired by (1.11). Then (3.5) follows from Theorem 3.15 and 5. of Example 3.14. It is the main reduction in Theorem 4.22, which uses Theorem 3.19.

The second cloning based reduction for ξ uses vertex cloning and is (3.4), the reduction for the independent set polynomial. Using (1.9) and (1.15), we can write it in terms of ξ:

$$\begin{aligned} &x^{k|V|}\xi\Big(\kappa_{V(G)}^{k-1}G;\frac{1}{x}+1,-1,\frac{1}{x}\Big)\\ =&x^{|V|}\xi\Big(G;\frac{1}{(x+1)^k-1}+1,-1,\frac{1}{(x+1)^k-1}\Big). \end{aligned} \tag{3.6}$$

Even though this is obtained by vertex cloning, I do not see how ξ could be written as a VSP of some polynomial that supports vertex cloning. The independent set polynomial can, and we could generalize the bivariate chromatic polynomial to a VSP as

$$P(G;\mathbf{x},y) = \sum_{\bar{A}\subseteq V} x_{\bar{A}}P(G[A];y).$$

To further extend this to some variant of ξ, we would need a multivariate version of the recursion that establishes the connection between the bivariate chromatic polynomial and the extended bivariate chromatic polynomial, i.e.[1]

$$P(G;x,y) = P(G\setminus e;x,y) - P(G/e;x,y) + (x-y)P(G\dagger e;x,y).$$

But this seems to be impossible as there is only one term "$x-y$", while we could have a different x_a for different vertices $a\in V$.

Thus, cloning yields two different point-to-point reductions for the extended bivariate chromatic polynomial ξ, (3.5) and (3.6). But for the proof of one of these, we have to consider a polynomial (the multivariate independent set polynomial) that is not contained in ξ.

[1] Cf. (1.12) and (1.15).

Matchings

Let us consider algorithms that count mmatchings (see Example 3.6) in graphs, i.e. in graphs that may have multiple edges and self loops. For such algorithms, Theorem 3.19, Example 3.6 and 4. of Example 3.14 immediately yield: For almost all weights $x \in \mathbb{Q}$, weighted counting of mmatchings, i.e. $M'(-;x)$, is as hard as counting mmatchings, i.e. $M'(-;1)$. Counting mmatchings, in turn, is #P-hard as it implies counting ordinary matchings in simple graphs, which is known to be #P-hard [Val79b].

That we used mmatchings is not a severe restriction: If we consider algorithms that count ordinary matchings, but still in graphs that may have multiple edges, we come to the same conclusion using Remark 3.16.

For algorithms that work on simple graphs only, it seems that our method does not suffice.[2]

3.2 Graph Transformations for the Interlace Polynomial

In this section, we devise graph transformations for the interlace polynomial. First, we show that vertex cloning is supported by the graph polynomial that generates the interlace polynomial as a VSP. Then we provide additional graph transformations (adding "combs", cycles, or paths), which are very useful for the interlace polynomial. The main results of this section are the point-to-point reductions: Theorem 3.21, Theorem 3.24, Theorem 3.27, and Theorem 3.31 describe the effect of these graph transformations on the interlace polynomial.

3.2.1 Clones

The simplest graph transformation for the interlace polynomial is vertex cloning. It arises naturally if one thinks of a way to extend a graph such that the rank of the adjacency matrix (of all vertex induced subgraphs) does not change, see (3.7).

Lemma 3.20. *$G \mapsto u^{\operatorname{rk} G}$ supports vertex cloning.*

Proof. Let $G = (V, E)$ be a graph. Let $a \in V$ be any vertex (the one which will be cloned) and N the set of neighbors of a, and $M = (V \setminus \{a\}) \setminus N$. The adjacency

[2] Averbouch and Makowsky consider the complexity of multivariate matching polynomials [AM07]. In the proof of their Theorem 2, they give a reduction from $EVAL(A)$ to $EVAL(A, D)$, which is based on a construction of Dyer and Greenhill [DG00, Theorem 3.2]. Unfortunately, this construction does not preserve the property of being a line graph, which is needed for the final step from $EVAL(A, D)$ to the matching polynomial [AM07, Proposition 8]. For instance, performing p-thickening, $p \geq 3$, on a single edge and then a 2-stretch on every edge yields a graph that is not a line graph. Thus, to fully establish the hardness result for the matching polynomial on *simple* graphs [AM07, Theorem 2], there is more work to be done.

matrix B of G and the adjacency matrix B_{aa} of $\kappa_a G$ are

$$B = \left(\begin{array}{c|ccc} & a & N & M \\ \hline a & b & \mathbf{1} & \mathbf{0} \\ N & \mathbf{1} & A_{11} & A_{12} \\ M & \mathbf{0} & A_{21} & A_{22} \end{array}\right) \quad \text{and}$$
$$B_{aa} = \left(\begin{array}{c|cccc} & a' & a & N & M \\ \hline a' & b & b & \mathbf{1} & \mathbf{0} \\ a & b & b & \mathbf{1} & \mathbf{0} \\ N & \mathbf{1} & \mathbf{1} & A_{11} & A_{12} \\ M & \mathbf{0} & \mathbf{0} & A_{21} & A_{22} \end{array}\right), \tag{3.7}$$

where $b = 1$ if a has a self loop and $b = 0$ otherwise. As the first column of B_{aa} equals its second column, as well as the first row equals the second row, we can remove the first row and the first column of B_{aa} without changing the rank. Thus, $u^{\operatorname{rk} G} = u^{\operatorname{rk} \kappa_a G}$. □

Theorem 3.21 ([BH08, Theorem 3.3])**.** *Let G be a graph and G_k be obtained out of G by cloning each vertex of G exactly $k-1$ times. Then*

$$P(G_k; u, x) = P(G; u, (1+x)^k - 1). \tag{3.8}$$

Proof. By Example 3.4, Lemma 3.20 and Theorem 3.15. □

As we will use it in the proof of Theorem 4.9, we note the following identity for q, which can be derived from Theorem 3.21 using Lemma 1.17:

$$q(G_k; x, y) = q(G; (x-1)\frac{y^k - 1}{y - 1} + 1, y^k). \tag{3.9}$$

Theorem 3.21 also implies the following reduction for the interlace polynomial, which is the foundation for our results in Section 4.1.

Proposition 3.22 ([BH08, Proposition 3.4])**.** *Let $B_2 = \{0, -1, -2\}$ and x be an indeterminate. For $\mu \in \widetilde{\mathbb{Q}}, \xi \in \widetilde{\mathbb{Q}} \setminus B_2$ we have $\bar{q}(-; \mu, x) \preceq_T \bar{q}(-; \mu, \xi)$.*

Proof. By Example 3.4, Lemma 3.20, and Theorem 3.19. □

3.2.2 Combs

Some points, such as $\xi_0 = -1$, do not behave nice in the sense that they do not fulfill the prerequisite of Theorem 3.19. This means that cloning does not help to recognize the power of evaluation at such points. Using the comb transformation, we obtain a reduction $\bar{q}(-; \mu, \xi_0') \preceq^{\mathtt{m}} \bar{q}(-; \mu, -1)$ for some "nice" point point ξ_0'. This shows that $\bar{q}(-; \mu, -1)$ is as powerful as evaluation at the "nice" points (see the proof of Proposition 4.5).

The comb transformation complements cloning in another way, too: Assume that we can evaluate the interlace polynomial at some point ξ. If $|1+\xi| > 1$, cloning allows us to evaluate it also at very large points, as $|1+\xi|^k$ quickly tends to infinity. If, however, $|1+\xi| < 1$, this does not hold anymore. But in this case, combing yields a sequence of evaluation points that quickly tends to infinity. We will use this in the proof of Lemma 4.11 to show inapproximability of the independent set polynomial at almost every point. Another application is in Corollary 5.4.

Definition 3.23 (Comb)**.** *Let $G = (V, E)$ be a graph and $a \in V$ some vertex. Then we define the* k-comb *of G at a as $G_{a,k} = (V \cup \{a_1, \ldots, a_k\}, E \cup \{\{a, a_1\}, \ldots, \{a, a_k\}\})$, with $a_1, \ldots, a_k$ being new vertices.*

The general technical argument of the following theorem is very similar to, but slightly more general than, the one in the proof of Theorem 3.15. In Section 3.3, we will formulate it in an abstract way, see Theorems 3.45 and 3.49.

Theorem 3.24 ([BH08, Theorem 3.5])**.** *Let G be a graph and G_k be obtained out of G by performing a k-comb operation at every vertex. Then*

$$\bar{q}(G_k; u, x) = p(k, u, x)^{|V(G)|} \bar{q}(G; u, x/p(k, u, x)), \tag{3.10}$$

where $p(k, u, x) = (1+x)^k(xu^2+1) - xu^2$.

Proof. The adjacency matrices of the original graph G (the graph $G_{a,k}$ with a k-comb at a, resp.) are

$$\left(\begin{array}{c|cc} & a & V' \\ \hline a & b & \mathbf{c} \\ V' & \mathbf{c}^T & A \end{array}\right) \quad \text{and} \quad \left(\begin{array}{c|ccccccc} & a_1 & a_2 & \ldots & a_k & a & V' \\ \hline a_1 & & & & & 1 & \\ a_2 & & & & & 1 & \\ \vdots & & & & & \vdots & \\ a_k & & & & & 1 & \\ a & 1 & 1 & \ldots & 1 & b & \mathbf{c} \\ V' & & & & & \mathbf{c}^T & A_{11} \end{array}\right), \text{ resp.,} \tag{3.11}$$

with empty entries being zero. Consider $A \subseteq V(G_{a,k})$. Let $M := A \cap \{a, a_1, \ldots, a_k\}$. By (3.11), the rank of $G_{a,k}$ is related to the rank for G in the following way:

- If $a \notin M$, then $\mathrm{rk}(G_{a,k}[A]) = \mathrm{rk}(G[A \setminus M])$.
- If $a \in M$ and $M \cap \{a_1, \ldots, a_k\} \neq \emptyset$, then $\mathrm{rk}(G_{a,k}[A]) = \mathrm{rk}(G[A \setminus M]) + 2$: Let w.l.o.g. $a_1 \in M$. Consider the adjacency matrix of $G_{a,k}[A]$ and the following operations on it, which leave the rank unchanged. Using the first column we remove all 1s in the a-row, except the 1 in the first column. Using the first row we remove all 1s in the a-column, except the 1 in the first row. The resulting matrix B is a $(k+|V|) \times (k+|V|)$ matrix with 1s at positions (a, a_1) and (a_1, a), the submatrix of A_{11} induced by $A \setminus M$ in the lower right corner and zeros everywhere else. Thus $\mathrm{rk}(B) = \mathrm{rk}(G[A \setminus M]) + 2$.

- If $M = \{a\}$, then $\mathrm{rk}(G_{a,k}[A]) = \mathrm{rk}(G[A])$.

Letting $r(A) := \mathrm{rk}(G[A])$ and $r_a(A) := \mathrm{rk}(G[A \cup \{a\}])$ for $A \subseteq V'$, we see that $\bar{q}(G_{a,k}; u, \mathbf{x})$ equals

$$\sum_{A \subseteq V'} x_A \Big(u^{r(A)} \big(\underbrace{\sum_{\emptyset \subseteq S \subseteq \{a_1,\ldots,a_k\}} x_S + x_a u^2 \cdot \sum_{\emptyset \subsetneq S \subseteq \{a_1,\ldots,a_k\}} x_S}_{=:p(k,u,\mathbf{x})} \big) + x_a u^{r_a(A)} \Big)$$

Note that $p(k, u, \mathbf{x})$ does only depend on $x_a, x_{a_1}, \ldots, x_{a_k}$, but not on x_v for any $v \in V'$. As we have

$$\bar{q}(G; u, \mathbf{X}) = \sum_{A \subseteq V'} X_A (u^{r(A)} + X_a u^{r_a(A)}),$$

we conclude

$$\bar{q}(G_{a,k}; u, \mathbf{x}) = p(k, u, \mathbf{x}) \bar{q}(G; u, \mathbf{X}),$$

where $X_v = x_v$ for $v \in V'$ and $X_a = \frac{x_a}{p(k,u,\mathbf{x})}$.

We can perform a k-comb operation at every $a \in V$ and call the result G_k. Substituting x for $x_v, v \in G_k$, concludes the proof. □

Theorem 3.24 implies the following reduction.

Proposition 3.25 ([BH08, Proposition 3.6])**.** *Let $p(k, u, x) = (1 + x)^k(xu^2 + 1) - xu^2$. Let k be a positive integer and $\mu, \xi \in \widetilde{\mathbb{Q}}$. If $p(k, \mu, \xi) \neq 0$, we have $\bar{q}(-; \mu, \xi/p(k, \mu, \xi)) \preceq^{\mathrm{m}} \bar{q}(-; \mu, \xi)$.* □

3.2.3 Cycles

For similar reasons as with combing, we introduce a graph transformation that adds cycles. It is used in the proofs of Proposition 4.5, Proposition 4.6, Theorem 4.12 and Corollary 5.4.

Definition 3.26 (Adding Cycles)**.** *Let $G = (V, G)$ be a graph and $a \in V$ some vertex. Consider the graph $G_{a,k} = (V \cup \{1, 2, \ldots, k-1\}, E \cup \{\{a, 1\}, \{a, k-1\}\} \cup \{\{i-1, i\} \mid 1 < i < k\})$, with $1, 2, \ldots, k-1$ being new vertices. We say that $G_{a,k}$ has been obtained out of G by* adding a k-cycle *to a.*

We analyze the effect of adding cycles only for cycles of length 3 and 4. Using the method in Section 3.2.4, it should be possibly to analyze it for arbitrary large cycles.

Theorem 3.27 ([BH08, Theorem 3.7])**.** *Let G be a graph and G_k be obtained out of G by adding a k-cycle to every vertex. Then $\bar{q}(G_k; u, x) = p_k(u, x) \bar{q}(G; u, q_k(u, x)/p_k(u, x))$ for $k = 3, 4$ with $p_3(u, x) = 1 + 2x + 3x^2u^2$, $q_3(u, x) = x + x^3u^2$, $p_4(u, x) = 1 + 3x + x^2 + 2x^2u^2 + x^3u^2$, and $q_4(u, x) = x^2 + 2x^3u^2 + x^4u^2$.*

Proposition 3.28 ([BH08, Proposition 3.8]). *$\bar{q}(-;0,1) \preceq^{\mathtt{m}} \bar{q}(-;0,-1)$ and $\bar{q}(-;\mu,-4) \preceq^{\mathtt{m}} \bar{q}(-;\mu,-2)$ for every $\mu \in \widetilde{\mathbb{Q}}$.*

Proof. The first reduction follows from Theorem 3.27 adding 3-cycles, the second adding 4-cycles. □

Proof of Theorem 3.27. We use the same idea as in the proof of Theorem 3.24. Consider the case of a 3-cycle added at vertex a. Let $V' = V \setminus \{a\}$. The adjacency matrix of $G_{a,3}$ is

$$\left(\begin{array}{c|cccc} & 1 & 2 & a & V' \\ \hline 1 & & 1 & 1 & \\ 2 & 1 & & 1 & \\ a & 1 & 1 & b & \mathbf{c} \\ V' & & & \mathbf{c}^T & A_{11} \end{array}\right)$$

with empty entries being zero. Adding the second row to the first row and the second column to the first column and subsequently the first row to the third row and the first column to the third column does not change the rank and gives

$$\left(\begin{array}{c|cccc} & 1 & 2 & a & V' \\ \hline 1 & & 1 & & \\ 2 & 1 & & & \\ a & & & b & \mathbf{c} \\ V' & & & \mathbf{c}^T & A_{11} \end{array}\right).$$

This shows that $\operatorname{rk}(G_{a,3}[A] = \operatorname{rk}(G[A\setminus\{1,2\}])+2$ for all A, $\{1,2,a\} \subseteq A \subseteq V(G_{a,3})$. Using arguments similar to this one and the ones in the proof of Theorem 3.24 we find that

- $\operatorname{rk}(G_{a,3}[A]) = \operatorname{rk}(G[A \cap V'] + 2$ for all A with $\{a\} \subseteq A \subseteq V(G_{a,3})$ and either $1 \in A$ or $2 \in A$;
- $\operatorname{rk}(G_{a,3}[A]) = \operatorname{rk}(G[A]$ for all A with $\{a\} \subseteq A \subseteq V(G_{a,3})$ and $\{1,2\} \cap A = \emptyset$;
- $\operatorname{rk}(G_{a,3}[A]) = \operatorname{rk}(G[A \cap V']) + \operatorname{rk}(P_2[A \cap V(P_2)])$ for all $A \subseteq V(G_{a,3})$ with $a \notin A$, where P_2 is the the path with two vertices $1, 2$.

Letting again $r(A) := \operatorname{rk}(G[A])$ and $r_a(A) := \operatorname{rk}(G[A \cup \{a\}])$ for $A \subseteq V'$, we see that $\bar{q}(G_{a,3}; u, \mathbf{x})$ equals

$$\sum_{A \subseteq V'} x_A \Big(u^{r(A)} \big(\underbrace{1 + x_1 + x_2 + x_1x_2u^2 + x_1x_au^2 + x_2x_au^2}_{=:p_3(u,\mathbf{x})} \big) + u^{r_a(A)} \big(\underbrace{x_a + x_1x_2x_au^2}_{=:q_3(u,\mathbf{x})} \big) \Big),$$

which equals $p_3(u,\mathbf{x})\bar{q}(G;u,\mathbf{X})$ if we define $\mathbf{X}$ by $X_v = x_v$ for $v \in V'$ and $X_a = q_3(u,\mathbf{x})/p_3(u,\mathbf{x})$. We can use this identity for every vertex a and substitute x_a, $a \in V$, by a single variable x. This gives the statement of the theorem concerning 3-cycles. For 4-cycles, we proceed in a similar fashion. □

3.2.4 Paths

In this subsection, we analyze the effect of paths on the interlace polynomial $\bar{q}(G;u,x)$. We will use a combination of clones and adding paths of different length to the different clones in Chapter 5. This will provide n different point-to-point reductions, while the transformed graphs are only $\mathsf{poly}(\log n)$ times larger than the original graph.

The technique in this section differs from the ones used for clones, combs, and cycles. Basically, we solve a recursion (cf. the proof of the formula for $q(P_n)$ by Arratia et al. [ABS04b, Proposition 14]).

The following definition and lemma deals with exceptional values of the parameters (u,x) for $\bar{q}$.

Definition 3.29. *Let $u,x \in \mathbb{C}$. We say that (u,x) is* nondegenerate for path reduction *if*

1. *the two roots λ_1,λ_2 of $\lambda^2-\lambda-x(1+xu^2)$ are distinct,*
2. *$\lambda_2 \neq 1$, $\lambda_1 \neq 1$, $\lambda_2 \neq 0$, and $\lambda_1 \neq 0$,*
3. *$x+\lambda_1 \neq 0$ and $x+\lambda_2 \neq 0$, and*
4. *$x \neq 0$.*

Otherwise, we say that (u,x) is degenerate for path reduction.

Lemma 3.30. *For every $u \in \mathbb{C}$, $u^2 \neq 1$, there are at most five $x \in \mathbb{C}$ such that (u,x) is degenerated for path reduction.*

Proof. $\lambda_1 = \lambda_2$ implies $x = -1/4$ (if $u = 0$) or $x = -1/(2u^2) \pm 1/(2u)\sqrt{1/u^2-1}$ (if $u \neq 0$).

$\lambda_1 = 1$ implies $x = 0$ or $x = -1/u^2$. $\lambda_2 = 1$, $\lambda_1 = 0$, and $\lambda_2 = 0$ imply the same.

$-x = \lambda_1$ implies $x^2(u^2-1) = 0$. So does $-x = \lambda_2$. □

Theorem 3.31. *Let $G=(V,E)$ be a graph and $a_0 \in V$ a vertex. For a positive integer k, let $\tau_k G$ denote the graph G with a path of length k added at a_0, i.e. $\tau_k G = (V \cup \{a_1,\ldots,a_k\}, E \cup \{a_0a_1, a_1a_2, \ldots, a_{k-1}a_k\})$ with $a_1,\ldots,a_k$ being new vertices. Let $\mathbf{x}$ be a vertex labeling of $\tau_k G$ with variables. Then the following polynomial equation holds:*

$$\bar{q}(\tau_k G;u,\mathbf{x}) = C_k \bar{q}(G;u,\mathbf{X}), \tag{3.12}$$

where $X_v = x_v$ for all $v \in V \setminus \{a_0,\ldots,a_k\}$, $X_{a_0} = B_k/C_k$, and $B_0 = x_k$, $C_0 = 1$ and, for $0 \leq i < k$,

$$\begin{pmatrix} B_{i+1} \\ C_{i+1} \end{pmatrix} = M(x_{k-i-1}) \begin{pmatrix} B_i \\ C_i \end{pmatrix}, \tag{3.13}$$

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

Let now $(u,x) \in \mathbb{C}^2$ *be nondegenerate for path reduction,* λ_1, λ_2 *be the two roots of* $\lambda^2 - \lambda - x(1+xu^2)$*,* $x_a = x$ *for all* $a \in \{a_0, \ldots, a_k\}$ *and* ${\lambda_1}^{k+2} \neq {\lambda_2}^{k+2}$*. Then* (3.12) *holds with*

$$\begin{aligned} B_k &= \frac{x}{\lambda_2 - \lambda_1} \cdot \big(\lambda_1^k(\lambda_2 - 1) + \lambda_2^k(1 - \lambda_1)\big) \\ &= \frac{x}{\lambda_2 - \lambda_1} \cdot \big(-\lambda_1^{k+1} + \lambda_2^{k+1}\big) \quad \textit{and} \\ C_k &= \frac{1}{\lambda_2 - \lambda_1} \cdot \big(\lambda_1^{k+1}(\lambda_2 - 1) + \lambda_2^{k+1}(1 - \lambda_1)\big) \\ &= \frac{1}{\lambda_2 - \lambda_1} \cdot \big(-\lambda_1^{k+2} + \lambda_2^{k+2}\big). \end{aligned}$$

Proof. We reduce the edges of the path edge by edge using an argument as in the proof of Theorem 3.24. If τG is G with a leaf b added to vertex $a \in V(G)$, we have the polynomial equation

$$\bar{q}(\tau G; x_a, x_b) = (1 + x_b(1 + x_a u^2)) \cdot \bar{q}\Big(G; \frac{x_a}{1 + x_b(1 + x_a u^2)}\Big), \tag{3.14}$$

where $\bar{q}(\tau G; y, z)$ denotes $\bar{q}(\tau G; u, \mathbf{x})$ with $x_a = y$ and $x_b = z$, and $\bar{q}(G; y)$ denotes $\bar{q}(G; u, \mathbf{X})$ with $X_a = y$ and $X_v = x_v$ for all $v \in V(G) \setminus \{a\}$.

Let us write $\bar{q}(\tau_k G; x_0, \ldots, x_k)$ for $\bar{q}(\tau_k G; u, \mathbf{x})$ where $x_{a_j} = x_j$, $0 \leq j \leq k$. Let us argue that we defined the B_i, C_i such that, for $0 \leq i \leq k$,

$$\bar{q}(\tau_k G; x_0, \ldots, x_k) = C_i \bar{q}\left(\tau_{k-i} G; x_0, \ldots, x_{k-i-1}, \frac{B_i}{C_i}\right). \tag{3.15}$$

This is trivial for $i = 0$. As we have

$$1 + \frac{B_i}{C_i}\left(1 + x_{k-i-1}u^2\right) = \frac{C_i + B_i\left(1 + x_{k-i-1}u^2\right)}{C_i} = \frac{C_{i+1}}{C_i},$$

we see that (3.15) holds for all $1 \leq i \leq k$: Use (3.14) in the following inductive step:

$$\begin{aligned} \bar{q}(\tau_k G; x_0, \ldots, x_k) &= C_i \bar{q}(\tau_{k-i} G; x_0, \ldots, x_{k-i-1}, \frac{B_i}{C_i}) \\ &= C_i \frac{C_{i+1}}{C_i} \bar{q}\left(\tau_{k-i-1} G; x_0, \ldots, x_{k-i-2}, \frac{x_{k-i-1}}{\frac{C_{i+1}}{C_i}}\right) \\ &= C_{i+1} \bar{q}\left(\tau_{k-i-1} G; x_0, \ldots, x_{k-i-2}, \frac{B_{i+1}}{C_{i+1}}\right). \end{aligned}$$

Thus, (3.12) holds as a polynomial equality.

Let us now consider u and x as complex numbers. As (u,x) is nondegenerate, matrix $M(x)$ in can be diagonalized as $M(x) = SDS^{-1}$ with

$$S = \begin{pmatrix} x & x \\ \lambda_1 & \lambda_2 \end{pmatrix}, \quad D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \quad S^{-1} = \frac{1}{x(\lambda_2 - \lambda_1)} \begin{pmatrix} \lambda_2 & -x \\ -\lambda_1 & x \end{pmatrix},$$

$\lambda_{1,2} = \frac{1}{2} \pm \sqrt{\frac{1}{4} + x(1 + xu^2)}$. Now we substitute variable x_v by complex number x for all $v \in \{a_0, \ldots, a_k\}$ and $M(x)$ by SDS^{-1}. This yields the statement of the theorem. □

3.3 Uniform Local Transformations

The aim of this section is twofold. First, we generalize vertex/edge induced subgraph polynomials. While graph polynomials of this class are very pleasant, they can count induced subgraphs only. If we want to count substructures such as vertex covers, we need a slightly more general concept. Therefore we introduce vertex/edge function polynomials (Definition 3.33).

As we have seen in Section 3.2 and will discuss below, there are more useful graph transformations but only cloning. All these graph transformations yield point-to-point reductions by virtually the same proof technique. The second aim of this section is to exhibit a general property of graph transformation (Definition 3.40 and Definition 3.46) that enables to use this proof technique.

Definition 3.32 (Vertex/Edge Functions)**.** *Let $\mathcal{G}$ be the class of graphs, $\mathcal{V}$ be the class of all vertices, and $\mathcal{E}$ be the class of all edges. Let R be a ring and $\mathcal{X}$ a set of $\mathcal{G}$-indexed variables.*

A function $f : \mathcal{G} \times 2^{\mathcal{V}} \to R[\mathcal{X}]$ is called a vertex function *if it has the following property: If φ is a graph isomorphism from $G_1 \in \mathcal{G}$ onto $G_2 \in \mathcal{G}$, then*

$$f(G_1, A)_{\varphi} = f(G_2, \varphi(A)) \quad \text{for all } A \subseteq V(G_1). \tag{3.16}$$

A function $f : \mathcal{G} \times 2^{\mathcal{E}} \to R[\mathcal{X}]$ is called an edge function *if it has the following property: If φ is a graph isomorphism from $G_1 \in \mathcal{G}$ onto $G_2 \in \mathcal{G}$, then*

$$f(G_1, A)_{\varphi} = f(G_2, \varphi(A)) \quad \text{for all } A \subseteq E(G_1). \tag{3.17}$$

Definition 3.33 (Vertex/Edge Function Polynomials)**.** *Let R be a ring, $\mathcal{X}$ a set of $\mathcal{G}$-indexed variables, $f : \mathcal{G} \times 2^{\mathcal{V}} \to R[\mathcal{X}]$ be a vertex function and $\mathbf{x}$ a vertex indexed variable that is not in $\mathcal{X}$. Then we define the* vertex function polynomial *(VFP) of f, $\mathrm{VFP}\, f := \mathrm{VFP}_{\mathbf{x}} f$, to be the following function, which maps a graph $G = (V, E)$ to a polynomial in $R[\mathbf{x}, \mathcal{X}]$:*

$$(\mathrm{VFP}_{\mathbf{x}} f)(G; \mathbf{x}) = \sum_{A \subseteq V} x_A f(G, A). \tag{3.18}$$

Let f be an edge function and $\mathbf{x}$ an edge indexed variable that is not in R. Then we define the edge function polynomial *(EFP) of f, $\mathrm{EFP}\, f := \mathrm{EFP}_{\mathbf{x}} f$, to be the following function, which maps a graph $G = (V, E)$ to a polynomial in $R[\mathbf{x}, \mathcal{X}]$:*

$$(\mathrm{EFP}_{\mathbf{x}} f)(G; \mathbf{x}) = \sum_{A \subseteq E} x_A f(G, A) \tag{3.19}$$

Remark 3.34. *Every VFP and every EFP is a graph polynomial.*

Proof. Similar to the proof of Remark 3.2 using (3.16) and (3.17). □

Remark 3.35. *Every vertex (edge) induced subgraph polynomial is a vertex (edge) function polynomial.*

Example 3.36. *Let us define a graph polynomial that counts vertex covers of a graph $G = (V, E)$.*

$$\mathrm{vc}(G; \mathbf{x}) = \sum_A x_A, \tag{3.20}$$

where the sum is over all $A \subseteq V$ that are vertex covers of G. A subset A is called a vertex cover of G if every edge of G has one of its end vertices in A.

It is not clear whether vc *can be expressed as a vertex induced subgraph polynomial. But* vc *is immediately seen to be a vertex function polynomial:* $\mathrm{vc} = \mathrm{VFP}\, f$*, where*

$$f(G, A) = \begin{cases} 1 & \text{if } A \text{ is a vertex cover in } G \\ 0 & \text{otherwise.} \end{cases} \tag{3.21}$$

Example 3.37. *Consider the domination polynomial defined by Alikhani and Peng [AP09]. Just like the vertex cover polynomial, it can not be defined as a VSP but as a VFP: For a graph $G = (V, E)$ and $A \subseteq V$, let*

$$d(G, A) = \begin{cases} 1 & \text{if } A \text{ is a dominating set for } G \\ 0 & \text{otherwise.} \end{cases} \tag{3.22}$$

($A \subseteq V$ is a dominating set for G if, for every vertex $v \in V$, $v \in A$ or v has a neighbor in A.)

We define the multivariate domination polynomial $D = \mathrm{VFP}d$. Then, $D(G, x)$, the domination polynomial of Alikhani and Peng, equals $D(G; \mathbf{x})$ if we let $x_a = x$ for all $a \in V$.

As we have mentioned at the beginning of Section 3.1, multivariate graph polynomials can "simulate" a part of the graph by a single vertex/edge if the corresponding vertex- or edge-indexed variable is set to an appropriate value. Let us define a class of graph transformations that do the "inverse": a single vertex/edge is substituted by a larger structure.

Definition 3.38. *A* local transformation *for vertices (edges) is a mapping $\tau : \mathcal{V} \times \mathcal{G} \to \mathcal{G}$ ($\tau : \mathcal{E} \times \mathcal{G} \to \mathcal{G}$) that takes a vertex (edge) a of a graph $G = (V, E)$ and maps it to the graph $\tau_a G$ such that the following holds: There is a finite set T_a such that $V(\tau_a G) = (V \setminus \{a\}) \cup T_a$ ($E(\tau_a G) = (E \setminus \{a\}) \cup T_a$).*

Example 3.39.
1. *Vertex (edge) cloning is a local transformation for vertices (edges).*
2. *Adding combs, cycles, or paths is a local transformation for vertices.*

3. *Edge stretching (i.e. replacing an edge by a path) is a local transformation for edges.*

Whenever the following criterion is fulfilled, a local transformation yields a point-to-point reduction by simulating the introduced substructure via a well-chosen vertex/edge weight.

Definition 3.40. *Let f be a vertex (edge) function mapping into a ring R and τ a local transformation for vertices (edges, resp.). We call τ* uniform *with respect to f if the following holds: For every graph $G = (V, E)$, every vertex (edge) a of G, and every $S \subseteq T_a$ there exist $f_{a,0}(S)$ and $f_{a,1}(S)$ in R such that*

$$f(\tau_a G, B \cup S) = f_{a,0}(S) f(G, B) + f_{a,1}(S) f(G, B \cup \{a\}) \ \forall B \subseteq A \setminus \{a\}, \quad (3.23)$$

where $A = V$ ($A = E$, resp.).

Example 3.41. 1. *Cloning is uniform with respect to every graph polynomial that supports cloning. We have $f_{a,0}(\emptyset) = 1$, $f_{a,1}(\emptyset) = 0$, $f_{a,0}(\{a\}) = f_{a,0}(\{a'\}) = f_{a,0}(\{a, a'\}) = 0$, and $f_{a,1}(\{a\}) = f_{a,1}(\{a'\}) = f_{a,1}(\{a, a'\}) = 1$. This follows from the fact that graph polynomials map isomorphic graphs to the same polynomial and from Definition 3.12.*

2. *Adding combs, cycles, or paths (resp.) is uniform with respect to $(G, A) \mapsto u^{\operatorname{rk} G[A]}$. This can be seen in the proofs of Theorems 3.24 and 3.27, and it is implicit in the proof of Theorem 3.31.*

3. *Edge stretching is uniform with respect to $(G, A) \mapsto q^{k(G[A])}$. This is what Sokal observes in his series-reduction identity for the Tutte polynomial [Sok05, (4.24)]. More precisely, he observes it for the case of replacing an edge by a path of length 2. Then it follows for paths of arbitrary length by repeated application of his "mnemonic" [Sok05, (4.26)], as Bläser and Makowsky observe [BM06].*

Example 3.42. *Vertex cloning is uniform with respect to the f that defines the vertex cover polynomial, see* (3.21).

Proof. Let $G = (V, E)$ be a graph, $a \in V$ a vertex, $V' = V \setminus \{a\}$ and $B \subseteq V'$. Then:

1. B is a vertex cover in $\kappa_a G$ iff B is a vertex cover in G.
2. $B \cup \{a\}$ is a vertex cover in $\kappa_a G$ iff B is a vertex cover in G: The edges incident to a' have to be covered by B.
3. $B \cup \{a'\}$ is a vertex cover in $\kappa_a G$ iff B is a vertex cover in G.
4. $B \cup \{a, a'\}$ is a vertex cover in $\kappa_a G$ iff $B \cup \{a\}$ is a vertex cover in G. □

Example 3.43. *Vertex cloning is uniform with respect to the d that defines the multivariate dominating polynomial, see* (3.22).

Proof. Completely analogous to the proof of Example 3.42. □

Counter Example 3.44. *Edge stretching is not uniform with respect to m, the function that defines the matching polynomial (Example 3.6).*

Proof. Let G be a path of length 3, consisting of edges $e_1 = \{u, v\}$, $e_2 = \{v, w\}$, $e_3 = \{w, x\}$. Let us stretch e_2, so as to obtain $e_2' = \{v, v'\}$, $e_2'' = \{v', w\}$. In $\tau_{e_2}G$, e_2', e_2'' replace e_2.

For $B = \{e_1\}$, we have $f(G, B) = 1$, $f(G, B \cup \{e_2\}) = 0$ and $f(\tau_{e_2}G, B \cup \{e_2'\}) = 0$. Thus, if stretching is uniform, we have

$$f_0(\{e_2'\}) = 0. \tag{3.24}$$

On the other hand, for $B = \{e_3\}$, we have $f(G, B) = 1$, $f(G, B \cup \{e_2\}) = 0$ as well. But $f(\tau_{e_2}G, B \cup \{e_2'\}) = 1$. Thus, for stretching to be uniform, we also need $f_0(\{e_2'\}) = 1$. This contradicts (3.24). □

The proof of the following theorem captures what all the point-to-point reduction proofs—i.e. the one for cloning, adding combs, cycles or path, as well as Sokal's serial reduction—have in common. It will also yield new point-to-point reductions for the vertex cover polynomial and the domination polynomial.

Theorem 3.45. *Let f be a vertex (edge) function mapping into a ring R and τ that is uniform with respect to f. Let*

$$g_{a,i} = \sum_{S \subseteq T_a} x_S f_{a,i}, \quad i = 0, 1,$$

and, for given vertex (edge) indexed $\mathbf{x}$, let $\mathbf{X}$ be a vertex (edge) indexed variable with $X_b = x_b$ for $b \in V \setminus \{a\}$ ($b \in E \setminus \{a\}$, resp.) and $X_a = \frac{g_{a,1}}{g_{a,0}}$. Let $F = \mathrm{VFP}\, f$ ($F = \mathrm{EFP}\, f$, resp.). Then we have

$$F(\tau_a G; \mathbf{x}) = g_{a,0} \cdot F(G; \mathbf{X}). \tag{3.25}$$

Proof. We give the proof for vertex functions. The case of edge functions is completely analogous.

Let $G = (V, E)$ be a graph, $a \in V$, $V' = V \setminus \{a\}$.

$$\begin{aligned}
\mathrm{VFP}\, f(\tau_a G; \mathbf{x}) &= \sum_{B \subseteq V'} x_B \sum_{S \subseteq T_a} x_S f(\tau_a G, B \cup S) \\
&= \sum_{B \subseteq V'} x_B \sum_{S \subseteq T_a} x_S (f_{a,0} f(G, B) + f_{a,1} f(G, B \cup \{a\})) \\
&= \sum_{B \subseteq V'} x_B (g_{a,0} f(G, B) + g_{a,1} f(G, B \cup \{a\})) \\
&= g_{a,0} \mathrm{VFP}\, f(G; \mathbf{X}).
\end{aligned}$$

□

Applying a local transformation to one particular vertex/edge is only an intermediate step in most of the applications. Often, we apply the same local transformation to all vertices/edges. The criteria in the following definition distinguish transformations that support this endeavor.

Definition 3.46. *Let f be a vertex (edge) function mapping into a ring R and τ be a local transformation for vertices (edges, resp.).*

1. *We say that τ is* strongly uniform *with respect to f if*
   a) *it is uniform,*
   b) *it is commutative, i.e. $\tau_a\tau_b G = \tau_b\tau_a G$ for every graph G and every pair (a, b) of vertices (edges) of G, and*
   c) *the size of T_a in the definition of local transformation (Definition 3.38) as well as the functions $f_{a,0}$ and $f_{a,1}$ in Definition 3.40 do not depend on a.*
2. *Let τ be strongly uniform with respect to f. For a vertex (edge) set $A = \{a_1, \dots, a_\ell\}$ of G, we define*

$$\tau_A G = \tau_{a_\ell} \cdots \tau_{a_1} G. \tag{3.26}$$

Note that the commutativity condition uses vertices/edges a, b in the *original* graph G. It does *not* include a statement about transforming vertices/edges that have been created by a preceding transformation.

Example 3.47. *Cloning is strongly uniform with respect to graph polynomials that support cloning.*

Example 3.48. *Edge stretching is strongly uniform with respect to $(G, A) \mapsto q^{k(G[A])}$.*

If we can apply the same local transformation to all vertices/edges, we can continue to obtain a point-to-point reduction in ordinary, i.e. not G-indexed, variables.

Theorem 3.49. *Let f be a vertex (edge) function mapping into a ring R and τ be a local transformation that is strongly uniform with respect to f. Let $F = \mathrm{VFP}\, f$ ($F = \mathrm{EFP}\, f$, resp.). Let*

$$g_i = \sum_{S \subseteq T_a} x^{|S|} f_{a,i}, \quad i = 0, 1,$$

for any vertex (edge) a. Let $A = V$ ($A = E$, resp.). Then we have

$$F(\tau_A G; x) = (g_0)^{|A|} \cdot F(G; \frac{g_1}{g_0}). \tag{3.27}$$

Proof. We apply Theorem 3.45 for every vertex (edge) of G. □

Example 3.50. *1. Theorems 3.24 (adding combs), 3.27 (adding cycles), and 3.31 (adding paths) are instances of Theorem 3.49. However, to determine the actual g_0 and g_1, we used a different technique in the proof of Theorem 3.31.*

2. The formulas that relate the Tutte polynomial of k-stretch of a graph to the Tutte polynomial of the original graph [JVW90, BM06] are instances of Theorem 3.49.

Example 3.51.

$$\mathrm{vc}(\kappa_V^{k-1}G; x) = ((1+x)^k - x^k)^{|V|}\mathrm{vc}\Big(G; \frac{x^k}{(1+x)^k - x^k}\Big). \tag{3.28}$$

Proof. We generalize the observations in Example 3.42 and let τ denote the transformation cloning vertex a $k-1$ times. Let $T = \{a_1, \ldots, a_k\}$ the substitute for vertex a. Let $V' = V \setminus \{a\}$. For $B \subseteq V'$ we have that,

1. if $S \neq T$, $B \cup S$ is a vertex cover in $\tau_a G$ iff B is a vertex cover in G; and
2. $B \cup T$ is a vertex cover in $\tau_a G$ iff $B \cup \{a\}$ is a vertex cover in G.

Thus, we can apply Theorem 3.49 with $g_1 = x^k$ and

$$g_0 = \sum_{\substack{S \subseteq T \\ S \neq T}} x^{|S|} = (1+x)^k - x^k. \qquad \square$$

Example 3.52.

$$D(\kappa_V^{k-1}G; x) = ((1+x)^k - x^k)^{|V|}D\Big(G; \frac{x^k}{(1+x)^k - x^k}\Big). \tag{3.29}$$

Proof. Completely analogous to the proof of Example 3.51 $\square$

Remark 3.53. *From* (3.28) *and* (3.29) *it follows that, for every $\xi \in \mathbb{Q}\setminus\{0, -\frac{1}{2}, -1\}$, $\mathrm{vc}(-; x) \preceq_T \mathrm{vc}(-; \xi)$ and $D(-; x) \preceq_T D(-; \xi)$. In particular, evaluating the vertex cover polynomial is #P-hard for all these ξ.*

Proof (Sketch). Use a similar technique as in Theorem 3.19. To obtain an interpolating family, observe that, for $\xi \notin \{0, -1\}$ and positive integers $k \neq \ell$, we have

$$\frac{\xi^k}{(1+\xi)^k - \xi^k} = \frac{\xi^\ell}{(1+\xi)^\ell - \xi^\ell}$$

iff $\xi = -1/2$. #P-hardness of counting vertex covers is known [Val79b]. $\square$

3.4 Discussion of the General Methods

Let us compare the concepts of Section 3.1 and Section 3.3. VSPs/ESPs are very elegant as they lead to a result as strong as Theorem 3.19 by assuming just (3.1). They have many applications, see the examples at the end of Section 3.1. We only use the cloning transformation[3], which always yields the point-to-point reduction $x \leftrightarrow (1+x)^k - 1$ (Theorem 3.15).

Strongly uniform local transformations and VFPs/EFPs, on the other hand, are more flexible. They can define graph polynomials, such as the vertex cover polynomial, that are not known to be definable as a VSP/ESP. Uniformity of a local transformation captures what is necessary to obtain *any* point-to-point reduction via expressing changes to the graph in changes of the weights. The concrete reduction can vary; and it is sometimes tedious to state it explicitly.

Even though our machinery is helpful and obtains very interesting results, it clearly has its limits. The first problem, of course, is to come up with a graph transformation that actually supports cloning with our polynomial or is (strongly) uniform. We have seen in Counter Example 3.44 that this actually may happen; and we could give more examples.

Another problem is more fundamental: We might not be able to define a meaningful multivariate version of the graph polynomial we are considering. Take the following example. The chromatic polynomial $\chi(G;q)$ of a graph $G=(V,E)$ equals

$$\sum_{\substack{A\subseteq E\\ A \text{ contains no broken circuit}}} (-1)^{|A|} x^{|V|-|A|}, \tag{3.30}$$

where the sum is over all $A \subseteq E$ that do not contain a broken circuit. A broken circuit is defined in the following way. Fix an order on the edges of the graph. A broken circuit is an edge set obtained from a circuit by removing the last (according to the just defined order) edge. In this way, (3.30) may depend on the edge order. But in fact, by Whitney's broken circuit theorem [Whi32], (3.30) does not depend on the edge order.

Let us try to generalize (3.30) to a multivariate polynomial. The straightforward way would be to consider

$$F(G;\mathbf{x}) = \sum_{\substack{A\subseteq E(G)\\ A \text{ contains no broken circuit}}} x_A. \tag{3.31}$$

and to try to write this as an EFP, $F = \mathrm{EFP}\, f$. For a fixed edge order R we could define

$$f(G,A) = \begin{cases} 1 & \text{if } A \text{ contains no broken circuit with respect to } R \\ 0 & \text{if } A \text{ contains a broken circuit with respect to } R. \end{cases}$$

[3] even though we could apply other transformations as well

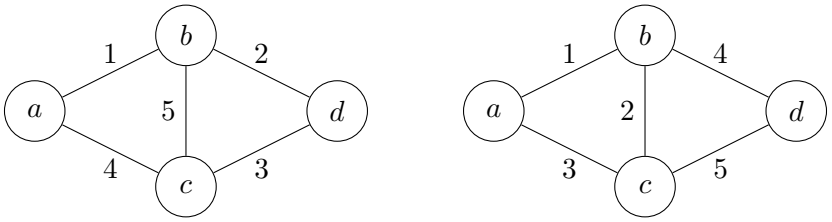

Figure 3.1: Two edge orders of a graph showing that (3.31) is not a graph polynomial.

However, consider the following example, which shows that this f is not an edge function as it violates (3.17): Let G be the triangle with edges e_1, e_2, e_3. If the edge order is $e_1 < e_2 < e_3$, then $f(\{e_1, e_3\}) = 1$ and $f(\{e_1, e_2\}) = 0$ as $\{e_1, e_2\}$ is the only broken circuit with respect to this order. Consider the function φ that maps e_1 to e_2, e_2 to e_3, and e_3 to e_1. This function can be extended to a graph isomorphism from G onto itself by defining it on the vertices of the triangle in the obvious way. Slightly abusing notation, let us denote this isomorphism by φ, too. Now we have $1 = f(G, \{e_1, e_3\}) = f(G, \{e_1, e_3\})_\varphi$, but $f(G, \varphi(\{e_1, e_3\})) = f(G, \{e_1, e_2\}) = 0$.

In fact, (3.31) depends on the edge order and, thus, is not a multivariate graph polynomial: Consider Figure 3.1. According to the edge order shown on the left half, $A = \{ab, bd, cd, ac\}$, the only cycle of length 4, contains two broken circuits, $\{ab, ac\}$ and $\{bd, cd\}$. But according to the edge order shown on the right half, this A does not contain a broken circuit as edge bc is contained in every broken circuit.

Despite these limits, the hardness results in the subsequent two chapters are based almost entirely on cloning or other strongly uniform local transformations.

3.5 Open Problems

The following questions are directions for further research regarding the topics of this chapter:

- Is there a graph transformation based point to-point-reduction for the interlace polynomial $\bar{q}(G; u, x)$ that reduces between different values of u? Theorems 3.21, 3.24, 3.27, and 3.31 reduce only between different values of x.
- Following Remark 3.53, can we generalize Theorem 3.19 to VFPs/EFPs?

4 Hardness for #P and NP

We give hardness results that show that exact or approximate evaluation of several graph polynomials needs superpolynomial time, unless #P = FP or NP = RP.

First, we prove that the two-variable interlace polynomial of Arratia, Bollobás, and Sorkin is #P-hard to evaluate almost everywhere on the plane. This includes the complexity of the independent set polynomial and the vertex-rank interlace polynomial. Section 4.2 contains an inapproximability result for the independent set polynomial. In the last section, we show that the extended bivariate chromatic polynomial is #P-hard to evaluate at almost every point of $\mathbb{Q}^3$. This includes a similar statement for the bivariate chromatic polynomial.

In most proofs of the hardness results, we use reductions that are obtained by cloning or other graph transformations introduced in Chapter 3.

4.1 Evaluation of the Interlace Polynomial

The goal of this section is to uncover the complexity maps for $\bar{q}$ and q as indicated in Figure 4.1. While the left hand side (complexity map for $\bar{q}$) is intended to follow the *arguments* that prove the hardness, the right hand side (complexity map for q) focuses on presenting the *results*.

Remark 4.1. *As* $\bar{q}(G;\mu,0) = 1$ *and* $\bar{q}(G;1,\xi) = (1+\xi)^{|V|}$, $\bar{q}(-;\mu,0)$ *and* $\bar{q}(-;1,\xi)$ *are trivially solvable in polynomial time for every* $\mu,\xi \in \widetilde{\mathbb{Q}}$. □

Thus, on the thick black lines $x = 0$ and $u = 1$ in the left half of Figure 4.1, $\bar{q}$ can be evaluated in polynomial time. By Lemma 1.17, these lines in the complexity map for $\bar{q}$ correspond to the point $(1,1)$ and the line $x = y$, resp., in the complexity map for q, see the right half of Figure 4.1.

4.1.1 Known-To-Be-Hard Points

Let us state what is already known about the complexity of evaluating the interlace polynomial: The Tutte-Martin connection and Vertigan's hardness result for evaluating the Tutte polynomial on planar graphs tell us that $\bar{q}$ is #P-hard to evaluate almost everywhere on the dashed hyperbola in Figure 4.1 (Corollary 4.2). Remark 4.3 shows that $\bar{q}$ is #P-hard to evaluate at $(0,1)$.

We set $\alpha = \sqrt{2}$ and $\beta = 1/\sqrt{2}$. Let $B_1 = \{\pm 1, \pm\beta, 0\}$.

Corollary 4.2. *Evaluating the vertex-nullity interlace polynomial* q_N *is #P-hard almost everywhere. In particular, we have:*

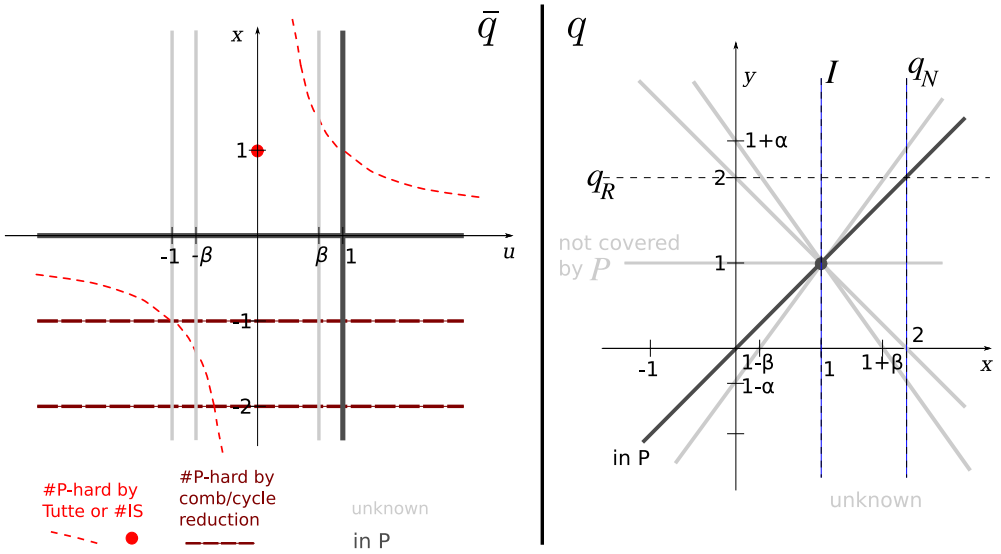

Figure 4.1: Complexity of the interlace polynomials $\bar{q}$ and q. $\alpha = \sqrt{2}$, $\beta = 1/\sqrt{2}$

- *The problem $q_N(-;2)$ is trivially solvable in polynomial time.*
- *For every $\upsilon \in \widetilde{\mathbb{Q}} \setminus \{0, 1, 2, 1 \pm \alpha\}$, the problem $q_N(-;\upsilon) = q(-;2,\upsilon)$ is #P-hard. Or, in other words, for every $\mu \in \widetilde{\mathbb{Q}} \setminus B_1$, the problem $\bar{q}(-;\mu,1/\mu)$ is #P-hard.*

Proof. Theorem 1.39 and Theorem 1.19. □

Remark 4.3. *$\bar{q}(-;0,1)$ is #P-hard as $\bar{q}(G;0,1)$ equals the number of independent sets of G, which is #P-hard to compute [Val79b].* □

4.1.2 Reduction from Hard Points

The cloning reduction allows us to spread the known hardness over almost the whole plane:

Proposition 4.4. *Let $B_1 = \{\pm 1, \pm\beta, 0\}$ and $B_2 = \{0, -1, -2\}$ (as defined on Pages 77 and 63, resp.). Let $(\mu,\xi) \in ((\widetilde{\mathbb{Q}} \setminus B_1) \cup \{0\}) \times (\widetilde{\mathbb{Q}} \setminus B_2)$. Then $\bar{q}(-;\mu,\xi)$ is #P-hard.*

Proof. Combine Corollary 4.2 and Remark 4.3 with Proposition 3.22. □

This tells us that $\bar{q}$ is #P-hard to evaluate at every point in the left half of Figure 4.1 not lying on one of the seven thick lines (three of which are solid gray ones, two of which are solid black ones, and two of which are dashed brown ones). Using the comb and cycle reductions, we are able to reveal the hardness of the interlace polynomial $\bar{q}$ on the lines $x = -1$ and $x = -2$:

Proposition 4.5. *For $\mu \in (\widetilde{\mathbb{Q}} \setminus B_1) \cup \{0\}$, the problem $\bar{q}(-;\mu,-1)$ is #P-hard.*

Proof. For $\mu = 0$, we use Proposition 3.28 and Remark 4.3. If $\mu \neq 0$, we can use Proposition 3.25, which yields $P(\mu,-1/\mu^2) \preceq^{\mathtt{m}} P(\mu,-1)$. For $\mu = \pm 1$, this reduces $(\pm 1,-1)$ to itself. For $\mu = \pm\beta$, this reduces $(\beta,-2)$ to $(\beta,-1)$ and $(-\beta,-2)$ to $(-\beta,-1)$. For other μ, this gives a reduction of some point, which is already known as #P-hard by Proposition 4.4, to $(\mu,-1)$. □

Proposition 4.6. *For $\mu \in (\widetilde{\mathbb{Q}} \setminus B_1) \cup \{0\}$ the problem $P(\mu,-2)$ is #P-hard.*

Proof. Use Proposition 3.28 and Proposition 4.4. □

4.1.3 Conclusion

First we summarize our knowledge about $\bar{q}$.

Theorem 4.7. *Let $\beta = 1/\sqrt{2}$.*

1. *$\bar{q}(-;\mu,\xi)$ is computable in polynomial time on the lines $\mu = 1$ and $\xi = 0$.*
2. *For $(\mu,\xi) \in (\widetilde{\mathbb{Q}} \setminus \{-1,-\beta,\beta,1\}) \times (\widetilde{\mathbb{Q}} \setminus \{0\})$, the problem $\bar{q}(-;\mu,\xi)$ is #P-hard.*

Proof. Summary of Remark 4.1, Proposition 4.4, Proposition 4.5, Proposition 4.6. □

An interesting special case is the next corollary about the complexity of the independent set polynomial, which also follows from earlier work by Averbouch and Makowsky [AM07].

Corollary 4.8. *Evaluating the independent set polynomial, i.e. $I(-;\lambda) = \bar{q}(-;0,\lambda) = q(-;1,1+\lambda)$, is #P-hard at all $\lambda \in \mathbb{Q}$, except at $\lambda = 0$, where it is computable in polynomial time.*

Now we turn to the complexity of q, see also the right half of Figure 4.1.

Theorem 4.9. *The two-variable interlace polynomial q is #P-hard to evaluate almost everywhere. In particular, we have:*

1. *$q(-;\xi,\upsilon)$ is computable in polynomial time on the line $\xi = \upsilon$.*
2. *Let $\xi \in \widetilde{\mathbb{Q}} \setminus \{1\}$ and x be an indeterminate. Then $q(-;\xi,1)$ is as hard as computing the whole polynomial, $q(-;x,1)$.*
3. *$q(-;\xi,\upsilon)$ is #P-hard for all*

$$(\xi,\upsilon) \in \{(\xi,\upsilon) \in \widetilde{\mathbb{Q}}^2 \mid \upsilon \neq \pm(\xi-1)+1 \text{ and } \upsilon \neq \pm\sqrt{2}(\xi-1)+1 \text{ and } \upsilon \neq 1\}.$$

Proof. (1) and (3) follow from Remark 4.1 and Theorem 4.7 using Lemma 1.17.

For $\xi \neq 1$, (3.9) gives $q(G_k;\xi,1) = q(G;k(\xi-1)+1,1)$. This shows that q has an interpolating family at $(\xi,1)$. Now (2) follows from Theorem 3.18. □

Theorem 4.9 implies the following result on the vertex-rank interlace polynomial.

Corollary 4.10. *Let $\beta = 1/\sqrt{2}$. Evaluating the vertex-rank interlace polynomial, $q_R(-;\xi)$, is #P-hard at all $\xi \in \widetilde{\mathbb{Q}}$ except at $\xi = 0, 1-\beta, 1+\beta$ (complexity open) and $\xi = 2$ (computable in polynomial time).* □

4.2 Inapproximability of the Independent Set Polynomial

Provided we can evaluate the independent set polynomial at some fixed point, vertex cloning (adding combs, resp.) allows us to evaluate it at very large points. Let us exploit this to prove that the independent set polynomial is hard to approximate almost everywhere.

Recall the definition of FPRAS (Definition 1.41) and $2^{n^{1-\varepsilon}}$-approximation (Definition 1.42).

Lemma 4.11. *For every $\lambda \in \mathbb{Q}$, $0 \neq |1+\lambda| \neq 1$, and every ε, $0 < \varepsilon < 1$, there is no randomized polynomial time $2^{n^{1-\varepsilon}}$-approximation algorithm for $I(-;\lambda)$, unless* RP = NP.

Theorem 4.12. *For every $\lambda \in \mathbb{Q} \setminus \{0\}$ and every ε, $0 < \varepsilon < 1$, there is no randomized polynomial time $2^{n^{1-\varepsilon}}$-approximation algorithm (and thus also no FPRAS) for $I(-;\lambda)$, unless* RP = NP.

Proof. Lemma 4.11 gives the inapproximability at $\lambda \in \mathbb{Q} \setminus \{-2,-1,0\}$. By (3.10), we could turn an approximation algorithm for $I(-;-2)$ into an approximation algorithm for $I(-;2)$, which would imply RP = NP by Lemma 4.11. For $I(-;-1)$, we use Theorem 3.27. □

Proof of Lemma 4.11. Fix $\lambda \in \mathbb{Q}, 0 \neq |1+\lambda| \neq 1$, and ε, $0 < \varepsilon < 1$. Assume we have a randomized $2^{n^{1-\varepsilon}}$-approximation algorithm $\mathcal{A}$ for $I(-;\lambda)$. Given a graph G, Theorem 3.21 and Theorem 3.24, resp., will allow us to evaluate the independent set polynomial at a point ξ with $|\xi|$ that large, that an approximation of $I(G;\xi)$ can be used to recover the degree of $I(G;x)$, which is the size of a maximum independent set of G. As computing this number is NP-hard, a randomized $2^{n^{1-\varepsilon}}$-approximation algorithm for $I(-;\lambda)$ would yield an RP-algorithm for an NP-hard problem, which implies RP = NP.

Let $G = (V,E)$ be a graph with $|V| = n$. We distinguish two cases. If $|1+\lambda| > 1$, we choose a positive integer ℓ such that $(n\ell)^{1-\varepsilon} \geq n^2$ and with $\xi := (1+\lambda)^\ell - 1$ we have

$$|\xi| > 2^{2(n\ell)^{1-\varepsilon}+n+2}. \tag{4.1}$$

As λ and ε are constant, this can be achieved by choosing $\ell = \mathsf{poly}(n)$. If $0 < |1+\lambda| < 1$, we choose a positive integer ℓ such that with $\xi := \frac{\lambda}{(1+\lambda)^\ell}$ (4.1) holds. By Theorem 3.21 (Theorem 3.24, resp.) we have $I(G;\xi) = I(G_\ell;\lambda)$ ($I(G;\xi) = (1+\lambda)^{-\ell|V|} I(G_\ell;\lambda)$, resp.). Algorithm $\mathcal{A}$ returns on input G_ℓ within time $\mathsf{poly}(n\ell) =$

$\mathsf{poly}(n)$ an approximation $\widetilde{I}(G_\ell;\lambda)$, such that with $\widetilde{I}(G;\xi) := \widetilde{I}(G_\ell;\lambda)$ ($\widetilde{I}(G;\xi) := \frac{\widetilde{I}(G_\ell;\lambda)}{(1+\lambda)^{\ell|V|}}$, resp.) we have

$$2^{-(n\ell)^{1-\varepsilon}} I(G;\xi) \leq \widetilde{I}(G;\xi) \leq 2^{(n\ell)^{1-\varepsilon}} I(G;\xi) \tag{4.2}$$

with high probability.

Let c be the size of a maximum independent set of G, and let N be the number of independent sets of maximum size. We have

$$I(G;x) = Nx^c + \sum_{0\leq j\leq c-1} i(G;j)x^j$$

and thus

$$\begin{aligned}\left|\frac{I(G;\xi)}{\xi^c} - N\right| &\leq \sum_{0\leq j\leq c-1} i(G;j)|\xi|^{j-c} \\ &\leq c2^n|\xi|^{-1} \leq 2^{\log n+n}|\xi|^{-1} \overset{(4.1)}{<} \frac{1}{2}.\end{aligned} \tag{4.3}$$

If we could evaluate $I(G;\xi)$ exactly, we could try all $c \in \{1,\ldots,n\}$ to find the one for which $\frac{I(G;\xi)}{\xi^c}$ is a good estimation for N, $1 \leq N \leq 2^n$. This c is unique as $|\xi| > 2^{n^2}$. The following calculation shows that this is also possible using the approximation algorithm $\mathcal{A}$.

Using $\mathcal{A}$ we compute $\widetilde{N}(\tilde{c}) := \frac{\widetilde{I}(G;\xi)}{\xi^{\tilde{c}}}$ for all $\tilde{c} \in \{1,\ldots,n\}$. We claim that c is the unique $\tilde{c}$ with

$$2^{-(n\ell)^{1-\varepsilon}-1} \leq \widetilde{N}(\tilde{c}) \leq 2^{(n\ell)^{1-\varepsilon}+n+1}. \tag{4.4}$$

Let us prove this claim. As $1 \leq N \leq 2^n$ and by (4.3), we know that

$$\frac{1}{2} \leq \frac{I(G,\xi)}{\xi^c} \leq 2^{n+1}. \tag{4.5}$$

Thus, by (4.2), $\tilde{c} = c$ fulfills (4.4).

On the other hand, when $\tilde{c} \leq c-1$ we have

$$|\widetilde{N}(\tilde{c})| \overset{(4.2),(4.5)}{\geq} 2^{-(n\ell)^{1-\varepsilon}-1}|\xi| \overset{(4.1)}{>} 2^{(n\ell)^{1-\varepsilon}+n+1}.$$

When $\tilde{c} \geq c+1$ we have $|\widetilde{N}(\tilde{c})| < 2^{-(n\ell)^{1-\varepsilon}-1}$ by similar arguments. This shows that no integer $\tilde{c}, \tilde{c} \neq c$, fulfills (4.4). Thus, c can be found in randomized polynomial time using $\mathcal{A}$. □

4.3 Evaluation of the Extended Bivariate Chromatic Polynomial

In this section, we investigate the complexity of evaluating the extended bivariate chromatic polynomial ξ. First, we show that ψ, a slightly modified version of ξ, is

an edge induced subgraph polynomial (ESP) which stems from a graph polynomial that supports edge cloning. This enables us to use our machinery from Section 3.1 of Chapter 3. One immediate consequence are equations that describes the effect of edge thickening on ξ and ψ (Theorem 4.18).

After that, we show that the (ordinary) bivariate chromatic polynomial P is #P-hard to evaluate at almost every point of the plane (Theorem 4.19). We use two ingredients: First, a reduction based on Linial's construction. To this end, we prove (4.10), a generalization of (2.2), in Lemma 4.20. The second ingredient is the fact that the independent set polynomial and the chromatic polynomial, which are #P-hard almost everywhere, are contained in P.

Finally, we show that the extended bivariate chromatic polynomial ξ is #P-hard to evaluate at every point $(x, y, z) \in \mathbb{Q}^3$, except for a set of dimension 2 (Theorem 4.22). In the proof, we reduce from the ordinary bivariate chromatic polynomial. The reduction is the reduction we gain from edge cloning (i.e. edge thickening) using the machinery of Section 3.1.

4.3.1 Definition in Terms of an Edge Induced Subgraph Polynomial

We define ψ as an ESP of a product of two graph polynomials (Definition 4.13). Then we show that these graph polynomials support edge cloning (Proposition 4.14 and 4.15) and that ψ is closely related to the extended bivariate chromatic polynomial (Proposition 4.16, Lemma 4.17). Finally, we state point-to-point reductions for ψ and ξ (Theorem 4.18).

Definition 4.13. *Let z be a variable. For a graph $G = (V, E)$, define the following auxiliary function, which obviously is a graph polynomial:*

$$h(G; z) = \sum_{A \subseteq E} \begin{cases} z^{k_{\mathrm{cov}}(A)} & \text{if } V(A) \cap V(E \setminus A) = \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

Slightly abusing notation, we write $x^{k(G)}$ to denote the function that maps every graph G to $x^{k(G)}$. (This function is a graph polynomial in the variable x.) The product of two functions is defined pointwise.

We define

$$\psi = \mathrm{ESP}_{\mathbf{y}}(x^{k(G)} \cdot h).$$

Proposition 4.14. *$x^{k(G)}$ supports edge cloning.*

Proof. Adding an edge between two vertices that are already linked by an edge does not change the connectivity of graph. □

Proposition 4.15. *h supports edge cloning.*

Proof. Let $G = (V, E)$ be a graph and $a \in E$ an edge between vertices $u, v \in V$. Consider a subset $A \subseteq E(\kappa_a G)$. We distinguish two cases.

1. $a, a' \notin A$ or $a, a' \in A$. Then the summand for A in $h(\kappa_a G)$ corresponds to the summand for $A \setminus \{a'\}$ in $h(G)$.

2. Exactly one of a, a' is in A. Then $\{u, v\} \subseteq V(A) \cap V(E(\kappa_a G) \setminus A)$, thus the summand for A in $h(\kappa_a G)$ is zero.

We have shown that, except for zero summands, the summands of $h(\kappa_a G)$ equal the summands of $h(G)$. Thus, $h(\kappa_a G) = h(G)$. □

Proposition 4.16. *Let $G = (V, E)$ be a graph.*

$$\psi(G; x, \mathbf{y}, z) = \sum_{\substack{A \cup B \subseteq E \\ V(A) \cap V(B) = \emptyset}} x^{k(G[A \cup B])} y_{A \cup B} z^{k_{\mathrm{cov}}(B)}. \tag{4.7}$$

Proof. By the definition, we have

$$\begin{aligned}
\psi(G; x, \mathbf{y}, z) &= \sum_{C \subseteq E} y_C x^{k(G[C])} h(G[C]) \\
&= \sum_{C \subseteq E} y_C x^{k(G[C])} \sum_{A \subseteq C} \begin{cases} z^{k_{\mathrm{cov}}(A)} & \text{if } V(A) \cap V(C \setminus A) = \emptyset \\ 0 & \text{otherwise.} \end{cases} \\
&= \sum_{C \subseteq E} y_C x^{k(G[C])} \sum_{\substack{A \subseteq C \\ V(A) \cap V(C \setminus A) = \emptyset}} z^{k_{\mathrm{cov}}(A)}.
\end{aligned}$$

Now we can write $C = A \cup B$ and $B = C \setminus A$ and exchange the roles of A and B. This yields (4.7). □

Lemma 4.17. *We have the polynomial identities*

$$\psi(G; x, y, zx^{-1}y^{-1}) = \xi(G; x, y, z) \quad \textit{and} \quad \xi(G; x, y, zxy) = \psi(G; x, y, z).$$

Proof. Follows easily from (1.11) and Proposition 4.16. □

Theorem 4.18 ([Hof10, Theorem 3])**.** *Let G_k be the k-thickening of G (i.e. the graph obtained from G by replacing each edge e by k different copies of e). Then*

$$\psi(G_k; x, y, z) = \psi(G; x, (1+y)^k - 1, z) \quad \textit{and} \tag{4.8}$$

$$\xi(G_k; x, y, z) = \xi\Big(G; x, (1+y)^k - 1, z\frac{(1+y)^k - 1}{y}\Big). \tag{4.9}$$

Proof. By Propositions 4.14 and 4.15 and Remark 3.13, we can apply Theorem 3.15 on ψ. This yields (4.8). Now (4.9) follows from Proposition 4.16 and Lemma 4.17. □

4.3.2 Hardness of Evaluation

We give the hardness results concerning the evaluation of the ordinary and extended bivariate chromatic polynomial. The following theorem has been proven independently by Ilia Averbouch (Johann A. Makowsky, personal communication, October 2007), but it has not been published.

Theorem 4.19 ([Hof10, Theorem 4])**.** *Let P denote the bivariate chromatic polynomial [DPT03], see Page 18. For every $(x, y) \in \mathbb{Q}$, $y \neq 0$, $(x, y) \notin \{(1, 1), (2, 2)\}$, $P(-; x, y)$ is #P-hard.*

Moreover, if (x_0, y_0) and (x_1, y_1) fulfill the prerequisites of the theorem and additionally y_0 and y_1 are not positive integers[1] and $x_0 \neq y_0$ and $x_1 \neq y_0$, then $P(-; x_0, y_0) \preceq_T P(-; x_1, y_1)$.

Lemma 4.20. *Let $G = (V, E)$ be a graph and G' be the graph obtained from G by inserting a new vertex v' connecting all vertices of G to v'. Then*

$$P(G'; x, y) = yP(G; x - 1, y - 1) + (x - y)P(G; x, y). \tag{4.10}$$

Proof. In the following, the first and the last equality are by (1.8), the third equality is by (2.2).

$$\begin{aligned}
&P(G'; x, y)\\
=&\sum_{\bar{A} \subseteq V(G')} (x - y)^{|\bar{A}|} \chi(G'[A]; y)\\
=&\sum_{\substack{\bar{A} \subseteq V(G')\\ v' \in A}} (x - y)^{|\bar{A}|} \chi(G'[A]; y) + \sum_{\substack{\bar{A} \subseteq V(G')\\ v' \in \bar{A}}} (x - y)^{|\bar{A}|} \chi(G'[A]; y)\\
=&\sum_{\substack{\bar{A} \subseteq V(G')\\ v' \in A}} (x - y)^{|\bar{A}|} y\chi(G[A]; y - 1) + \sum_{\substack{\bar{A} \subseteq V(G')\\ v' \in \bar{A}}} (x - y)^{|\bar{A}|} \chi(G[A]; y)\\
=&y \sum_{\bar{A} \subseteq V(G)} (x - y)^{|\bar{A}|} \chi(G[A]; y - 1) + (x - y) \sum_{\bar{A} \subseteq V(G)} (x - y)^{|\bar{A}|} \chi(G[A]; y)\\
=&yP(G; x - 1, y - 1) + (x - y)P(G; x, y).
\end{aligned}$$

□

Proposition 4.21. *For every $y, d \in \mathbb{Q}$, $y \neq 0$, we have*

$$P(-; 1 + d, 1) \preceq_T P(-; y + d, y).$$

Moreover, if y is not a positive integer, we have $P(-; Y + d, Y) \preceq_T P(-; y + d, y)$, where Y is a variable.

Proof. Assume that we have an algorithm A for $P(-; y + d, y)$. Then we can do the following on input G: Compute G' as in Lemma 4.20. Using A, compute

[1] i.e. are zero, negative integers, or elements of $\mathbb{Q} \setminus \mathbb{Z}$

$P(G'; y+d, y)$ and $P(G; y+d, y)$. By (4.10), this gives us $P(G; y+d-1, y-1)$. Thus, we have reduced evaluation of P at $(y+d-1, y-1)$ to evaluation at $(y+d, y)$.

Let us apply this repeatedly. If y is a positive integer, we can solve the evaluation problem $P(-; 1+d, 1)$ using this reduction just a finite number of times. (This number is independent of $|G|$, the input size.) This completes the proof for positive integers y.

Otherwise, note that $P(G; Y+d, Y)$ is a polynomial in Y of degree at most $2|G|$ (by (1.8) and Proposition 1.6). As $y, y-1, y-2, \ldots$ does not contain 0, we can evaluate $P(G; Y+d, Y)$ at $2|G|+1$ different points using the reduction. The size of the graphs that are constructed during this procedure are bounded by $3|G|$. This allows us to interpolate $P(G; Y+d, Y)$. Thus, we have shown: $P(-; Y+d, Y) \preceq_T P(-; y+d, y)$. The reduction $P(-; 1+d, 1) \preceq^{\mathtt{m}} P(-; Y+d, Y)$ is trivial. □

Proof of Theorem 4.19. Let (x, y) fulfill the assumptions of the theorem. By Proposition 4.21, we have $P(-; x-y+1, 1) \preceq_T P(-; x, y)$. On the line $y = 1$, polynomial P equals the independent set polynomial [DPT03, Corollary 2], see (1.9), which is #P-hard to evaluate everywhere except at 0 (Corollary 4.8). This proves the theorem for $x \neq y$. For $x = y$, P coincides with the chromatic polynomial, which is #P-hard to evaluate everywhere except at 0, 1, and 2 [Lin86, JVW90]. Thus, P is #P-hard to evaluate on the line $x = y$ except at $(0,0)$, $(1,1)$, and $(2,2)$.

For the "moreover" part, observe that we can interpolate along every line $x = y+d$, $d \neq 0$, and, applying Theorem 3.19 on the independent set polynomial, along the line $y = 1$. □

Theorem 4.22 ([Hof10, Theorem 5])**.** *For every $(x, y, z) \in \mathbb{Q}^3$, $x \neq 0$, $z \neq -xy$, $(x, z) \notin \{(1,0), (2,0)\}$, $y \notin \{-2, 0\}$, it is #P-hard to compute $\xi(G; x, y, z)$ from G.*

Moreover, if $(x_0, y_0, z_0), (x_1, y_1, z_1) \in \mathbb{Q}^3$ fulfill the prerequisites of the Theorem and additionally, for $i = 0, 1$,

1. *$y_i \notin \{0, -1, -2\}$,*
2. *$z_i y_i \neq x_i^2$, and*
3. *$z_i y_i \neq k x_i$ for all positive integers k,*

we have $\xi(-; x_0, y_0, z_0) \preceq_T \xi(-; x_1, y_0, z_0)$.

Proof. For $x, y \in \mathbb{Q}$, $x, y \neq 0$ and $(x, y) \notin \{(1,1), (2,2)\}$, the following problem is #P-hard by Theorem 4.19: Given G, compute

$$P(G; x, y) = \xi(G; x, -1, x-y) = \psi\Big(G; x, -1, \frac{y-x}{x}\Big),$$

where the first equality is (1.15) and the second by Lemma 4.17.

Let us argue that, for every fixed $\tilde{y} \in \mathbb{Q} \setminus \{-2, 0\}$, this reduces to compute $\psi(G; x, \tilde{y}, \frac{y-x}{x})$ from G. For $\tilde{y} = -1$, this is trivial. If $\tilde{y} \neq -1$, we just apply Theorem 3.19 on $G \mapsto \psi\big(G; x, \mathbf{y}, \frac{y-x}{x}\big) \in \mathbb{Q}[\mathbf{y}]$, which is an ESP as it is a specialization of ψ.

Now we use

$$\psi\Big(G; x, \tilde{y}, \frac{y-x}{x}\Big) = \xi(G; x, \tilde{y}, (y-x)\tilde{y}),$$

which holds by Lemma 4.17. Finally, an easy calculation converts the conditions on $x, \tilde{y}, y$ into conditions on x, y, z and yields the theorem.

For the "moreover" part observe that, for a variable Y, we have $\psi(-; x, Y, z) \preceq_T \psi(-; x, \tilde{y}, z)$ if $\tilde{y} \notin \{0, -1, -2\}$ (Theorem 3.19). The rest follows from (the proof of) Theorem 4.19. □

4.4 Open Problems

It would be nice to know the complexity of evaluating the interlace and extended bivariate chromatic polynomial at the points where we could not resolve it.

5 Exponential Time Hardness

Our main result of this chapter is that evaluation of the independent set polynomial needs almost exponential time, unless a counting version of the exponential time hypothesis fails. More precisely, the independent set polynomial of an n-vertex graph can not be evaluated in time $2^{o(n/\log^3 n)}$ at (almost) *any* point x, unless counting SAT can be solved in time $2^{o(n)}$ (Corollary 5.4).

As a first step, we give a reduction from a counting version of SAT to counting independent sets (Lemma 5.1 and Lemma 5.2). This reduction preserves solvability in subexponential time. This shows: If the independent sets of size $n/3$ in every n-vertex graph can be counted in subexponential time, the satisfying assignments of any CNF formula can be counted in subexponential time as well.

The second step is a reduction for the interlace polynomial $\bar{q}(G; u, x)$ along the x-axis that *almost* preserves solvability in subexponential time (Theorem 5.3). This reduction is based on a combination of vertex cloning and addition of paths, which we have introduced in Chapter 3.

5.1 Reduction from SAT to Independent Sets

A general goal of this work is to provide evidence that evaluating graph polynomials is computationally intractable at most of the points. Our general approach is to reduce evaluation at "hard" points to points of unknown complexity. Thus, when considering exponential time hardness, the first thing we need is a particular point where it requires exponential time to evaluate the interlace polynomial. Of course, we can not prove unconditional exponential lower bounds in the Turing machine model; when we, in what follows, say that a problem requires exponential time, we assume #ETH (Page 41).

A good candidate for an "exponential hard point" is $\bar{q}(-; 0, 1)$, as this counts independent sets. It is known that counting independent sets is #P-hard, even under certain restrictions on the input graph [Vad01]. However, exponential time hardness has not been considered yet. Thus, we have to provide a proof that counting SAT, the base problem for exponential time hardness in our setting, reduces to counting independent sets. The usual reductions in the literature do not yield such a reduction: Valiant [Val79b] reduces from SAT to vertex cover via the permanent, perfect matchings and prime implicants. The steps increase the size of the instance more than linearly[1]. Thus, the reduction does not preserve subexponential time.

[1] For instance, the step from perfect matchings to prime implicants includes transforming a $\Theta(n)$ vertex graph into a $\Theta(n^2)$ vertex graph.

The usual reductions for the decision versions of the problems [Kar72], [Pap94, Theorem 9.4] do not preserve the number of solutions, which renders them unusable for counting problems. Thus, we devise a simple reduction from some variant of SAT to independent sets that increases the instance size only linearly and preserves the number of solutions.

Let us first define the counting problems we are considering.

Name: #X3SAT
Input: Boolean formula $\varphi = C_1 \wedge \ldots \wedge C_m$ where each clause C_i is a disjunction of two or three literals over variables $x_1, \ldots, x_n$
Output: Number of assignments for φ such that in every clause exactly one literal is satisfied

Name: #IS
Input: Graph $G = (V, E)$ with $|V| = n$, positive integer k
Output: Number of independents sets $A \subseteq V$ of size k

Lemma 5.1. *#X3SAT requires time* $\exp(\Omega(m))$ *unless #ETH fails.*

Proof. Schaefer [Sch78, Lemma 3.5] gives the following construction. For a clause $C = (a \vee b \vee c)$, define $F = (a \vee u_1 \vee u_4)(b \vee u_2 \vee u_4)(u_1 \vee u_2 \vee u_5)(u_3 \vee u_4 \vee u_6)(c \vee u_3)$.

It is not hard to check that every assignment that satisfies C in the usual sense corresponds to exactly one assignment that satisfies F in the sense of #X3SAT.

Using this construction, we reduce #3SAT to #X3SAT. If an instance φ for #3SAT is given, we construct an instance φ' for #X3SAT by applying the above construction for every clause in φ, each time using "fresh" variables $u_1, \ldots, u_6$. Now an algorithm with running time $\exp(cm)$ for #X3SAT implies an algorithm with running time $\exp(5cm)$ for #3SAT. By Theorem 1.40, this contradicts the #ETH. □

Lemma 5.2. *#IS requires time* $\exp(\Omega(n))$ *unless #ETH fails.*

Proof. We reduce from #X3SAT. Let $\varphi = C_1 \wedge \ldots \wedge C_m$ be an instance for #X3SAT, i.e. a 3-CNF formula with m clauses in n variables. We assume that every variable appears in φ, otherwise a factor of 2^r is introduced in the following reduction, where r is the number of variables that do not appear in φ. Furthermore, we assume that no literal appears twice in a clause and that, if a literal ℓ appears in a clause, its negation $\neg\ell$ does not appear in the same clause. The construction in Lemma 5.1 complies with these assumptions. Therefore, we do not lose generality.

For each clause $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ of φ, we construct a triangle T_i whose vertices $v_{i,1}, v_{i,2}, v_{i,3}$ are labeled by $\ell(v_{i,j}) = \ell_{i,j}$, $1 \leq j \leq 3$, the literals of C_i. In this way, we obtain the vertex set $V = \{v_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq 3\}$ for the #IS instance G. Besides the triangle edges, we add the following edges to G: For each pair $\{u, v\}$ of vertices, where $\ell(u) = \ell(v)$ or $\ell(u) = \neg\ell(v)$, let u_2, u_3 be the other two vertices in u's triangle and v_2, v_3 be the other two vertices in v's triangle. If $\ell(u) = \ell(v)$, we

connect u to v_2 and v_3, and we connect v to u_2 and u_3. If $\ell(u) = \neg\ell(v)$, we connect u and v, and we connect every vertex of $\{u_2, u_3\}$ to every vertex of $\{v_2, v_3\}$.

In the rest of the proof we argue that the number of satisfying assignment for φ (i.e. assignments such that in each clause exactly on literal evaluates to true) equals the number of independent sets A of G with $|A| = m$.

Let a be a satisfying assignment for φ. We map a to the following independent set A of size m in G. For each triangle choose the vertex that corresponds to the literal that is evaluated to true by a. Let us argue that A is in fact independent considering two vertices u and v from A.

- As we choose only one vertex from every triangle, there can not be a triangle edge between u and v.
- If $\ell(u) = \ell(v')$ for some v' that belongs to the same triangle as v, then v's clause has two literals evaluating to true: $\ell(v)$ and $\ell(v') = \ell(u)$. This contradicts the fact that a is a satisfying assignment.
- By the construction of A, we can not have $\ell(u) = \neg\ell(v)$.
- If $\ell(u') = \neg\ell(v')$ for some vertex $u' \neq u$ from u's triangle and some vertex $v' \neq v$ from v's triangle, either u's or v's clause has two literals evaluating to true ($\ell(u)$ and $\ell(u')$, or $\ell(v)$ and $\ell(v')$).

Thus, there is no edge between u and v.

If we change $a(x_i)$ for any variable x_i, our mapping produces a different independent set. (Every triangle where x_i or $\neg x_i$ appears contributes a different vertex to A.) Thus, our mapping is injective.

Finally, we argue that every independent set $A \subseteq V$ of size m corresponds to one satisfying assignment a. We set

$$a(x_i) = \begin{cases} 0 & \text{if } \big(\exists u \in A : \ell(u) = \neg x_i\big) \ \vee \ \big(\exists u \in V \setminus A : \ell(u) = x_i\big), \\ 1 & \text{if } \big(\exists u \in A : \ell(u) = x_i\big) \ \vee \ \big(\exists u \in V \setminus A : \ell(u) = \neg x_i\big). \end{cases}$$

If we show that a is well-defined, it is a satisfying assignment as A contains exactly one vertex from every triangle and the literal of this vertex evaluates to true under a.

To show that a is well-defined, first consider a vertex $u \in A$.

- Assume that there is $u' \in A$ with $\ell(u) = \neg\ell(u')$. Then, by construction of G, there is an edge uv. This contradicts the fact that A is independent.
- Assume that there is $u' \in V \setminus A$ with $\ell(u) = \ell(u')$. Vertex u' is not from u's triangle. Let w be the vertex from u's triangle with $w \in A$. By construction of G, there is an edge uw – a contradiction.

Finally, consider a vertex $u \in V \setminus A$.

- We have already ruled out the existence of a vertex $u' \in A$ with $\ell(u) = \ell(u')$.

- Assume that there is $u' \in V \setminus A$ with $\ell(u) = \neg\ell(u')$. By construction of G, every vertex $w \neq u$ from u's triangle is connected to every vertex $w' \neq u'$ from u''s triangle. Thus, A has no vertex from u's triangle or no vertex from u''s triangle. A contradiction. □

5.2 Interpolation of the Interlace Polynomial

Given a graph G with n vertices and some $u \in \mathbb{Q}$, we would like to interpolate $\bar{q}(G; u, X)$, where X is a variable. The degree of this polynomial is at most n, thus we need to know $\bar{q}(G; u, x)$ for $n+1$ different values of x. As usual, our approach is to modify G in $n+1$ different ways to obtain $n+1$ different graphs G_0, ..., G_n. Then we evaluate $\bar{q}(G_0; u, x)$, $\bar{q}(G_1; u, x)$, ..., $\bar{q}(G_n; u, x)$. We will prove that $\bar{q}(G_i; u, x) = p_i \bar{q}(G; u, x_i)$ for $n+1$ easy to compute x_i and p_i, where all p_i are nonzero and $x_i \neq x_j$ for all $i \neq j$. This will enable us to interpolate $\bar{q}(G; u, X)$.

If a modified graph is c times larger than G, we lose a factor of c in the reduction, i.e. a 2^n running time lower bound for evaluating the graph polynomial at a particular point implies only a $2^{n/c}$ lower bound for the interpolated points. Thus, we can not afford vertex cloning (Chapter 3), where the size of the modified graphs are $2n, 3n, \ldots, n^2$. To overcome this problem, we transfer a technique of Dell, Husfeldt, and Wahlén [DHW10], which they developed for the Tutte polynomial to establish a similar reduction: We clone every vertex of the n vertex graph G $O(\log n)$ times and use n different ways to add paths of different (but at most $O(\log^2 n)$) length at the different clones[2]. Eventually, this will lead to the following result.

Theorem 5.3. *Let $u \in \mathbb{Q}$, $u^2 \neq 1$, and assume that there is an $x_0 \in \mathbb{Q}$ such that (u, x_0) is nondegenerate for path reduction, $\frac{1}{4} + x_0(1 + x_0 u^2) > 0$, and the interlace polynomial $\bar{q}$ of every n-vertex graph G can be evaluated at (u, x_0) in time $2^{o(n/\log^3 n)}$.*

Then, for every n-vertex graph G, the X-coefficients of the interlace polynomial $\bar{q}(G; u, X)$ can be computed in time $2^{o(n)}$. In particular, the interlace polynomial $\bar{q}(G; u, x_1)$ can be evaluated in this time for every $x_1 \in \mathbb{Q}$.

Corollary 5.4. *Let $x \in \mathbb{Q}$, $x \neq 0$. Every algorithm that, on input $G = (V, E)$, $n = |V|$, evaluates*

$$\sum_{\substack{A \subseteq V \\ A \text{ independent set}}} x^{|A|}$$

has worst-case running time $2^{\Omega(n/\log^3 n)}$ unless #ETH fails.

Proof. The independent set polynomial is the interlace polynomial $\bar{q}$ at $u = 0$. For this u, the values $x = 0$ and $x = -1/4$ are the only x that are degenerate for path

[2] Dell et al. use $m+1$ different generalized Theta graphs of size $O(\log^3 m)$, by which they substitute the m edges of the graph.

reduction (cf. Lemma 3.30). For $x > -1/4$, the corollary follows from Theorem 5.3 and Lemma 5.2.

Let us now consider $x < -2$. Then we have $|1+x| > 1$, which implies $(1+x)^2-1 > 0$. On input graph $G = (V, E)$, we have $I(\kappa_V^1 G; x) = I(G; (1+x)^2 - 1)$ by (3.4). Graph $\kappa_V^1 G$ has $2|V|$ vertices. This establishes a reduction from $I(-; (1+x)^2 - 1)$ to $I(-; x)$, where the instance size increases only by a constant factor. As $(1+x)^2 - 1 > 0$, we have reduced from an evaluation point where we have already proved the (conditional) lower bound of the lemma. Thus, the same bound, which is immune to constant factors in the input size, holds for $I(-; x)$.

Let us consider $x \in (-2, 0) \setminus \{-1\}$. We have $|x+1| < 1$ and $|x+1| \neq 0$. In a similar way as we just used a clone, we can use the comb reduction, (3.10): Let k be a positive even integer such that $k > \frac{\log(-2x)}{\log|x+1|}$. Then we have $y := \frac{x}{(1+x)^k} < -2$. On input graph $G = (V, E)$, we can construct G_k as in Theorem 3.24, and we have $I(G_k; x) = (1+x)^k I(G; y)$. As k does not depend on $n = |V|$, $|V(G_k)| = O(|V|)$. Thus, we have reduced from $y < -2$, an evaluation point where we have already proved the lower bound, to evaluation at x.

To handle $x = -2$ and $x = -1$, use the construction from Proposition 3.28. □

The rest of this chapter is devoted to the proof of Theorem 5.3, which is quite technical. The general idea is similar to Dell et al. [DHW10, Lemma 4, Theorem 3(ii)].

Definition 5.5. *Let S be a finite set of positive integers and $G = (V, E)$ be a graph. We define the S-*clone *$G_S = (V_S, E_S)$ of G as follows:*

- *For every vertex $a \in V$, there are $|S|$ vertices $a(|S|) := \{a_1, \ldots, a_{|S|}\}$ in V_S.*
- *For every edge $uv \in E$, there are edges in E_S that connect every edge in $u(|S|)$ to every edge in $v(|S|)$.*
- *Let $S = \{s_1, \ldots, s_\ell\}$. For every vertex $a \in V$, we add a path of length s_i to a_i, the ith clone of a. Formally: For every i, $1 \leq i \leq |S|$, and every $a \in V$ there are vertices $a_{i,1}, \ldots, a_{i,s_i}$ in V_S and edges $a_i a_{i,1}, a_{i,1}a_{i,2}, \ldots, a_{i,s_i-1}a_{i,s_i}$ in E_S.*
- *There are no other vertices and no other edges in G_S but the ones defined by the preceding conditions.*

Definition 5.6. *Let S be a set of numbers. Then we define $\|S\| = \sum_{s \in S} s$.*

Remark 5.7. *The S-clone G_S of a graph $G = (V, E)$ has $|V|(\|S\| + |S|)$ vertices.*

Lemma 5.8. *Let $G = (V, E)$ be a graph and $(u, x) \in \mathbb{C}^2$ nondegenerate for path reduction. Let S be a finite set of positive integers and ${\lambda_1}^{s+2} \neq {\lambda_2}^{s+2}$ for all $s \in S$, where we use the notation of Theorem 3.31. Then we have*

$$\bar{q}(G_S; u, x) = (\prod_{s \in S} C_s)^{|V|} \bar{q}(G; u, x(S)),$$

where

$$x(S) = -1 + \prod_{s \in S}(1 + \frac{B_s}{C_s})$$

and B_s and C_s are defined as in Theorem 3.31.

Proof. Follows from (3.3) and Theorem 3.31. □

Lemma 5.9. *Assume that $(u, x) \in \mathbb{C}^2$ is nondegenerate for path reduction and $\frac{1}{4} + x(1+xu^2) > 0$. Then there are sets $S_0, S_1, \ldots, S_n$ of positive integers, constructible in time $\mathsf{poly}(n)$, such that*

1. *$x(S_i) \neq x(S_j)$ for all $i \neq j$ and*
2. *$\|S_i\| \in O(\log^3 n)$ and $|S_i| \in O(\log n)$ for all i, $0 \leq i \leq n$.*

Proof. We use the notation from Theorem 3.31 and assume $\lambda_1 > \lambda_2$.

As $\left|\frac{\lambda_1}{\lambda_2}\right|^k \to \infty$ for $k \to \infty$, there is a positive integer s_0 such that

$$\left(\frac{\lambda_1}{\lambda_2}\right)^s \notin \left\{\left(\frac{\lambda_2}{\lambda_1}\right)^2, \frac{\lambda_2(x+\lambda_2)}{\lambda_1(x+\lambda_1)}\right\} \quad \forall s \geq s_0.$$

Thus, for every i, $0 \leq i \leq n$, the following set fulfills the precondition on S and T in Lemma 5.10:

$$S_i = \{s_0 + \Delta(2j + b_j^{(i)}) \mid 0 \leq j \leq \lfloor \log n \rfloor\},$$

where Δ is a positive integer defined later, $\Delta \in \Theta(\log n)$, and $[b^{(i)}_{\lfloor \log n \rfloor}, \ldots, b_1^{(i)}, b_0^{(i)}]$ is the binary representation of i. Note that this construction is very similar to Dell et al. [DHW10, Lemma 4]. It is important that the elements in these sets have distance at least Δ from each other. The sets are $\mathsf{poly}(n)$ time constructible as s_0 does not depend on n. We have $\|S_i\| \leq (1+\log n)(s_0+(1+2\log n)\Delta)$ and obviously $|S_i| \in O(\log n)$ for all i. Thus, the second statement of the lemma holds.

To prove the first statement, we use Lemma 5.10. Let $1 \leq i < j \leq n$ and $S = S_i \setminus S_j$, $T = S_j \setminus S_i$. Let s_1 be the smallest number in $S \cup T$ and $A_1 = (S \cup T) \setminus \{s_1\}$. Let us prove that $|f(A_1)| > \sum_{\substack{A \subseteq S \cup T \\ A \neq A_1}} |f(A)|$, which yields the statement of the lemma.

Assume without loss of generality that $s_1 \in S$. As (u, x) is nondegenerate, $C_1 := \min\{1, |\lambda_1|, |\lambda_2|, |x+\lambda_1|, |x|, |\lambda_1 - \lambda_2|\}$ is a nonzero constant. As $|S| = |T|$,

$$\begin{aligned} D(S, T, A_1) &= {\lambda_1}^{|T|}(x+\lambda_1)^{|S|-1}(x+\lambda_2) - {\lambda_1}^{|S|-1}\lambda_2(x+\lambda_1)^{|T|} \\ &= {\lambda_1}^{|S|-1}(x+\lambda_1)^{|S|-1}\big(\lambda_1(x+\lambda_2) - \lambda_2(x+\lambda_1)\big) \\ &= {\lambda_1}^{|S|-1}(x+\lambda_1)^{|S|-1}x(\lambda_1 - \lambda_2), \end{aligned}$$

and we have

$$|f(A_1)| \geq |\lambda_1|^{\|S \cup T\| - s_1}|\lambda_2|^{s_1} C_1^{7|S|}. \tag{5.1}$$

If $A = \emptyset$ or $A = S \cup T$, we have $D(A) = 0$, which implies $f(A) = 0$. For every $A \subseteq S \cup T$, $A \neq \emptyset$, $A \neq S \cup T$, $A \neq A_1$, we have $\|A\| \leq \|S \cup T\| - s_1 - \Delta$. Thus,

$$|f(A)| \leq |\lambda_1|^{\|S\cup T\|-s_1-\Delta}|\lambda_2|^{s_1+\Delta}C_2^{7|S|}, \tag{5.2}$$

where $C_2 = 2\max\{1, |\lambda_1|, |\lambda_2|, |x+\lambda_1|, |x+\lambda_2|\}$. There are less than $2^{\lfloor \log n \rfloor + 1} \leq 2n^2$ such A. Combining this with (5.1) and (5.2), it follows that we have proven the lemma if we ensure

$$\Big|\frac{\lambda_1}{\lambda_2}\Big|^{\Delta} > \Big(\frac{C_2}{C_1}\Big)^{7|S|} 2n^2.$$

This holds if

$$\Delta > 7\Big((\log n + 1)\log\frac{C_2}{C_1} + 2\log n + 1\Big)\Big/\log\frac{\lambda_1}{|\lambda_2|}.$$

As C_1, C_2, λ_1, λ_2 do not depend on n, we can choose $\Delta \in \Theta(\log n)$. □

Lemma 5.10. *Let S and T be two sets of positive integers. Let also $(u, x) \in \mathbb{C}^2$ be nondegenerate for path reduction and, for all $s \in S \cup T$,*

$$\left(\frac{\lambda_1}{\lambda_2}\right)^{s+2} \neq 1 \quad \textit{and} \tag{5.3}$$

$$\left(\frac{\lambda_1}{\lambda_2}\right)^{s+1} \neq \frac{x+\lambda_2}{x+\lambda_1}, \tag{5.4}$$

where λ_1, λ_2 are defined as in Theorem 3.31. Then we have $x(S) = x(T)$ iff

$$\sum_{A \subseteq S \triangle T} f(A) = 0,$$

where

$$\begin{aligned} f(A) &= {\lambda_1}^{\|A\|}{\lambda_2}^{\|(S\triangle T)\setminus A\|}(-\lambda_1)^{|A|}{\lambda_2}^{|(S\triangle T)\setminus A|} \cdot D(S \setminus T, T \setminus S, A), \\ D(S,T,A) &= c(S,T,A\cap S, A\cap T) - c(T,S,A\cap T, A\cap S), \\ c(S,T,S_0,T_0) &= {\lambda_1}^{|T_0|}{\lambda_2}^{|T\setminus T_0|}(x+\lambda_1)^{|S_0|}(x+\lambda_2)^{|S\setminus S_0|}. \end{aligned}$$

Proof. Let $\widetilde{S} = S \setminus T$ and $\widetilde{T} = T \setminus S$. We have $x(S) = x(T)$ iff $x(S) + 1 = x(T) + 1$. Condition (5.4) ensures $1 + \frac{B_s}{C_s} \neq 0$ for all $s \in S \cup T$. Thus, $x(S \cap T) + 1 \neq 0$, and $x(S) = x(T)$ iff $x(\widetilde{S}) + 1 = x(\widetilde{T}) + 1$. This is equivalent to $Y(\widetilde{S}, \widetilde{T}) = Y(\widetilde{T}, \widetilde{S})$, where $Y(S,T) = \prod_{s\in S}(C_s + B_s)\prod_{t\in T} C_t$. For sets of integers $M \subseteq N$, let us define

$$\begin{aligned} B(N,M) &= {\lambda_1}^{\|M\|}(-\lambda_1)^{|M|}{\lambda_2}^{\|N\setminus M\|}{\lambda_2}^{|N\setminus M|} \quad \text{and} \\ C(N,M) &= {\lambda_1}^{\|M\|}{\lambda_1}^{2|M|}(-1)^{|M|}{\lambda_2}^{\|N\setminus M\|}{\lambda_2}^{2|N\setminus M|}. \end{aligned}$$

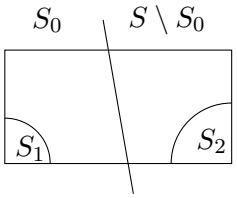

Figure 5.1: Partition of S.

Using this notation, it is

$$\begin{aligned}
Y(S,T) =& \prod_{s\in S}(B_s + C_s)\prod_{t\in T} C_t \\
=& \sum_{S_0\subseteq S}\prod_{s\in S_0} B_s \prod_{s\in S\setminus S_0} C_s \prod_{t\in T} C_t \\
=&(\lambda_2-\lambda_1)^{-|S|-|T|}\sum_{S_0\subseteq S} x^{|S_0|}\sum_{S_1\subseteq S_0} B(S_0,S_1)\sum_{S_2\subseteq S\setminus S_0} C(S\setminus S_0, S_2) \\
&\sum_{T_0\subseteq T} C(T,T_0).
\end{aligned}$$

We want to collect the terms ${\lambda_1}^{\|M\|}$ and ${\lambda_2}^{\|N\setminus M\|}$ in one place. Thus, we change the order in which S is split into subsets S_0, S_1, S_2 (cf. Figure 5.1) such that we first choose $S_{12} := S_1 \cup S_2 \subseteq S$, then $S_1 \subseteq S_{12}$ (which implies $S_2 = S_{12} \setminus S_1$), and finally S_0 as $S_1 \subseteq S_0 \subseteq S \setminus S_2$. Now we can write

$$\begin{aligned}
Y(S,T) = (\lambda_2-\lambda_1)^{-|S|-|T|}\sum_{S_{12}\subseteq S}\sum_{T_0\subseteq T} &{\lambda_1}^{\|S_{12}\|+\|T_0\|}{\lambda_2}^{\|S\setminus S_{12}\|+\|T\setminus T_0\|} \\
&(-\lambda_1)^{|S_{12}|+|T_0|}{\lambda_2}^{|S\setminus S_{12}|+|T\setminus T_0|} \\
&c(S,T,S_{12},T_0),
\end{aligned} \tag{5.5}$$

where

$$c(S,T,S_{12},T_0) = {\lambda_1}^{|T_0|}{\lambda_2}^{|T\setminus T_0|}\sum_{\substack{(S_1,S_2)\\ S_1\dot\cup S_2 = S_{12}}}\sum_{\substack{S_0\\ S_1\subseteq S_0\subseteq S\setminus S_2}} x^{|S_0|}{\lambda_1}^{|S_2|}{\lambda_2}^{|(S\setminus S_0)\setminus S_2|}.$$

Note that (5.5) as symmetric in S and T, except for the term $c(S,T,S_{12},T_0)$. Let us analyze this non-symmetrical term. We write $S_0 = S_1 \dot\cup \widetilde{S}_0$.

$$\begin{aligned}
c(S,T,S_{12},T_0) =& {\lambda_1}^{|T_0|}{\lambda_2}^{|T\setminus T_0|}\sum_{S_1\dot\cup S_2=S_{12}} {\lambda_1}^{|S_2|}x^{|S_1|}\sum_{\widetilde{S}_0\subseteq S\setminus S_{12}} x^{|\widetilde{S}_0|}{\lambda_2}^{|(S\setminus S_{12})\setminus\widetilde{S}_0|} \\
=& {\lambda_1}^{|T_0|}{\lambda_2}^{|T\setminus T_0|}\sum_{S_1\dot\cup S_2=S_{12}} {\lambda_1}^{|S_2|}x^{|S_1|}(x+\lambda_2)^{|S\setminus S_{12}|} \\
=& {\lambda_1}^{|T_0|}{\lambda_2}^{|T\setminus T_0|}(x+\lambda_1)^{|S_{12}|}(x+\lambda_2)^{|S\setminus S_{12}|}.
\end{aligned}$$

This implies the statement of the lemma. □

Proof of Theorem 5.3. On input $G = (V, E)$ with $|V| = n$, do the following. Construct G_{S_0}, G_{S_1}, ..., G_{S_n} with S_i from Lemma 5.9. Every G_{S_i} can be constructed in time polynomial in $|G_{S_i}|$, which is $\mathsf{poly}(n)$ by Remark 5.7 and by 2. of Lemma 5.9. Thus, the whole construction can be performed in time $\mathsf{poly}(n)$.

Again by 2. of Lemma 5.9, there is some $c' > 1$ such that all G_{S_i} have $\leq c' n \log^3 n$ vertices. Evaluate $\bar{q}(G_{S_0}; u, x)$, $\bar{q}(G_{S_1}; u, x)$, ..., $\bar{q}(G_{S_n}; u, x)$. By the assumption of the theorem, one such evaluation can be performed in time

$$2^{c\frac{c' n \log^3 n}{(\log(c' n \log^3 n))^3}} = 2^{\frac{cc' n \log^3 n}{(\log c' + \log n + 3 \log\log n)^3}} \leq 2^{\frac{cc' n \log^3 n}{(\log n)^3}} = 2^{cc' n}$$

for every $c > 0$.

Using Lemma 5.8 we can compute $\bar{q}(G; u, x(S_0))$, $\bar{q}(G; u, x(S_1), \ldots, \bar{q}(G; u, x(S_n))$ from the already computed $\bar{q}(G_{S_i}; u, x)$ in time $\mathsf{poly}(n)$.

By 1. of Lemma 5.9, the $n + 1$ values $x(S_i)$ are pairwise distinct. As $\bar{q}(G; u, X)$ is a polynomial of degree at most n in X, this enables us to interpolate $\bar{q}(G; u, X)$. The overall time needed is $\mathsf{poly}(n) 2^{cc' n} \leq 2^{(cc' + \varepsilon) n}$ for every $\varepsilon > 0$. □

5.3 Open Problems

Corollary 5.4 states that $\bar{q}(-; 0, x)$ needs time $\exp(\Omega(n/\log^3 n))$ for all $x \in \mathbb{Q} \setminus \{0\}$. What about $\bar{q}(-; u, x)$, $u \in \mathbb{Q}$, $u \neq 0$? When we proved that evaluating the interlace polynomial is #P-hard at almost all (u, x) (Theorem 4.7), we used a connection between the interlace polynomial and the Tutte polynomial of planar graphs. Whereas Dell et al. prove an $\exp(\Omega(n/\log^3 n))$ lower bound for evaluating the Tutte polynomial of general graphs (under #ETH), there is no (conditional) lower bound on evaluation of the Tutte polynomial on planar graphs yet[3]. Thus, even though we have an almost subexponential time preserving reduction, we can not generalize Corollary 5.4 to other values of u using the Tutte-interlace connection as we are missing the "hard points" for $u \neq 0$. Providing such hard points (or overcoming this issue in another way) is an interesting problem for future research.

It would also be nice to have a reduction that does not lose the factor $\Theta(\log^3 n)$ in the exponent of the running time.

[3] In fact, it is known that the Tutte polynomial of planar graphs can be evaluated in time $\exp(O(\sqrt{n}))$ [SIT95].

6 Computation of the Extended Bivariate Chromatic Polynomial

In this chapter, we consider the following task: We are given a graph G with n vertices and we want to compute the extended bivariate chromatic polynomial $\xi(G; x, y, z)$, which means that we want to compute the coefficients of this polynomial. We prove that there are algorithms for this problem that run in time $3^n\mathsf{poly}(n)$ and polynomial space, or in time and space $2^n\mathsf{poly}(n)$, respectively (Theorem 6.6).

Let n denote the number of vertices and m the number of edges of a graph G of which we want to compute ξ. It is immediate from (1.11) that $\xi(G; x, y, z)$ can be computed using $3^m \cdot \mathsf{poly}(n)$ operations. Thus, the trivial method to compute ξ has running time exponential in the number of *edges* of the graph. Inspired by the work of Björklund, Husfeldt, Kaski, and Koivisto [BHKK08], we show that $\xi(G; x, y, z)$ can be computed using $3^n \cdot \mathsf{poly}(n)$ operations, i.e. in time exponential in the number of *vertices*. As m can be up to $\Omega(n^2)$ and already a constant factor improvement in the exponent would be valuable, this is a considerable improvement. Basically, our method is: Follow the lines of Björklund et al. [BHKK08] and use (6.3) instead of the Fortuin-Kasteleyn identity.

Definition 6.1. *Let S be a finite set and R be a ring. For $f : 2^S \to R$,*

1. *the* zeta transform *of f is $f\zeta : 2^S \to R$, $f\zeta(Y) = \sum_{X \subseteq Y} f(X)$, and*
2. *the* Möbius transform *of f is $f\mu : 2^S \to R$, $f\mu(X) = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} f(Y)$.*

Remark 6.2. *By their definition, the zeta and the Möbius transform can be computed using 3^n ring operations and space for a constant number of ring elements plus $\mathsf{poly}(n)$ for a counter.*

Lemma 6.3. *(Fast zeta / Möbius transform) The zeta and the Möbius transform can be computed using $O(n2^n)$ ring operations and space for $O(2^n)$ ring elements.*

Proof. See Björklund et al. [BHKK07, Section 2.2], where the idea of the algorithm is attributed to Yates [Yat37]. □

Björklund et al. [BHKK08] use a special case of the following fact. For the reader's convenience, we give a proof.

Lemma 6.4. *Let $f_1, \ldots, f_q : 2^S \to R$. Then*

$$((f_1\zeta \cdot \ldots \cdot f_q\zeta)\mu)(S) = \sum_{U_1,\ldots,U_q} f_1(U_1) \cdots f_q(U_q), \tag{6.1}$$

where the sum is over all $U_1, \ldots, U_q \subseteq S$, $U_1 \cup \ldots \cup U_q = S$.

Proof.

$$
\begin{aligned}
&(f_1\zeta \cdot \ldots \cdot f_q\zeta)\mu(S) \\
&= \sum_{Y\subseteq S} (-1)^{|S\setminus Y|} \prod_{1\le i\le q} \sum_{X\subseteq Y} f_i(Y) \qquad (6.2)\\
&= \sum_{Y\subseteq S} (-1)^{|S\setminus Y|} \sum_{\sigma:[q]\to 2^Y} \prod_{1\le i\le q} f_i(\sigma(i)) \\
&= \sum_{\sigma:[q]\to 2^S} \prod_{1\le i\le q} f_i(\sigma(i)) \sum_{\substack{Y,Y\subseteq S\\ \forall i\in[q]:\sigma(i)\subseteq Y}} (-1)^{|S\setminus Y|} \\
&= \sum_{\sigma:[q]\to 2^S} \prod_{1\le i\le q} f_i(\sigma(i)) \sum_{\substack{Y,Y\subseteq S\\ \sigma(1)\cup\ldots\cup\sigma(q)\subseteq Y}} (-1)^{|S\setminus Y|} \\
&= \sum_{\sigma:[q]\to 2^S} \prod_{1\le i\le q} f_i(\sigma(i)) \begin{cases} 1 & \text{if } \sigma(1)\cup\ldots\cup\sigma(q) = S \\ 0 & \text{otherwise} \end{cases} \\
&= \sum_{U_1\cup\ldots\cup U_q = S} \prod_{1\le i\le q} f_i(U_i). \qquad \square
\end{aligned}
$$

Lemma 6.5. (6.1) *can be evaluated*

1. *using* $3^n\mathsf{poly}(n,q)$ *ring operations, space for* $\mathsf{poly}(q)$ *ring elements, and* $\mathsf{poly}(n)$ *space for a counter; and*

2. *using time and space* $2^n\mathsf{poly}(n,q)$.

Proof. The first statement follows from a direct implementation of (6.2).

For the $2^n\mathsf{poly}(n)$ bound, we can afford to store functions mapping from 2^S. First, we compute $f_1\zeta$, …, $f_q\zeta$ via fast Zeta transform (Lemma 6.3) and store them. Then, we compute the product and store it. Finally, we compute the Möbius transform, again using Lemma 6.3. $\square$

Theorem 6.6. *The extended bivariate chromatic polynomial on a graph with* n *vertices can be computed*

1. *in time* $3^n\mathsf{poly}(n)$ *and space* $\mathsf{poly}(n)$ *and*

2. *in time and space* $2^n\mathsf{poly}(n)$.

Proof. Averbouch, Godlin, and Makowsky give the following characterization of ξ [AGM10, Theorem 6]. Let x and p be positive integers and H be a clique on $x+2p$ vertices that has a self loop at every vertex. Let V_H be partitioned into three sets V_A, V_B, and V_I such that $|V_A| = x$ and $|V_B| = |V_I| = p$. Let $\alpha(v) = 1$ if $v \in V_A \cup V_B$ and $\alpha(v) = -1$ otherwise. Let $\beta(u,v) = 1 + y$ if $u = v \in V_A \cup V_B$

and $\beta(u,v) = 1$ otherwise. Then $\xi(G;x,y,py)$ equals the value $Z_H(G)$ of the homomorphism function with respect to H, i.e.

$$\xi(G;x,y,py) = \sum_{h:V\to V_H} \prod_{v\in V} \alpha(h(v)) \prod_{\{u,v\}\in E} \beta(h(u),h(v)). \tag{6.3}$$

Observe that α is relevant only for vertices that are mapped to V_I. Also, edges whose end vertices are not mapped to one and the same vertex in $V_A \cup V_B$, have no influence. For a mapping $h : V \to V_H$, let us denote the preimages of the x elements of V_A by $U_1,\ldots,U_x$, the preimages of the p elements of V_B by $U_{x+1},\ldots,U_{x+p}$, and the preimages of the p elements of V_I by $U_{x+p+1},\ldots,U_{x+2p}$. $(U_1,\ldots,U_{x+2p})$ is a partition of V_H and uniquely determines h. Thus, (6.3) can be written as

$$\sum_{U_1,\ldots,U_{x+2p}} \prod_{x+p+1\le i\le x+2p} g(U_i) \prod_{1\le i\le x+p} f(U_i), \tag{6.4}$$

where the sum is over all $(x+2p)$-tuples of subsets of vertices of G with $U_i \cap U_j = \emptyset$ for $i \neq j$,

$$f(U) = \prod_{e\in E(G[U])} (1+y),$$

and $g(U) = (-1)^{|U|}$. We compute (6.4) via Lemma 6.5: Just as Björklund et al., we set $\tilde{f}(U) = f(U)w^{|U|}$ and $\tilde{g}(U) = g(U)w^{|U|}$. Using Lemma 6.5 we can evaluate

$$F(w) = \sum_{\substack{U_1,\ldots,U_{x+2p}\\ U_1\cup\ldots\cup U_{x+2p}=V}} \prod_{x+p+1\le i\le x+2p} \tilde{g}(U_i) \prod_{1\le i\le x+p} \tilde{f}(U_i),$$

for arbitrary w. Note that the coefficient of $w^{|V|}$ of $F(w)$ equals (6.4). Thus, it is sufficient to evaluate $F(w)$ for $w = 0,1,\ldots(x+2p)n$ to obtain (6.4) and thus (6.3) for fixed numbers x,y,z.

As the degree of ξ is polynomial in the number of vertices, we can interpolate the coefficients of $\xi(G)$ from a polynomial number of evaluations, which concludes the computation of $\xi(G)$. □

7 Computation of the Interlace Polynomial

In this chapter, we propose some methods to compute the interlace polynomial.

First, we develop means to evaluate Courcelle's multivariate interlace polynomial (Definition 1.15) using dynamic programming on a tree decomposition of the input graph (Sections 7.2-7.5). This yields a parameterized algorithm (Algorithm 2) to evaluate every interlace polynomial we have mentioned in the introduction. The running time of our algorithm is $2^{O(k^2)}n$, where n is the size of the graph and k is the parameter, i.e. the width of a given tree decomposition of the input graph.

In Section 7.6, we discuss some variants of the algorithm: It can also be used as a method to construct an arithmetic circuits that describes (i.e. computes in the sense of algebraic complexity theory) the interlace polynomial. The size of this circuit is $2^{O(k^2)}n$, its depth is $O(\log^2 n)$. Another application is to compute coefficients of the multivariate interlace polynomial, in particular d-truncations. Nicely, our tree decomposition based method also enables us to evaluate the interlace polynomial on graphs of *arbitrary* treewidth asymptotically faster than by direct implementation of Definition 1.15 or Definition 1.14.

Section 7.7 contains algorithms to compute the interlace polynomial $\bar{q}(G; u, x)$ in two special cases. In these cases, we obtain a substantially improved running time: If the graphs do not have self loops and are of bounded maximum degree or if we only want to compute $\bar{q}(G; u, x)$ modulo u^k for a constant k, the interlace polynomial can be computed in time c^n for some $c < 2$.

Before we start, let us fix the notation for tree decompositions.

7.1 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 connected subgraph 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] (cf. also Downey and Fellows [DF99, Section 6.3]).

To evaluate the interlace polynomial we will use *nice* tree decompositions. Note that our definition slightly deviates from the usual one[1]. 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 $(\{X_i\}, T)$, T is a rooted tree with $|X_r| = 0$ for the root r 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 $a \in V \setminus X_j$ with $X_i = X_j \cup \{a\}$.
- Forget: node i has exactly one child j, and there is a vertex $a \in V \setminus X_i$ 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

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

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

7.2 Idea for a Tree Decomposition Based Algorithm

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)

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

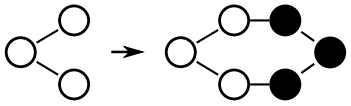

Figure 7.1: Interlace polynomial and rank behavior: Rank over $GF(2)$ of the adjacency matrix increases by 2 (from 2 to 4).

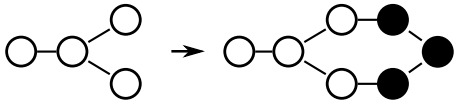

Figure 7.2: Interlace polynomial and rank behavior: Rank over $GF(2)$ of the adjacency matrix increases by 4 (from 2 to 6).

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 7.1. 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 7.2, the *same extension* causes a rank increase by 4.

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

Definition 7.1 (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 $G[V', U]$*.*

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 every 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 7.7) 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 7.3). The resulting matrix M' is symmetric again and has the same rank as M. Furthermore, M' is of a form as in Figure 7.3: 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

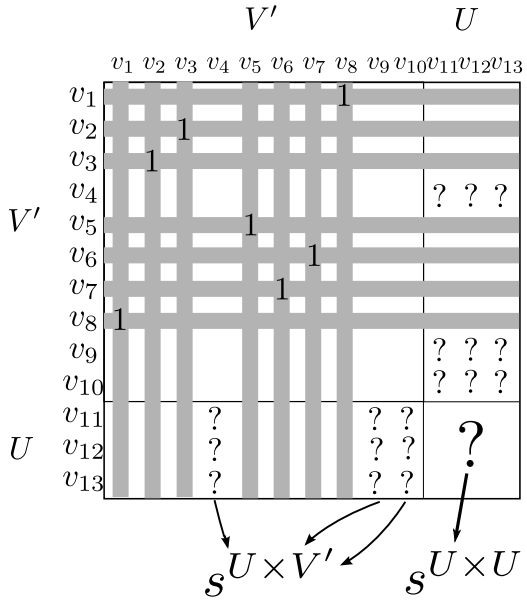

Figure 7.3: Adjacency matrix of $G[V' \cup U]$ after symmetric Gaussian elimination using V'. Empty entries are zero.

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 (rows) that are ruled by some edge or self loop in V' are also empty (i.e. entirely zero) in the $U \times V'$ ($V' \times U$, resp.) 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 $\mathsf{poly}(|U|)$ how the rank of the adjacency matrix of $G[V']$ increases when we add some vertices from U. This motivates the following definition.

Definition 7.2 (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 (7.1) in Section 7.5. 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 7.16, 7.17 and 7.18. These are based on the fact that scenarios are compliant with tree decompositions, which we will prove in Section 7.4 (Lemma 7.8, Lemma 7.10 and Lemma 7.12). An example for the overall procedure of the algorithm is depicted in Figure 7.4.

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 7.3. *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

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

7.3 Symmetric Gaussian Elimination

We want to convert adjacency matrices into matrices of a form as in Figure 7.3 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

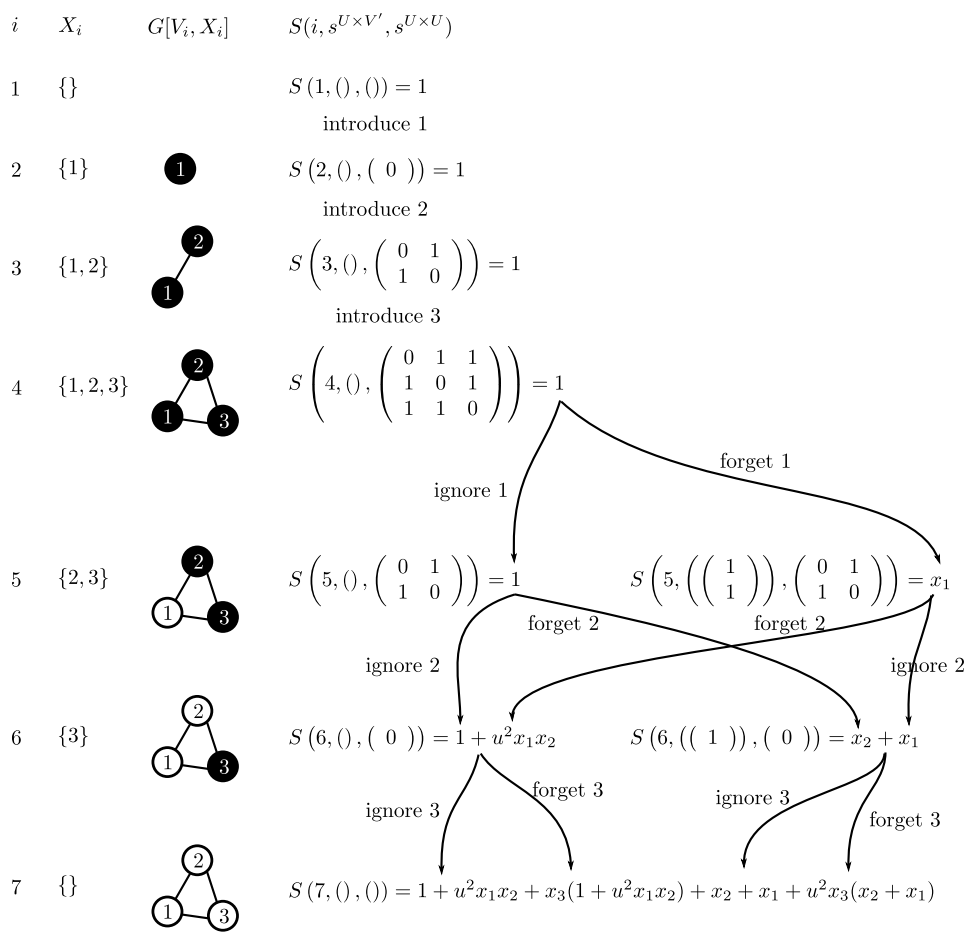

Figure 7.4: Computation of the interlace polynomial $q(G; u, \mathbf{x}) = C(G; y = 0, v = 1)$ of a triangle. For sake of clarity, our illustration ignores parameter D of (7.1), which handles the "self loop toggling feature" of the interlace polynomial $C(G)$.

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. 7.4 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'.

7.3.1 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 by an appropriate modification of M. Arratia et al. observe that edge pivot and local complementation are such appropriate modifications [ABS04b, Lemma 2, Lemma 5]. For our purposes, we want to control the order of the operations on the adjacency matrix. Thus, we do not use edge pivot and local complementation directly, but define a *symmetric Gaussian elimination step* on M using v in the following way:

- If v is an isolated vertex without a self loop, we have situation (1) of Figure 7.5. 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 7.5. 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.

  Note that – up to order of the operations – this is local complementation on v: The adjacency matrix of G^v is $M \rtimes v$ [ABS04b, Proof of Lemma 5].

- 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 7.5. 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 7.3.2. The rank of M equals 2 plus the rank of $M \rtimes vu$ with u- and v-column and u- and v-row deleted. Note that this matrix is also the adjacency matrix of $G^{vu}[V \setminus \{v, u\}]$.

  If u has a self loop we proceed analogously to obtain a matrix with a structure as (4) in Figure 7.5. Then we can eliminate the self loop at u by, say, adding

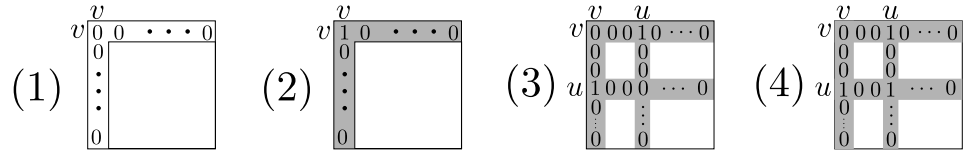

Figure 7.5: Effect of a symmetric Gaussian elimination step. Adjacency matrix with isolated unlooped vertex v (1), adjacency matrix after eliminating with a self loop at v (2), adjacency matrix after eliminating with edge vu (3).

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. 7.5.

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 7.4. *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 7.5. *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 7.5. Let us assume that $x \leq y$ (the case $x > y$ is analogous). The situation is depicted in Figure 7.6. 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. □

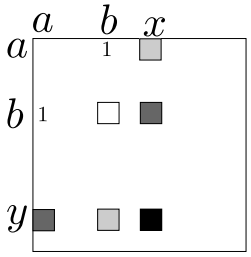

Figure 7.6: During a symmetric Gaussian elimination step using edge ab, entry (y, x) is affected only by the entries at (a, x), (y, b), (y, a), (b, x) and (b, b).

7.3.2 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.

Algorithm 1 Supplying a vertex order.

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

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

Definition 7.6. *Let $V' \subseteq V$ be a set of vertices of a graph $G = (V, E)$ with*

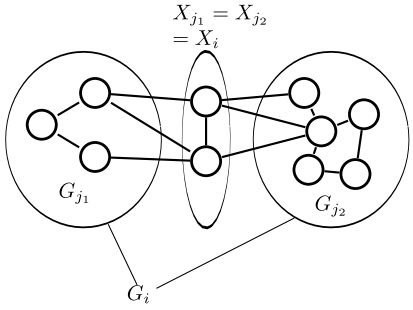

Figure 7.7: Graphs corresponding to a join node i and its child nodes j_1, j_2.

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.

Definition 7.7 (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 $M'_{UV'}$ that is not in the span of the so far collected columns.

7.4 Scenarios and Nice Tree Decompositions

7.4.1 Join Nodes

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 7.7: $G_{j_1} = G[V_{j_1}]$ 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 7.2 and (7.1)) 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 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[2] $G[V']$ 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 7.8 (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, $\operatorname{scen}(G[V_1, U]) = s_1$, and $\operatorname{scen}(G[V_2, U]) = s_2$, then $\operatorname{scen}(G[V_1 \cup V_2, U]) = s_3$. Moreover, s_3 can be computed from s_1, s_2 and $G[U]$ within $\mathsf{poly}(|U|)$ steps.*

Proof. We will apply Definition 7.7 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 in the upper part of Figure 7.8, 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 7.7, symmetric Gaussian elimination using $V_1 \cup V_2$ has to be performed on M to obtain M', which is of the form depicted in the lower part of Figure 7.8 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 column u. Let us argue that this affects neither the $V_1 \times V_2$ submatrix of M nor the $V_2 \times V_1$ submatrix of M. As $v \in V_1$, the V_2-entries in the v-row 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 $V_2 \times V_1$

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

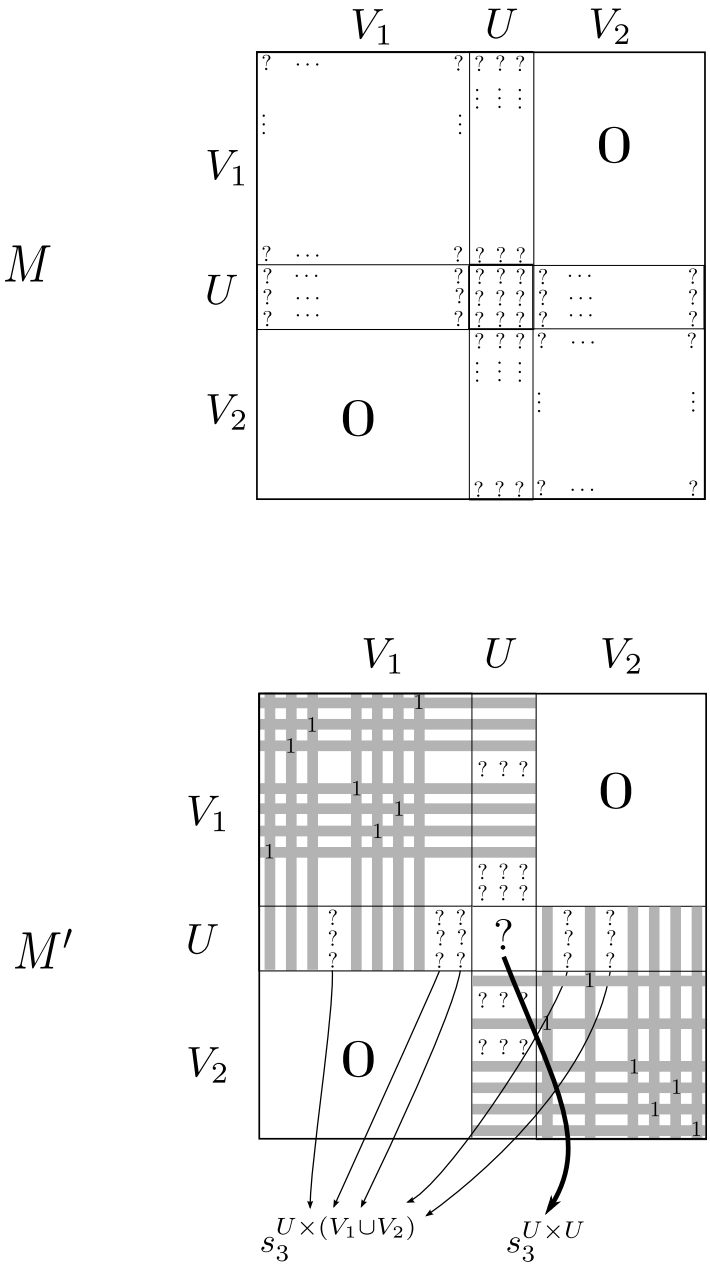

Figure 7.8: Effect of symmetric Gaussian elimination to gain the scenario of $G[V_1 \dot\cup V_2, U]$. Entries with question marks are either 0 or 1. Empty entries are 0.

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 7.6 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 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 7.9. *In the situation of Lemma 7.8, we write* $s_{\text{join}}(s_1, s_2, G[U])$ *for* s_3.

7.4.2 Introduce Nodes

To handle join nodes of the tree decomposition, we proved Lemma 7.8: From the scenario of two extended graphs $G[V_1, U]$ and $G[V_2, U]$ with a common extension

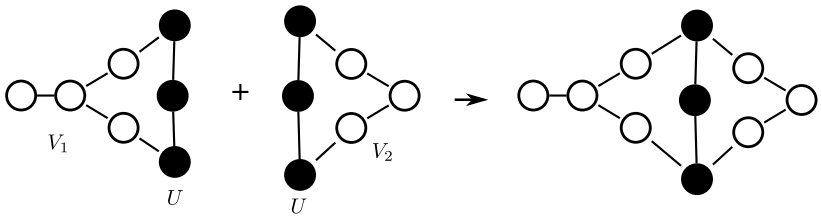

Figure 7.9: Joining the extended graphs $G[V_1, U]$ and $G[V_2, U]$.

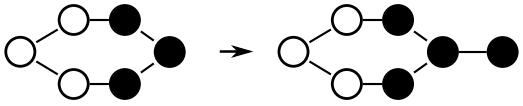

Figure 7.10: Adding a vertex to an extension.

U, we can compute the scenario of the joined extended graph $G[V_1 \cup V_2, U]$ (cf. Figure 7.9). To handle introduce and forget nodes as well, we prove two more lemmas (cf. Figure 7.10, Figure 7.11).

Lemma 7.10 (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 $\widetilde{U} = U \cup \{u\}$ such that the following holds: If $G[V']$ may be extended by $\widetilde{U}$ according to G, u is not connected to V' in G, and $\operatorname{scen}(G[V', U]) = s$, then $\operatorname{scen}(G[V', \widetilde{U}]) = \tilde{s}$. Moreover, $\tilde{s}$ can be computed from s and $G[\widetilde{U}]$ in $\mathsf{poly}(|U|)$ steps.*

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

Definition 7.11. *In the situation of Lemma 7.10, we write $s_{\text{introduce}}(s, u, G[\widetilde{U}])$ for $\tilde{s}$.*

7.4.3 Forget Nodes

Except for isolated vertices without self loops, every vertex has an effect on the rank of the adjacency matrix [ABS04b, Lemma 2, Lemma 5] (cf. Section 7.3.1). The following lemma states that this effect can be extracted from the scenario.

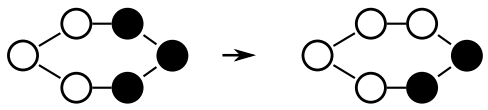

Figure 7.11: Transforming an extending vertex into a normal vertex.

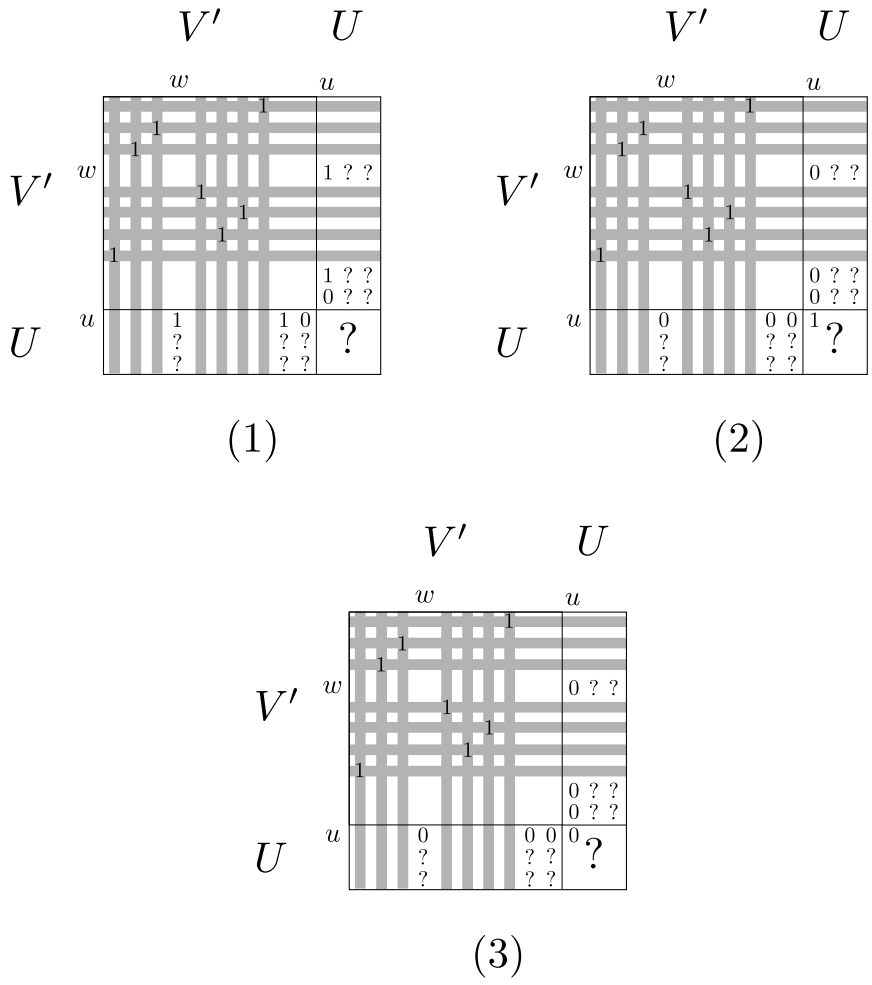

Figure 7.12: Cases when "forgetting" an extension vertex u. Entries with question marks are either 0 or 1. Empty entries are 0.

Lemma 7.12 (Forget vertex)**.** *Let $G = (V, E)$ be a graph, $u \in U \subseteq V$, $\widetilde{U} = U \setminus \{u\}$, $\widetilde{V} = V' \cup \{u\}$, and s a scenario of U. Then there is a unique scenario $\tilde{s}$ of $\widetilde{U}$ and $r \in \{0, 1, 2\}$, $n \in \{1, 0, -1\}$ 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 $\operatorname{scen}(G[V', U]) = s$, then $\operatorname{scen}(G[\widetilde{V}, \widetilde{U}]) = \tilde{s}$ and the rank (nullity) of the adjacency matrix of $G[\widetilde{V}]$ 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 $n = 1 - r$.*

Proof. Consider the situation after symmetric Gaussian elimination on $G[V' \cup U] = G[\widetilde{V} \cup \widetilde{U}]$ using V' (Figure 7.12). 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 $G[\widetilde{V} \cup \widetilde{U}]$ using $\widetilde{V}$ 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 7.12 (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 $\bar{s}^{\widetilde{U}\times\widetilde{V}}$. 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 $\bar{s}^{\widetilde{U}\times\widetilde{V}}$. If it is, let $\tilde{s}^{\widetilde{U}\times\widetilde{V}} = \bar{s}^{\widetilde{U}\times\widetilde{V}}$, otherwise let $\tilde{s}^{\widetilde{U}\times\widetilde{V}} = \bar{s}^{\widetilde{U}\times\widetilde{V}} \cup \{a\}$. Let $\tilde{s}^{\widetilde{U}\times\widetilde{U}}$ be $s^{U\times U}$ with first column and first row deleted. We have $r = 0$ and $n = 1$.

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 $r = 1$ and $n = 0$.

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 7.12 (1)). Compare symmetric Gaussian elimination on $G[V' \cup U]$ using V' (which is performed to obtain s) to symmetric Gaussian elimination on $G[\widetilde{V} \cup \widetilde{U}]$ using $\widetilde{V}$ (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 $\widetilde{V}$, 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 $\widetilde{V}$. However, we will prove in Lemma 7.15 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 $\tilde{s}^{\widetilde{U}\times\widetilde{V}} = \bar{s}^{U\times V'}$, and $\tilde{s}^{\widetilde{U}\times\widetilde{U}}$ 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 $\tilde{s}^{\widetilde{U}\times\widetilde{V}}$.

Finally note that we have $r = 2$ and $n = -1$ in case (1). □

Definition 7.13. *In the situation of Lemma 7.12, 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 7.13 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 7.14. *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 7.12.

Lemma 7.15. *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* $V'' = V' \cup \{u\}$, $\widetilde{V} = V' \setminus \{w\}$ *and* M *be the adjacency matrix of* $G[V' \cup U]$ *(cf. Figure 7.13). 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* $\widetilde{V}$.
2. *Symmetric Gaussian elimination on* M *using* V' *(i.e. the elimination steps using* $\widetilde{V}$, *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 $\widetilde{V} \times \widetilde{V}$ 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 $\widetilde{V}$. 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 $\widetilde{V}$ can be swapped with elimination using wu, i.e. the results of

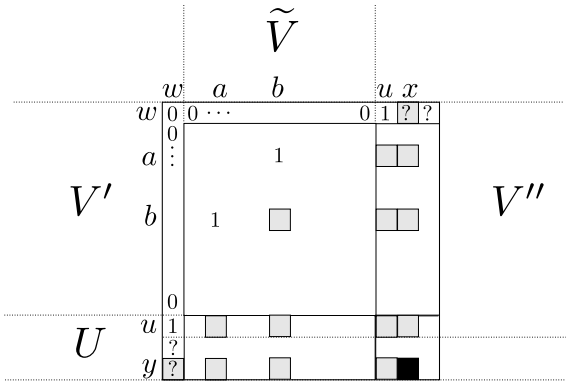

Figure 7.13: Symmetric Gaussian elimination using $\widetilde{V}$ (including steps such as eliminating with edge ab) and eliminating with edge wu can be swapped without changing the result. Empty entries and entries with a question mark are either 0 or 1.

$\rtimes ab \rtimes wu$ 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 $\widetilde{V}$, the case of a self loop at a in $\widetilde{V}$ can be dealt with similarly.

Let ab an edge in $\widetilde{V}$. 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 7.13.

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, $m_{yx}^{ab,wu}$ denotes the entry in row y and column x of $M^{ab,wu}$.

Case "ab first". By Lemma 7.5 we have

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

By Lemma 7.5 again, the final value of entry (y, x) is

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

where $m_{wx}^{ab} = m_{wx}$ and $m_{yw}^{ab} = m_{yw}$, as the elimination using edge ab does not affect column/row w (cf. Figure 7.13). Furthermore, we have

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

once more by Lemma 7.5.

Case "wu first". Here we have

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

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 7.5:

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

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

7.5 A Tree Decomposition Based Algorithm for Evaluation

Algorithm 2 evaluates Courcelle's multivariate interlace polynomial $C(G)$ 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 7.1 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.

7.5.1 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. For a node i, $D \subseteq X_i$, and a scenario s of X_i, one such part is defined in the following way:

$$S(i, D, s) = \sum_{A,B} x_A y_B u^{\mathrm{rk}((G_i \nabla B)[A \cup B])} v^{\mathrm{n}((G_i \nabla B)[A \cup B])}, \tag{7.1}$$

where the summation extends over all $A, B \subseteq V_i$ with $A \cap B = \emptyset$ and

$$\mathrm{scen}(G'[A \cup B, X_i]) = s, \quad G' = G \nabla (B \cup D).$$

Recall 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.

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.

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 every bag X_i of the tree decomposition contains at most k vertices

```
1: SupplyVertexOrder                                   ▷ Algorithm 1, Page 109
2: for all nodes i of the tree decomposition, in the order they appear in bottom-up traversal do
3:     for all D ⊆ X_i do
4:         if i is a leaf then
5:             S(i, D, ((), ())) ← 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:        end if
13:    end for
14: end for
15: return S(root, ∅, ((), ()))                        ▷ X_root = ∅
```

7.5.2 Join Nodes

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

Lemma 7.16. *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*

$$S(i, D, s) = \sum_{s_1, s_2} S(j_1, D, s_1) S(j_2, D, s_2), \tag{7.2}$$

where the summation extends over all scenarios s_1, s_2 of X_i such that

$$s_{\text{join}}(s_1, s_2, G \nabla D[X_i]) = s.$$

Proof. Recall (7.1) 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 7.7.)

We can apply Lemma 7.8 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 (7.1) corresponds to one pair $((A_1, B_1), (A_2, B_2))$ of the expanded version of (7.2). □

Algorithm 3 Computing the parts at a join node.

```
1: procedure JOIN(i, D)
2:     for all scenarios s for |X_i| vertices do
3:             ▷ i.e., enumerate all pairs s = (s^{X_i×V'}, s^{X_i×X_i}) with s^{X_i×V'} being
4:                a list of linear independent vectors from {0,1}^{X_i} and
5:                s^{X_i×X_i} a symmetric X_i × X_i matrix with entries from {0,1}
6:                – cf. Definition 7.2
7:         S(i, D, s) ← 0
8:     end for
9:     (j_1, j_2) ← (left child of i, right child of i)
10:    for all scenarios s_1, s_2 for |X_i| vertices do
11:        s ← s_join(s_1, s_2, GV D[X_i])                          ▷ Definition 7.9
12:        S(i, D, s) ← S(i, D, s) + S(j_1, D, s_1) · S(j_2, D, s_2)
13:    end for
14: end procedure
```

7.5.3 Introduce Nodes

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

Lemma 7.17. *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 7.10 it follows $s = s_{\text{introduce}}(s', a, G'[X_i])$. Conversely, Lemma 7.10 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 $\text{scen}(G'[A \cup B, X_i]) = s$. □

7.5.4 Forget Nodes

Finally, let us consider Algorithm 5, which handles forget nodes. As $\Delta n_{\text{forget}}(s', a, G')$ may be -1 in Lines 12 and 16, we have to assume $v \neq 0$. (The case $v = 0$ is discussed in Section 7.5.6.) Algorithm 5 is based on Lemma 7.18.

Lemma 7.18. *Let i be a forget node with child j and $X_j = X_i \cup \{a\}$. Let $D \subseteq X_i$,*

Algorithm 4 Computing the parts at an introduce node.

```
1: procedure INTRODUCE(i, D)
2:     for all scenarios s for |X_i| vertices do
3:         S(i, D, s) ← 0
4:     end for
5:     j ← child of i
6:     a ← vertex being introduced in X_i
7:     for all scenarios s' for |X_j| vertices do
8:         s ← s_introduce(s', a, G∇D[X_i])          ▷ Definition 7.11
9:         S(i, D, s) ← S(j, D \ {a}, s')
10:    end for
11: end procedure
```

$D' = D \cup \{a\}$ and s a scenario of X_i. Then

$$\begin{aligned} S(i,D,s) = &\sum_{s' \in \mathcal{S}_\mathrm{i}} S(j,D,s') \\ &+ \sum_{s' \in \mathcal{S}_\mathrm{f}} x_a u^{\Delta r_{\mathrm{forget}}(s',a,G\nabla D[X_j])} v^{\Delta n_{\mathrm{forget}}(s',a,G\nabla D[X_j])} S(j,D,s') \\ &+ \sum_{s' \in \mathcal{S}_{\mathrm{f}'}} y_a u^{\Delta r_{\mathrm{forget}}(s',a,G\nabla D'[X_j])} v^{\Delta n_{\mathrm{forget}}(s',a,G\nabla D'[X_j])} S(j,D',s'), \end{aligned} \tag{7.3}$$

where

$$\begin{aligned} \mathcal{S}_\mathrm{i} &= \{s' \mid s' \text{ scenario of } X_j \text{ with } s_{\mathrm{ignore}}(s',a) = s\}, \\ \mathcal{S}_\mathrm{f} &= \{s' \mid s' \text{ scenario of } X_j \text{ with } s_{\mathrm{forget}}(s',a,G\nabla D[X_j]) = s\}, \\ \mathcal{S}_{\mathrm{f}'} &= \{s' \mid s' \text{ scenario of } X_j \text{ with } s_{\mathrm{forget}}(s',a,G\nabla D'[X_j]) = s\}. \end{aligned}$$

Proof. We use (7.1) 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 (7.3). In case (2) we obtain the term corresponding to (A,B) from the second sum in (7.3), where we use Lemma 7.12 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. □

7.5.5 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.

Algorithm 5 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: **end for**
5: $j \leftarrow$ child of i
6: $a \leftarrow$ vertex being forgotten in X_i
7: **for all** scenarios s' for $|X_j|$ vertices **do**
8: $s \leftarrow s_{\text{ignore}}(s', a)$ ▷ Definition 7.14
9: $S(i, D, s) \leftarrow S(i, D, s) + S(j, D, s')$
10: $G' \leftarrow G\nabla D[X_j]$
11: $s \leftarrow s_{\text{forget}}(s', a, G')$ ▷ Definition 7.13
12: $S(i, D, s) \leftarrow S(i, D, s) + x_a u^{\Delta r_{\text{forget}}(s', a, G')} v^{\Delta n_{\text{forget}}(s', a, G')} S(j, D, s')$
13: $D' \leftarrow D \cup \{a\}$
14: $G' \leftarrow G\nabla D'[X_j]$
15: $s \leftarrow s_{\text{forget}}(s', a, G')$
16: $S(i, D, s) \leftarrow S(i, D, s) + y_a u^{\Delta r_{\text{forget}}(s', a, G')} v^{\Delta n_{\text{forget}}(s', a, G')} S(j, D', s')$
17: **end for**
18: **end procedure**

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 7.3). 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 11 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 7.4. Thus, the running time of Algorithm 2 is at most

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

As usual in the RAM model, 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 1.15), and this also holds for the parts (7.1), which are computed by the algorithm. Thus, we obtain the following result.

Theorem 7.19. *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, $v \neq 0$, and, for each $a \in V$, x_a and 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 $2^{3k^2+O(k)} \cdot n$ operations in the RAM model. 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] (Definition 1.14), which does not use self loop toggling in its definition, we do not need parameter D in (7.1) 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 7.19 but with running time reduced to $2^{1.5k^2+O(k)} \cdot n$.

7.5.6 Full-Rank Induced Subgraphs—the Case $v = 0$.

If $v = 0$, the summation in (1.22) extends only over the $A, B \subseteq V$, $A \cap B = \emptyset$, such that the adjacency matrix of $G\nabla B[A \cup B]$ has full rank. This sum can be evaluated using essentially the same techniques we have developed so far. Let us sketch briefly what changes have to be made.

Consider the situation described on Page 103, i.e. there is an extended graph $G[V', U]$, and symmetric Gaussian elimination on G using V' has been performed. The result is depicted in Figure 7.3. Let $\widetilde{S}$ denote the columns of the $U \times V'$ submatrix that are not "ruled" by any 1-entry of the $V' \times V'$ submatrix. (These columns are indicated by question marks in Figure 7.3.) Then the following holds: The adjacency matrix of $G[V' \cup U]$ has full rank only if $\widetilde{S}$ is linearly independent. If $U = \emptyset$, the converse is also true for trivial reasons. Following this observation, we can modify our algorithm to count full-rank induced subgraphs only and thus evaluate the interlace polynomial at points with $v = 0$.

The first modification is to extend Definition 7.7 as follows: The scenario of an extended graph $G[V', U]$ is said to have *full rank* if the column set $\widetilde{S}$ defined as above is linearly independent.

Next, we replace (7.1) by

$$S(i, D, s) = \sum_{A,B} x_A y_B u^{\mathrm{rk}((G_i \nabla B)[A \cup B])}, \tag{7.4}$$

where the summation extends over all A, B as in (7.1) with the additional restriction that the scenario of $G'[A \cup B, X_i]$, $G' = G\nabla(B \cup D)$, must have full rank.

Following the arguments in Section 7.4, it is possible to prove that full-rank scenarios can be used with tree decompositions in the same way as ordinary scenarios. For instance, the following version of Lemma 7.8 handles the join of full-rank scenarios:

Lemma 7.20 (Join for full-rank)**.** *Let $G = (V, E)$ be a graph, $U \subseteq V$, and s_1, s_2 be two scenarios of U. Then exactly one of the following cases applies:*

1. *For all disjoint subgraphs $G[V_1]$ and $G[V_2]$ of G such that*
   a) *$G[V_1]$ and $G[V_2]$ may be extended by U according to G,*
   b) *$G[V_1, U]$ has full-rank scenario s_1, and*

c) $G[V_2, U]$ has full-rank scenario s_2,

the scenario of $G[V_1 \cup V_2, U]$ is $s_{\text{join}}(s_1, s_2, G[U])$ but it does not have full rank.

2. *For the same family of graphs as in the first case, the following holds: The scenario of $G[V_1 \cup V_2, U]$ is $s_{\text{join}}(s_1, s_2, G[U])$ and it has full rank.*

Moreover, during the $\mathsf{poly}(|U|)$*-time computation of $s_{\text{join}}(s_1, s_2, G[U])$ as described in the proof of Lemma 7.8, it can be decided which of the two cases applies. We say that $s_{\text{join}}(s_1, s_2, G[U])$* preserves full rank *if the second case applies.*

In the algorithm, scenario-sums must be counted only if the scenario has full rank. For instance, join nodes can be handled by Algorithm 6, which is a slight modification of Algorithm 3.

Algorithm 6 Computing the full-rank parts at a join node.

```
1: procedure JOIN_FULL_RANK(i, D)
2:     for all scenarios s for |X_i| vertices do
3:         S(i, D, s) ← 0
4:     end for
5:     (j_1, j_2) ← (left child of i, right child of i)
6:     for all scenarios s_1, s_2 for |X_i| vertices do
7:         if s_join(s_1, s_2, G∇D[X_i]) preserves full rank then
8:             s ← s_join(s_1, s_2, G∇D[X_i])
9:             S(i, D, s) ← S(i, D, s) + S(j_1, D, s_1) · S(j_2, D, s_2)
10:        end if
11:    end for
12: end procedure
```

In this way, Theorem 7.19 can be established for the case $v = 0$ as well.

7.6 Variants of the Algorithm

7.6.1 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 with maximum bag size k. 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] (cf. Definition 1.25).

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 7.5 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.

7.6.2 Parallelization

Let us 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. This 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. Let $d = 2^k 2^{3(k+1)k/2}$ denote a upper bound on the dimension of the vector. 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. For leaves, we define the input to be just the output.

We associate a matrix A_i with each 1-node i. The computation of the 1-node i is $\sigma_i = A_i\sigma_j$, where j is the child of i: For an introduce node i with child j, we trivially can write $\sigma_i = A_i\sigma_j$ for some matrix A_i by Lemma 7.17. The entries of A_i are either 0 or 1. Now let i be a forget node with child j. Consider (7.3). 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, $x_a u^l v^{1-l}$ or $y_a u^l v^{1-l}$, where $l \in \{0, 1, 2\}$.

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

$$\sigma_i(D, s) = \sum \sigma_{j_1}(D, s_1)\sigma_{j_2}(D, s_2), \tag{7.5}$$

where the sum is taken over the same elements as in (7.2).

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 arith-

metic 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 and produces $O(n)$ gates.

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

Path contraction works as follows. For a sequence $i_1, i_2, \ldots, i_\ell$ 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 $\widetilde{A} = A_{i_\ell} \cdot \ldots \cdot A_{i_1}$ associated with it and gets σ_j as input. Every matrix multiplication can be performed adding $O(d^\omega)$ gates, where ω is the exponent of matrix multiplication. 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 (7.5). This increases the depth by 2. The number of gates added is bounded by $O(d^2)$. In case (2), node i becomes a 1-node: The $\sigma_{j_1}(D, s)$ in (7.5) become coefficients of a new matrix $\widetilde{A}$ associated to i. As 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 multiplication of matrix and a vector, $A_i \sigma_j$, to compute σ_i. This adds $O(d^2)$ gates and increases the depth by a constant. Thus, removing all leaves in L adds $O(d^2)$ gates and 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, the tree decomposition is reduced to a single leaf after $O(\log n)$ steps. The depth increases by at most $O(\log n)$ in each step, which gives a $O(\log^2 n)$ bound on the depth of the constructed arithmetic circuit. The number of matrix multiplications is $O(n)$ as every node of the tree decomposition corresponds to at most one matrix multiplication. This proves the following theorem.

Theorem 7.21. *The interlace polynomial $C(G)$ of a graph G with n vertices and a tree decomposition of width k with $O(n)$ nodes can be computed by an arithmetic*

circuit of depth $O(\log^2 n)$ and size $2^{\frac{3}{2}\omega k^2+O(k)} \cdot n$, where $\omega \geq 2$ is the exponent of matrix multiplication.

7.6.3 Computation of Coefficients

As discussed in Section 7.6.1, 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 7.22 ([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 7.23. *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 7.23. 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 7.6.1, 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

$$|\mathcal{M}| \leq (2n+1)^d(n+1)^2 = n^{d\left(1+\frac{O(1)}{\log n}\right)+O(1)}. \tag{7.6}$$

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 7.22, 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. □

7.6.4 Graphs of Unbounded Treewidth

Assume that a graph G is given, the treewidth of which is possibly unbounded. It is easy to see that the two-variable interlace polynomial $q(G; x, y)$ by Arratia et al. (Def. 1.14) can be evaluated using $2^n\mathsf{poly}(n)$ arithmetic operations. Similarly, Courcelle's multivariate interlace polynomial $C(G)$ (Def. 1.15) can be evaluated using $3^n \cdot \mathsf{poly}(n)$ operations. This is not the best possible: Using the techniques developed in this chapter, we can improve to $o(2^n)$ ($o(3^n)$, resp.) arithmetic operations.

Every graph with vertices $V = \{v_1, \ldots, v_n\}$ has the trivial tree decomposition

$$\{\}, \{v_1\}, \{v_1, v_2\}, \ldots, \{v_1, \ldots, v_n\}, \{v_2, \ldots, v_n\}, \ldots, \{v_n\}, \{\}.$$

We use this tree decomposition as input for Algorithm 2. The running time is basically determined by the number of parts $S(i, D, s)$ that are processed. In particular, no join steps occur. We notice that we only have to maintain the nonzero parts $S(i, D, s)$. Let us first consider the interlace polynomial without the self loop toggling feature, i.e. $q(G; x, y)$ and $\bar{q}(G; u, x)$. Then we do not need the parameter D and the parts can be written as $S(i, s)$, where i is a node of the tree decomposition and s a scenario.

At the beginning we have only one scenario, the trivial scenario. The corresponding part has value 1. The first n steps are introduce steps. If we proceed from node i to node j with an introduce step, each part $S(j, s)$ either equals a part $S(i, s')$, where s' is some scenario of X_i, or is zero. No two parts $S(i, s'_1)$, $S(i, s'_2)$, $s'_1 \neq s'_2$ correspond to the same part $S(j, s)$. Thus, after an introduce step, the number of nonzero parts of j equals the number of nonzero parts of i. Consequently, after the first n steps, at node $i = \{v_1, \ldots, v_n\}$, there is only one scenario s such that $S(i, s) \neq 0$.

We have the following tradeoff for the following n forget steps:

- In a forget step from node j to node i, the number of scenarios defining a nonzero part can at most double: each ignore can create a new nonzero sum and each forget can create a new nonzero sum. Thus, the number of nonzero parts of the respective node after each of the last n steps is bounded by

$$1, 2, 4, \ldots, 2^n.$$

- On the other hand, the size of the nodes of the tree decomposition during the last n forget steps are n, $n-1$, $n-2$, …, 0. Thus, by Lemma 7.3, the number of nonzero parts at the respective node after these steps is bounded by

$$2^{(3n+1)n/2}, 2^{(3(n-1)+1)(n-1)/2}, 2^{(3(n-2)+1)(n-2)/2}, \ldots, 1.$$

Thus, in the first n steps, we only have one nonzero part each time. In the ith of the last n steps, the number of nonzero parts we have to consider is bounded by

$$\min\{2^i, 2^{(3(n-i)+1)(n-i)/2}\}.$$

For $0 \leq i \leq n$, we have

$$i > (3(n-i)+1)(n-i)/2$$

if and only if

$$i > n + 1/2 - \sqrt{2n/3 + 1/4}.$$

Thus, the algorithm can be implemented such that it uses at most

$$2^{n+1/2-\sqrt{2n/3+1/4}} \cdot \mathsf{poly}(n) \leq 2^{n-0.816\sqrt{n}+O(\log n)}$$

arithmetic operations. We have proven the following theorem.

Theorem 7.24. *The interlace polynomials without self loop toggling, i.e. $q(G; x, y)$, $\bar{q}(G; u, x)$, can be computed with $2^{n-\Omega(\sqrt{n})} = o(2^n)$ operations, $n = |V(G)|$.*

Proof. As we have argued, $\bar{q}(G; u, x)$ can be evaluated in $o(2^n)$ operations. To compute the coefficients, we use interpolation on a two-dimensional grid. As the degree of $\bar{q}(G; u, x)$ in u and x is bounded by n, this needs only $O(n^2)$ evaluations. Thus, the running time is not increased substantially.

Once we have computed $\bar{q}(G; u, x)$, we can compute $q(G; x, y)$ by Lemma 1.17. □

Theorem 7.25. *The interlace polynomial $C(G)$ can be computed with $3^{n-\Omega(\sqrt{n})} = o(3^n)$ operations.*

Proof. We use the trivial tree decomposition again. Consider a node i.

- We can compute (7.1) directly. For each of the $2^{|X_i|}$ sets D this takes $3^{|V_i|}\mathsf{poly}(n)$ operations. This means $2^{|X_i|+|V_i|\log 3}\mathsf{poly}(n)$ operations for node i.
- On the other hand, there are at most $2^{(3|X_i|+1)|X_i|/2}2^{|X_i|}$ pairs (s, D) and thus at most that many sums $S(i, D, s)$ for node i. For each such sum $S(i, D, s)$, we need only $\mathsf{poly}(n)$ operations to compute to which sums of the next node of the tree decomposition the sum ($S(i, D, s)$ contributes.

Let us consider the last n nodes of the trivial tree decomposition, for the ith of which we have $V_i = \{v_1, \ldots, v_i\}$ and $X_i = \{v_{i+1}, \ldots, v_n\}$. On this nodes, the time needed by the first method increases from $2^n\mathsf{poly}(n)$ to $3^n\mathsf{poly}(n)$ operations. The time needed by the second method decreases from $2^{n+(3n+1)n/2}\mathsf{poly}(n)$ to $\mathsf{poly}(n)$ operations. Let us compute the i, $0 \leq i \leq n$, where the second method becomes cheaper than the first. Substituting $n - i$ by j, this is

$$(n-j)\log 3 + \ell \log n = (3j+1)j/2 + k \log n$$

for some constants $k, \ell \in \mathbb{N}$. This means

$$0 = j^2 + pj - q^2, \tag{7.7}$$

where $p = \frac{1+2\log 3}{3}$ and $q = \sqrt{\frac{2}{3}(n\log 3 - (k-\ell)\log n)}$. We are only interested in the solution where $0 \le i \le n$, which is

$$i = n - \sqrt{q^2 + \frac{p^2}{4}} + \frac{p}{2}. \tag{7.8}$$

We have $q^2 + \frac{p^2}{4} \ge (q - \frac{p}{2})^2$. Thus, by (7.8), the second method is cheaper than the first if

$$i \ge \hat{i} := n - q + p. \tag{7.9}$$

Therefore, we switch from the first method to the second as soon as (7.9). The overall running time is bounded by

$$\begin{aligned} 2^{n-\hat{i}+\hat{i}\log 3}\mathsf{poly}(n) &\le 2^{n+(\log 3-1)(n-q+p)+O(\log n)} \\ &= 3^{n-q(1-\log_3 2)+O(\log n)} \le 3^{n-0.379\sqrt{n}}. \end{aligned}$$

□

7.7 Faster Algorithms for Special Cases

How fast can the interlace polynomial of an n-vertex graph G be computed? For $\bar{q}(G;u,x)$, we have improved the trivial $2^n\mathsf{poly}(n)$ bound to $2^{n-\Omega(\sqrt{n})}$ (Theorem 7.24). Now we discuss two special cases, where we give algorithms running in time c^n, $c < 2$.

7.7.1 Graphs with Bounded Maximum Degree

The first case is that Δ, the maximum degree of G, is bounded by a constant and G has no self loops.

Theorem 7.26. *Let G be a simple graph with n vertices and maximum degree Δ. Then the interlace polynomials $q(G;x,y)$ and $\bar{q}(G;u,x)$ can be computed in time λ^n with $\lambda = (2^{\Delta+1} - \Delta - 1)^{\frac{1}{\Delta+1}} < 2$.*

Proof. Let a be a vertex of $G = (V,E)$ whose neighbors are exactly $b_1, \ldots, b_\ell$. Let $B = \{b_1, \ldots, b_\ell\}$ and $V' = V \setminus \{a, b_1, \ldots, b_\ell\}$. Then we have

$$\operatorname{rk} G[A \cup \{a,b\}] = \operatorname{rk} G[A] + 2, \quad b \in \{b_1, \ldots, b_\ell\}, \tag{7.10}$$

$$\operatorname{rk} G[A \cup \{a\}] = \operatorname{rk} G[A] \tag{7.11}$$

for every $A \subseteq V'$. For $W \subseteq V$ and $S \subseteq V \setminus W$, define

$$q(G;W,S) = \sum_{A \subseteq W} x_A u^{\operatorname{rk} G[A \cup S]}.$$

Now let $W \subseteq V$ be such that $a, b_1, \ldots, b_\ell \in W$. Define $W' = W \setminus \{a, b_1, \ldots, b_\ell\}$. Then we have

$$\begin{aligned} q(G; W, S) = &\sum_{T \subseteq \{a, b_1, \ldots, b_\ell\}} x_T q(G; W', S \cup T) \\ =&(1 + x_a(1 + u^2(x_{b_1} + \ldots x_{b_\ell})))q(G; W', S) + \\ &\sum_{\substack{T \subseteq \{a, b_1, \ldots, b_\ell\} \\ T \neq \emptyset,\ T \neq \{a\}, \\ T \neq \{a, b\} \text{ for all } b \in B}} x_T q(G; W', S \cup T) \qquad \text{by (7.10) and (7.11).} \end{aligned} \tag{7.12}$$

Equation (7.12) is a recursion to compute $q(G; W, S)$. Let $T(n)$ denote the maximum number of operations to compute $q(G; W, S)$ for W with $|W| = n$ and arbitrary $S \subseteq V \setminus W$. Then (7.12) implies

$$T(n) \leq (2^{\ell+1} - (\ell + 1))T(n - (\ell + 1)). \tag{7.13}$$

The solution of this is $T(n) = O(\lambda^n)$ with $\lambda = \sqrt[\ell+1]{(2^{\ell+1} - (\ell + 1))}$.

We evaluate the interlace polynomial $\bar{q}(G; u, x)$ by applying (7.12) recursively, starting with $q(G; V, \emptyset)$. Evaluating $\bar{q}(G; u, x)$ at n^2 points of a $u \times x$ grid, we can interpolate $\bar{q}(G; u, x)$. Via Lemma 1.17, we can obtain $q(G; x, y)$. □

If G has self loops, we can proceed in the same way as long as there is at least one vertex that does not have a self loop. If all vertices have self loops, we can branch using an recursion similar to (1.19). In one of the two branches, there will be a vertex without a self loop. It seems plausible that this approach also leads to an algorithm with running time better than 2^n. We leave it as a research problem to give a precise analysis.

7.7.2 Small u-Coefficients of $\bar{q}(G; u, x)$

Counting independent sets, which is equivalent to evaluation of the interlace polynomial $\bar{q}$ at $u = 0$, can be done in time c^n for some $c < 2$. In fact, there has been a lot of work on the independent set problem, starting with Tarjan and Trojanowski [TT77]. They gave an algorithm for finding a maximum independent set in time $O(1.2599^n)$, which was the first algorithm better than the trivial $O(2^n)$. In most of the subsequent improvements, the interest has been on finding maximum independent sets [Jia86, Rob86, FGK06], but counting independent sets has also been considered [DJ02]. We take no attempt to translate all the techniques developed for the independent set problem to the interlace polynomial. But we give evidence that computation of the interlace polynomial can gain from techniques for counting independent sets.

Let $G = (V, E)$ be a simple graph, $W, S \subseteq V$, $W \cap S = \emptyset$, u a variable, and $\mathbf{x}$ a V-indexed variable. We define

$$q_i(G, W, S; u, \mathbf{x}) = \sum_{A \subseteq W} x_A u^{\operatorname{rk} G[A \cup S]} \bmod u^i. \tag{7.14}$$

Lemma 7.27. *Let ab be an edge, $a, b \in W$, and $W' = W \setminus \{a, b\}$. Then we have*

$$\begin{aligned} q_i(G, W, S) =& q_i(G, W', S) + x_a q_i(G, W', S \cup \{a\}) \\ &+ x_b q_i(G, W', S \cup \{b\}) + x_a x_b u^2 q_{i-2}(G^{ab}, W', S). \end{aligned} \tag{7.15}$$

Proof.

$$\begin{aligned} q_i(G, W, S) = \sum_{A \subseteq W'} x_A \big(& u^{\operatorname{rk} G[A \cup S]} + x_a u^{\operatorname{rk} G[A \cup S \cup \{a\}]} \\ & + x_b u^{\operatorname{rk} G[A \cup S \cup \{b\}]} + x_a x_b u^{\operatorname{rk} G[A \cup S \cup \{a,b\}]} \big) \end{aligned}$$

Let us analyze $\operatorname{rk} G[A \cup S \cup \{a, b\}]$. The adjacency matrix of $G[A \cup S \cup \{a, b\}]$ has the form

$$M = \left(\begin{array}{c|cc|cccc} & a & b & N_{ab} & N_a & N_b & R \\ \hline a & 0 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ b & 1 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ N_{ab} & \mathbf{1} & \mathbf{1} & A_{ab,ab} & A_{ab,a} & A_{ab_b} & A_{ab,R} \\ N_a & \mathbf{1} & \mathbf{0} & A^T_{ab,a} & A_{a,a} & A_{a_b} & A_{a,R} \\ N_b & \mathbf{0} & \mathbf{1} & A^T_{ab,b} & A^T_{a,b} & A_{b_b} & A_{b,R} \\ R & \mathbf{0} & \mathbf{0} & A^T_{ab,R} & A^T_{a,R} & A^T_{b_R} & A_{R,R} \end{array} \right),$$

where N_{ab} are the common neighbors of a and b, N_a (N_b) the neighbors of a (b) that are not neighbors of b (a), and $R = V \setminus (N_{ab} \cup N_a \cup N_b)$. We add the b column to the N_{ab} and N_a columns and do the same with the respective columns. Then we add the a row to the N_{ab} and N_b rows and do the same with the respective columns. This yields

$$M' = \left(\begin{array}{c|cc|cccc} & a & b & N_{ab} & N_a & N_b & R \\ \hline a & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ b & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ N_{ab} & \mathbf{0} & \mathbf{0} & A_{ab,ab} & (A_{ab,a})^C & (A_{ab,b})^C & A_{ab,R} \\ N_a & \mathbf{0} & \mathbf{0} & (A^T_{ab,a})^C & A_{a,a} & (A_{a,b})^C & A_{a,R} \\ N_b & \mathbf{0} & \mathbf{0} & (A^T_{ab,b})^C & (A^T_{a,b})^C & A_{b,b} & A_{b,R} \\ R & \mathbf{0} & \mathbf{0} & A^T_{ab,R} & A^T_{a,R} & A^T_{b,R} & A_{R,R} \end{array} \right).$$

Thus, $\operatorname{rk} G[A \cup S \cup \{a, b\}] = 2 + \operatorname{rk} G^{ab}[A \cup S]$, which implies the lemma. □

It is interesting to note that (7.15) corresponds to the following branching method to count independent sets: the independent sets of $G[W]$ either

- contain neither a nor b, which implies a recursive call on an instance of size $n - 2$ – this corresponds to the term $q_i(G, W', S)$; or
- contain a but not b, which implies a recursive call on an instance of size $n - 2$ (as a and b are neighbors, b can not be in an independent set that contains a) – this corresponds to the term $q_i(G, W, S \cup \{a\})$; or

- contain b but not a, which implies a recursive call on an instance of size $n-2$ – this corresponds to the term $q_i(G, W, S \cup \{b\})$.

It is impossible that an independent set of $G[W]$ contains both a and b. Thus, our simple branching algorithm can count independent sets in time $T(n)$, where $T(n) = 3T(n-2)$. The asymptotic of the solution is $T(n) = O(1.7321^n)$, which improves on the trivial $O(2^n)$ algorithm for counting independent sets.

The improvement is due to the observation that subsets $A \subseteq W$ with $a, b \in W$ are "easy" subsets, as they can not be independent. Thus, they do not cause a recursive call. For the interlace polynomial, these subsets are not that trivial that we did not have to recurse on them. But the recursion they initiate are "easier" than the others in the sense that we only need q_{i-2} instead of q_i. This is enough to obtain an algorithm computing the interlace polynomial $\bar{q}(G; u, x) \bmod u^k$ faster than in time $O(2^n)$ as long as k is not too large: Let $T_i(n)$ be the time to evaluate $q_i(G, W, S)$ for an arbitrary simple graph $G = (V, E)$ with $W \subseteq V$, $|W| = n$ and $S \subseteq V \setminus W$. By (7.15), we have

$$T_i(n) = 3T_i(n-2) + T_{i-2}(n-2)$$

for $i \geq 2$ and $T_0(n) = 3T_0(n-2)$. Induction shows that $T_k(n) = \sqrt{3}^n O(n^k)$, which establishes the following result.

Theorem 7.28. *Let G be a graph with n vertices. The interlace polynomial $\bar{q}(G; u, \mathbf{x}) \bmod u^k$ can be evaluated using $(\sqrt{3})^n O(n^k)$ operations in the RAM model. If the bit length of u and all x_a, $a \in V(G)$, is at most ℓ, the bit length of the numbers occurring during the computation is at most $O(\ell n)$.*

Proof. It only remains to argue on the bit length of the numbers occurring during the computations. The claim of the theorem follows from that fact that all intermediate results, considered as polynomials in u and x, have degree at most $2n$. □

7.8 Open Problems

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.

Can the proofs in Section 7.4 be simplified, possibly using inspiration from Brijder and Hoogeboom [BH09a, BH09b]?

Can the running time bound of Algorithm 2 (Theorem 7.19), be decreased using the technique of van Rooij, Bodlaender, and Rossmanith [vRBR09]?

Considering Theorem 7.21, is there a method to produce an arithmetic circuit of depth $O(\log n)$ computing the interlace polynomial of a graph with n vertices and treewidth k (cf. Elberfeld, Jakoby, and Tantau [EJT10])?

A precise analysis of the approach sketched at the end of Section 7.7.1 has not been provided yet.

Bibliography

[ABCS00] Richard Arratia, Béla Bollobás, Don Coppersmith, and Gregory B. Sorkin. Euler circuits and DNA sequencing by hybridization. *Discrete Applied Mathematics*, 104(1-3):63–96, 15 August 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.

[AGM10] Ilia Averbouch, Benny Godlin, and J.A. Makowsky. An extension of the bivariate chromatic polynomial. *European Journal of Combinatorics*, 31(1):1–17, 2010.

[AM07] Ilia Averbouch and J. A. Makowsky. The complexity of multivariate matching polynomials, March 2007. Preprint.

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

[AP09] Saeid Alikhani and Yee-hock Peng. Introduction to domination polynomial of a graph, 2009. Preprint, arXiv:math.CO/0905.2251v1.

[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.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*, volume - of *International Computer Science Institute (ICSI) Berkeley, Calif.: Report*. Springer, Berlin - Heidelberg - New York, 1998.

[BDM10] Markus Bläser, Holger Dell, and Johann A. Makowsky. Complexity of the Bollobás-Riordan polynomial. exceptional points and uniform reductions. *Theory Comput. Syst.*, 46(4):690–706, 2010.

[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)*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 97–108. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. Updated full version: arXiv:cs.CC/0707.4565v3.

[BH09a] Robert Brijder and Hendrik Jan Hoogeboom. The group structure of pivot and loop complementation on graphs and set systems. *CoRR*, abs/0909.4004, 2009.

[BH09b] Robert Brijder and Hendrik Jan Hoogeboom. Nullity invariance for pivot and the interlace polynomial. *CoRR*, abs/0912.0878, 2009.

[BHKK07] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 67–74. ACM, 2007.

[BHKK08] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *FOCS*, pages 677–686. IEEE Computer Society, 2008.

[Bir12] George D. Birkhoff. A determinant formula for the number of ways of coloring a map. *The Annals of Mathematics*, 14(1/4):42–46, 1912.

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

[BM06] Markus Bläser and Johann Makowsky. Hip hip hooray for Sokal, 2006. Unpublished note.

[BO92] Thomas Brylawski and James Oxley. The Tutte polynomial and its applications. In Neil White, editor, *Matroid Applications*, Encyclopedia of Mathematics and its Applications, pages 123–225. Cambridge University Press, 1992.

[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.

[Bol98] Béla Bollobás. *Modern Graph Theory*. Springer, 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.

[Bou94] André Bouchet. Circle graph obstructions. *J. Comb. Theory Ser. B*, 60(1):107–144, 1994.

[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.

[Bür00] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Springer, 2000.

[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.

[CL72] Martin Cohn and Abraham Lempel. Cycle decomposition by disjoint transpositions. *Journal of Combinatorial Theory, Series A*, 13(1):83–89, 1972.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2. ed. edition, 2001.

[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.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[DFJ02] Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM Journal on Computing*, 31(5):1527–1541, 2002.

[DG00] Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3-4):260–289, 2000.

[DHW10] Holger Dell, Thore Husfeldt, and Martin Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. Technical Report TR10-078, Electronic Colloquium on Computational Complexity, 2010.

[Die05] Reinhardt Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, 3. edition, 2005.

[DJ02] Vilhelm Dahllöf and Peter Jonsson. An algorithm for counting maximum weighted independent sets and its applications. In *SODA*, pages 292–298, 2002.

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

[DPT03] Klaus Dohmen, André Pönitz, and Peter Tittmann. A new two-variable generalization of the chromatic polynomial. *Discrete Mathematics & Theoretical Computer Science*, 6(1):69–90, 2003.

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. Technical Report TR10-062, Electronic Colloquium on Computational Complexity, 2010.

[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.

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

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[FGK06] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer: a simple $O(2^{0.288n})$) independent set algorithm. In *SODA*, pages 18–25. ACM Press, 2006.

[FGLS10] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

[FK72] C. M. Fortuin and P. W. Kasteleyn. On the random-cluster model: I. Introduction and relation to other models. *Physica*, 57(4):536–564, 1972.

[GG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge Univ. Press, Cambridge, 2. ed. edition, 2003.

[GH83] I. Gutman and F. Harary. Generalizations of the matching polynomial. *Utilitas Math.*, 24:97–106, 1983.

[GJ07] Leslie Ann Goldberg and Mark Jerrum. Inapproximability of the Tutte polynomial. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 459–468, New York, NY, USA, 2007. ACM Press.

[GJS76] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[Gol91] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Computer Science and Applied Mathematics. Academic Press, New York - London - Toronto, 3. edition, 1991.

[GP06] Roland Glantz and Marcello Pelillo. Graph polynomials from principal pivoting. *Discrete Mathematics*, 306(24):3253–3266, 2006.

[GSH89] Csaba P. Gabor, Kenneth J. Supowit, and Wen-Lian Hsu. Recognizing circle graphs in polynomial time. *J. ACM*, 36(3):435–473, 1989.

[HL94] Cornelis Hoede and Xueliang Li. Clique polynomials and independent set polynomials of graphs. *Discrete Mathematics*, 125(1-3):219 – 228, 1994.

[Hof10] Christian Hoffmann. A most general edge elimination polynomial - thickening of edges. *Fundamenta Informaticae*, 98(4):373–378, 2010. Preliminary version: arXiv:0801.1600v1 [math.CO], 2008.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[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.

[Jer81] Mark Jerrum. *The complexity of evaluating multivariate polynomials.* PhD thesis, Department of Computer Science, University of Edinburgh, 1981.

[Jia86] Tang Jian. An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. *IEEE Trans. Computers*, 35(9):847–851, 1986.

[JVV86] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comp. Sc.*, 43:169–188, 1986.

[JVW90] F. Jaeger, D. L. Vertigan, and D. J. A. Welsh. On the computational complexity of the Jones and the Tutte polynomials. *Math. Proc. Cambridge Philos. Soc.*, 108:35–53, 1990.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

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

[Kot68] Anton Kotzig. Eulerian lines in finite 4-valent graphs and their transformations. In *Theory of Graphs*, Proc. Colloq., Tihany, 1966, pages 219–230. Academic Press, New York, 1968.

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

[Lin86] Nathan Linial. Hard enumeration problems in geometry and combinatorics. *SIAM J. on Algebraic and Discrete Methods*, 7:331–335, 1986.

[LM04] Martin Lotz and Johann A. Makowsky. On the algebraic complexity of some families of coloured Tutte polynomials. *Advances in Applied Mathematics*, 32(1–2):327–349, 2004.

[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.

[LV97] Michael Luby and Eric Vigoda. Approximately counting up to four (extended abstract). In *STOC*, pages 682–687, 1997.

[Mak08] Johann A. Makowsky. From a zoo to a zoology: Towards a general theory of graph polynomials. *Theory Comput. Syst.*, 43(3–4):542–562, 2008.

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

[MS08] Kurt Mehlhorn and Peter Sanders. *Algorithms and data structures.* Springer, Berlin ; Heidelberg, 2008.

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

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[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.

[OW79] J. G. Oxley and D. J. A Welsh. The Tutte polynomial and percolation. In J. A. Bondy and U. S. R. Murty, editors, *Graph Theory and Related Topics*, pages 329–339. Academic Press, New York, 1979.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison Wesley Longman, 1994.

[Rob86] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[Rot96] Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

[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.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226. ACM, 1978.

[Sin88] Alistair Sinclair. *Randomized algorithms for counting and generating combinatorial structures*. PhD thesis, Department of Computer Science, University of Edinburgh, 1988.

[SIT95] Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the Tutte polynomial of a graph of moderate size. In John Staples, Peter Eades, Naoki Katoh, and Alistair Moffat, editors, *ISAAC*, volume 1004 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 1995.

[Sok04] Alan D. Sokal. Chromatic roots are dense in the whole complex plane. *Combinatorics, Probability & Computing*, 13(2):221–261, 2004.

[Sok05] Alan D. Sokal. The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In Bridget S. Webb, editor, *Surveys in Combinatorics 2005*. Cambridge University Press, 2005. arXiv:math.CO/0503607v1.

[Spi94] Jeremy Spinrad. Recognition of circle graphs. *J. Algorithms*, 16(2):264–282, 1994.

[SS05] Alexander D. Scott and Alan D. Sokal. The repulsive lattice gas, the independent-set polynomial, and the Lovász local lemma. *J. Stat. Phys.*, 118:1151, 2005.

[Sta73] Richard P. Stanley. Acyclic orientations of graphs. *Disc. Math.*, 5:171–178, 1973.

[Str73] Volker Strassen. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 264:184—-202, 1973.

[Tra09] Lorenzo Traldi. Binary nullity, Euler circuits and interlace polynomials, 2009. Preprint, arXiv:0903.4405v2. To appear in European Journal of Combinatorics.

[Tra10a] Lorenzo Traldi. On the interlace polynomials, 2010. Preprint, arXiv:1008.0091v2.

[Tra10b] Lorenzo Traldi. Weighted interlace polynomials. *Combinatorics, Probability & Computing*, 19(1):133–157, 2010.

[Tsa00] Michael J. Tsatsomeros. Principal pivot transforms: properties and applications. *Linear Algebra and its Applications*, 307:151–165, 2000.

[TT77] Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977.

[Tut84] W. T. Tutte. *Graph Theory.* Addison Wesley, 1984.

[Vad95] Salil Vadhan. The complexity of counting. Bachelor's Thesis, Havard College, Cambridge, Massachusetts, 1995.

[Vad01] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.

[Val79a] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Val79b] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Ver05] Dirk Vertigan. The computational complexity of Tutte invariants for planar graphs. *SIAM Journal on Computing*, 35(3):690–712, 2005.

[vRBR09] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009.

[Wel93] D. J. A. Welsh. *Complexity: knots, colourings and counting.* Cambridge University Press, New York, NY, USA, 1993.

[Whi32] Hassler Whitney. A logical expansion in mathematics. *Bull. Amer. Math. Soc.*, 38:572–579, 1932.

[Yat37] Frank Yates. The design and analysis of factorial experiments. Technical Communication No. 35, Commonwealth Bureau of Soil Science, Harpenden, UK, 1937.