The Complexity of Graph Polynomials

Steven D. Noble

New College

Oxford

Trinity Term 1997

Submitted For D. Phil

This thesis examines graph polynomials and particularly their complexity. We give short proofs of two results from Gessel and Sagan (1996) which present new evaluations of the Tutte polynomial concerning orientations. A theorem of Massey et al (1997) gives an expression concerning the average size of a forest in a graph. We generalise this result to any simplicial complex. We answer a question posed by Kleinschmidt and Onn (1995) by showing that the language of partitionable simplicial complexes is in NP.

We prove the following result concerning the complexity of the Tutte polynomial.

Theorem 1. *For any fixed k, there exists a polynomial time algorithm $\mathcal{A}$, which will input any graph G, with tree-width at most k, and rational numbers x and y and evaluate the Tutte polynomial, $T(G; x, y)$.*

The rank generating function S of a graphic 2-polymatroid was introduced by Oxley and Whittle (1993). It has many similarities to the Tutte polynomial and we prove the following results.

Theorem 2. *Evaluating S at a fixed point (u, v) is $\#P$-hard unless $uv=1$ when there is a polynomial time algorithm.*

Theorem 3. *For any fixed k, there exists a polynomial time algorithm $\mathcal{A}$, which will input any graph G, with tree-width at most k, and rational numbers x and y and evaluate $S(G; u, v)$.*

We consider a class of graphs $\mathcal{S}$, which are those graphs which are obtainable from a graph with no edges using the unsigned version of Reidemeister moves. We examine the relationship between this class and other similarly defined classes such as the delta-wye graphs. There remain many open questions such as whether $\mathcal{S}$ contains every graph. However we have an invariant of the moves, based on the Tutte polynomial, which allows us to determine from which graph with no edges, if any, a particular graph can be obtained.

Finally we consider a new polynomial on weighted graphs which is motivated by the study of weight systems on chord diagrams. We give three states model and a recipe theorem. An unweighted version of this polynomial is shown to contain as specialisations, a wide range of graph invariants, such as the Tutte polynomial, the polymatroid polynomial of Oxley and Whittle (1993) and the symmetric function generalisation of the chromatic polynomial introduced by Stanley (1995). We close with a discussion of complexity issues proving hardness results for very restricted classes of graphs.

Contents

List of Figures

List of Algorithms

Acknowledgements

It is with great pleasure that I acknowledge the contribution of my supervisor, Dominic Welsh. Without his time, enthusiasm, encouragement and particularly his inexhaustible supply of interesting problems, this thesis would never have been written. I am very grateful that he did not seem to mind any of my many vague claims of finding a proof.

There are many others who I wish to thank, notably Wilson Sutherland and David Mayers for their teaching and more recently encouragement, and the other members of the combinatorics group for lively and useful discussions. I would also like to thank Eric Bartels for his help with computers.

I am grateful to many others in Oxford, particularly those who I have had the pleasure to play cricket and bridge with and above all, the other five members of the 'dream team' for a wonderful year of great success and enjoyment.

Introduction

This thesis examines graph polynomials, mainly focusing on their complexity. One of the reasons for studying graph polynomials is that they can provide a link between apparently quite diverse graph invariants, for instance an evaluation of the Tutte polynomial gives the number of spanning trees of a graph whilst other evaluations give the number of colourings.

Much, but not all, of this thesis is concerned with complexity. There are two obvious complexity questions concerning graph polynomials namely the complexity of evaluating a polynomial at a particular point and the complexity of computing a coefficient of a polynomial. In this thesis we mainly consider the former question.

We begin by introducing the main topics of the thesis, namely graphs, complexity and the Tutte polynomial. Also included in the first chapter are three short results which illustrate these concepts. We give shorter proofs of some results from [GS96] which describe new graph invariants involving orientations. The next section consists of [Nob96] where we answer a question raised in [KO95] concerning the complexity of a natural problem involving simplicial complexes. A result from [MSS$^+$95] gives an equation involving the Tutte polynomial, which as a special case leads to an interesting expression concerning the average size of a forest. In the final section we give a simple

proof of a slight extension of this result and show that it generalises to any simplicial complex.

The Tutte polynomial has been shown in [JVW90, VW92] to be $\#P$-hard to evaluate at most fixed points even for a fairly restricted input such as bipartite planar graphs. A natural problem is then to find a large class of graphs for which there is a polynomial time algorithm to evaluate the Tutte polynomial. Many well-known hard problems have been shown to have polynomial time algorithms to solve them when the input is restricted to a class of graphs with bounded tree-width, see for instance [Bod93] or [AP89]. Chapter 2, which is mainly the contents, of [Nob97/98] gives a polynomial time algorithm to evaluate the Tutte polynomial for graphs of bounded tree-width. We also consider the problem of computing the whole polynomial and show that there is a polynomial time algorithm to compute the polynomial but that asymptotically it requires strictly more time than for evaluation.

In Chapter 3 we consider a polynomial, introduced by Oxley and Whittle in [OW93], that has a similar flavour to the Tutte polynomial. It is the rank generating function of a natural 2-polymatroid associated with a graph. We show that its complexity is almost exactly the same as that of the Tutte polynomial in that it is $\#P$-hard to evaluate at most fixed points but there is a polynomial time algorithm when the input graph is restricted to a class of graphs with bounded tree-width.

The final two chapters are joint work with Dominic Welsh. The first of these contains results of a different type. A paper of Schwärzler and Welsh [SW93] investigates how well the Jones polynomial determines whether a knot is the unknot. Any knot can be represented as a signed graph and the Reidemeister moves, which characterise equivalence of knots, can be expressed in terms of signed graphs. This leads to an equivalence relation on

the class of all graphs. The members of the equivalence classes that contain empty graphs represent graphs which are equivalent to a link whose components are unknots. All this leads us to consider a class $\mathcal{S}$ of unsigned graphs which can be obtained from an empty graph using the unsigned version of Reidemeister moves. We consider the containment relationship with other similarly defined classes of graph such as the delta wye graphs. We have an invariant based on the Tutte polynomial which determines to which, if any, empty graph, a particular graph is related. However there remain many open questions, for instance we cannot determine whether $\mathcal{S}$ contains all graphs although we believe not and further believe that none of the Petersen family belong to $\mathcal{S}$.

The final chapter introduces a new graph polynomial. Vassiliev invariants are knot invariants which have recently received much attention. They lead to weight systems on chord diagrams and the study of these is equivalent to the study of a polynomial on weighted graphs. This polynomial does not include all of the Tutte-Grothendieck invariants as specialisations but with a slight change in the definition we obtain a polynomial W which does include all Tutte-Grothendieck invariants. W has some similarities with the Tutte polynomial in that it is defined with a deletion / contraction relation. We give three states model expansions for W and a Recipe Theorem analogous to that for the Tutte polynomial, see [OW79]. The special case when all weights are one, leads to a polynomial U on unweighted graphs, which contains a very wide range of invariants, for instance, the Tutte polynomial, the polymatroid polynomial of Oxley and Whittle [OW93] and a symmetric function generalisation of the chromatic polynomial introduced by Stanley [Sta95]. The chapter ends with a discussion of complexity in which we prove hardness results for very restricted classes of graphs such as stars and complete graphs.

Chapter 1

Graphs, complexity and the Tutte polynomial

1.1 Introduction

In this first chapter we introduce the topics that recur throughout the thesis, namely graph theory, complexity and the Tutte polynomial. We also give some short results that illustrate these concepts. In [GS96], various graph invariants are shown to be evaluations of the Tutte polynomial using depth first search. We give shorter proofs of two of these results in Section 1.5. In Section 1.6, which is essentially the contents of [Nob96], we answer a question raised by Kleinschmidt and Onn in [KO95], and investigate the complexity of recognising those simplicial complexes which are partitionable. A result from [MSS+95] concerning the Tutte polynomial, leads to an interesting equation involving the number of forests of a graph. In the final section we give a short proof showing that this equation holds and demonstrate how it extends to any simplicial complex.

1.2 Graphs and matroids

We assume familiarity with the basic ideas of graph theory. For more information see [Wil72]. All our graphs are allowed to have loops and multiple edges unless stated otherwise. The vertex set and edge set of a graph G are denoted respectively by $V(G)$ and $E(G)$ or just V and E. A *parallel class* is a maximal set of edges all of which are in parallel, that is they have the same endpoints. The *multiplicity* of an edge e, denoted by $m(e)$, is the size of the parallel class containing e. A graph is *simple* if it contains no loops or multiple edges.

A *subgraph* of $G = (V, E)$ is a graph with vertex set $V' \subseteq V$ and edge set $E' \subseteq E$. A subgraph (V', E') is *spanning* if $V' = V$. Given a set A of edges, $G \setminus A$ denotes the graph formed from G by *deleting* all the edges in A. The *deletion* of a single edge e is denoted by G'_e. Similarly the *contraction* of a set A of edges of G is denoted by G/A and the *contraction* of a single edge e is written as G''_e. H is a *minor* of G if it can be obtained from G by deleting and contracting edges and deleting vertices. The *restriction* of G to a set A of edges, $G|A$ is formed by deleting from G all edges except those contained in A. Given a set U of vertices, the graph $G : U$ has vertex set U and edge set consisting of those edges of G with both endpoints in U. Such a graph is said to be *induced* by U. Similarly given a set A of edges, the vertices of $G : A$ are all vertices which are incident with an edge in A and the edge set of $G : A$ is A. Hence $G : A$ is formed from $G|A$ by deleting any isolated vertices. The number of connected components of G is denoted by $k(G)$. The *rank* of a set A of edges, which is denoted by $r(A)$, is given by $r(A) = |V(G)| - k(G|A)$.

A graph is a *tree* if it is connected and contains no cycles. Given a graph

G, a set A of edges is a *tree* if $G|A$ is connected and contains no cycles. A graph is a *forest* if it contains no cycles or equivalently if its connected components are trees. Similarly, given a graph G, a set A of edges is a *forest* if $G|A$ is a forest.

Matroids were introduced in the 1930s in an attempt to generalise linear independence in vector spaces. They are also closely related to graphs and indeed for any graph it is possible to construct a matroid in a natural way. We only consider matroids in one place and then only briefly, so here we give little more than the definition and explain the connection between graphs and matroids. A *matroid* M is a pair (E, r) where E is a finite set, and r is an integer valued *rank function* on 2^E such that whenever A, B are subsets of E

1. $0 \leq r(A) \leq |A|$,
2. $A \subseteq B \Rightarrow r(A) \leq r(B)$,
3. $r(A \cup B) + r(A \cap B) \leq r(A) + r(B)$.

An independent set of a matroid (E, r) is a subset A of E such that $r(A) = |A|$. It follows easily from the definition that any subset of an independent set is itself independent. If G is a graph and r is its rank function then (E, r) is a matroid. Any matroid which is isomorphic to a matroid arising from a graph in this way is called *graphic*. The independent sets of a graphic matroid are the forests of the underlying graph. For a detailed introduction to matroids see either [Oxl92] or [Wel76].

1.3 Complexity

We give only a brief introduction to complexity theory. For more information see [GJ79] and [Wel93].

A *polynomial time* algorithm is one for which there exists a polynomial p such that for any input with size n the algorithm runs in time at most $p(n)$.

Much of complexity concerns decision problems and determining whether a decision problem has a polynomial time algorithm. We define a *language* to be a set of finite strings over a finite alphabet. Given an alphabet Σ, the set of strings of length n is denoted by Σ^n and the set of all finite strings from Σ is denoted by Σ^*. The class P is the set of languages that can be recognised in polynomial time by a deterministic Turing machine.

Often we will be interested in problems where we need more than just a yes / no answer. To this end we define FP to be the class of all functions that are computable in polynomial time. Most of the time we will not distinguish carefully between decision problems and more general problems and just refer to problems which can be solved by polynomial time algorithms.

Given a particular problem we will be interested in relating its complexity to another problem whose complexity is known. To this end we define the following two notions of reducibility between languages or problems.

A language L_1 is *polynomially reducible* to a language L_2, written as $L_1 \propto L_2$ if there exists a polynomially computable function f such that

$$x \in L_1 \Leftrightarrow f(x) \in L_2 \qquad \text{whenever } x \in \Sigma^*.$$

An apparently weaker form of reduction is the following. A problem π_1 is *Turing reducible* to a problem π_2 if there exists a deterministic Turing machine which solves π_1 in polynomial time using an oracle for π_2 which returns an answer in unit time. We write this as $\pi_1 \propto_T \pi_2$.

We now move on to the set of languages that are accepted by a non-deterministic Turing machine. NP is the set of languages L for which there exists a polynomially computable relation R, and polynomial p such that the following hold:

1. If $x \in L$ then $(x, y) \in R$ for at least one $y \in \sum^{p(|x|)}$.
2. If $x \notin L$ then $(x, y) \notin R$ for any $y \in \sum^{p(|x|)}$.

A string y such that $(x, y) \in R$ is called a *witness* or *certificate* for x. An easy observation is that $P \subseteq NP$.

A well-studied class of problems are the "hardest" members in NP, the NP-complete problems. A language $L_1 \in NP$ is *NP-complete* if for any $L_2 \in NP$, $L_2 \propto L_1$. A typical NP-complete problem is the language of 3-colourable graphs. A slightly weaker notion is that of NP-hardness. A problem π is *NP-hard* if for any $L \in NP$, $L \propto_T \pi$. A problem which is NP-hard does not have to be a member of NP.

In this thesis we will be more concerned with the complexity class $\#P$, the counting analogue of NP. We define $\#P$ to be the class of functions that count the number of witnesses of a language $L \in NP$ with respect to a particular binary relation R. For any language L there will be infinitely many binary relations which exhibit membership of NP but usually there is an obvious natural one which we will assume is used.

There is a notion of hardness corresponding to $\#P$. A function f is *$\#P$-hard* if for any $g \in \#P$, $g \propto_T f$. A polynomial reduction showing that a language is NP-complete will often easily lead to a proof of $\#P$-hardness for the counting version of the language providing the polynomial reduction preserves the number of witnesses, a *parsimonious* reduction, or for instance multiplies them by a constant factor, a *weakly parsimonious* reduction.

1.4 The Tutte polynomial

The majority of this thesis is concerned with graph polynomials. One polynomial, the Tutte polynomial, occurs many times here and is extremely well-studied in general. The *Tutte Polynomial*, of a graph G, is the two variable polynomial given by

$$T(G;x,y) = \sum_{A \subseteq E} (x-1)^{r(E)-r(A)}(y-1)^{|A|-r(A)}.$$

The Tutte polynomial can be calculated recursively using the following:

1. If e is a loop then

$$T(G;x,y) = yT(G'_e;x,y). \tag{1.4.1}$$

2. If e is an isthmus then

$$T(G;x,y) = xT(G''_e;x,y). \tag{1.4.2}$$

3. Otherwise

$$T(G;x,y) = T(G'_e;x,y) + T(G''_e;x,y). \tag{1.4.3}$$

It is also easy to see that the definition implies that if G has connected components $G_1, G_2, \ldots, G_k$ then

$$T(G;x,y) = T(G_1;x,y)T(G_2;x,y)\cdots T(G_k;x,y). \tag{1.4.4}$$

The definition of the Tutte polynomial can be extended to matroids since it only depends on the rank function, but here we will almost exclusively focus on the Tutte polynomial of a graph.

The Tutte polynomial contains a great deal of information about a graph. For instance:

- At $(1,1)$, T counts the number of maximal forests of G, spanning trees if G is connected.
- At $(2,1)$, T counts the number of forests of G (or independent sets of a matroid).
- T is said to contain the chromatic polynomial, $P(G;\lambda)$, in that

$$P(G;\lambda) = \lambda^{k(G)}(-1)^{r(E)}T(G;1-\lambda,0). \tag{1.4.5}$$

- If G is connected then the all-terminal reliability $R(G;p)$ is given by

$$R(G;p) = (1-p)^{|E|-r(E)}p^{r(E)}T\left(G;1,\frac{1}{1-p}\right).$$

- The number of acyclic orientations of G, that is the number of ways of assigning a direction to each edge of G giving no directed cycles, is the evaluation of T at $(2,0)$.

The family of hyperbolae H_α defined by

$$H_\alpha = \{(x,y) : (x-1)(y-1) = \alpha\}$$

seems to play a special role in the theory, for instance the partition function of the Ising model is an evaluation along H_2, and along H_q, for any positive integer q, T specialises to the partition function of the q-state Potts model.

The following theorem, known as the Recipe Theorem, is useful in finding specialisations of T.

Theorem 1.4.6 (Oxley and Welsh [OW79]). *For any function f defined on all graphs and satisfying*

1. *$f(G) = 1$ if G has no edges,*

2. $f(G) = y_0 f(G'_e)$ *if* e *is a loop,*

3. $f(G) = x_0 f(G''_e)$ *if* e *is an isthmus,*

4. $f(G) = af(G'_e) + bf(G''_e)$ *if* e *is neither a loop nor an isthmus,*

we have

$$f(G) = a^{|E|-r(E)} b^{r(E)} T\left(G; \frac{x_0}{b}, \frac{y_0}{a}\right).$$

The following theorem, which is also useful, seems to be well known but not explicitly stated anywhere, although it is certainly used in [BO92].

Theorem 1.4.7. *Let* f *be a function defined on all graphs taking the value one on a graph with no edges and such that for any other graph* G *there is an edge* e *of* G *for which*

1. $f(G) = y_0 f(G'_e)$ *if* e *is a loop,*

2. $f(G) = x_0 f(G''_e)$ *if* e *is an isthmus,*

3. $f(G) = af(G'_e) + bf(G''_e)$ *if* e *is neither a loop nor an isthmus.*

Then

$$f(G) = a^{|E|-r(E)} b^{r(E)} T\left(G; \frac{x_0}{b}, \frac{y_0}{a}\right).$$

Proof. The proof is by an easy induction on the number of edges of G. The result is obviously true when G has no edges. Suppose G has at least one edge, so there is an edge e satisfying the conditions of the theorem. Assume first that e is neither a loop nor an isthmus and so $f(G) = af(G'_e) + bf(G''_e)$. Using induction

$$\begin{aligned} f(G) &= aa^{|E|-1-r(E)} b^{r(E)} T\left(G'_e; \frac{x_0}{b}, \frac{y_0}{a}\right) \\ &\quad + ba^{|E|-1-(r(E)-1)} b^{r(E)-1} T\left(G''_e; \frac{x_0}{b}, \frac{y_0}{a}\right) \\ &= a^{|E|-r(E)} b^{r(E)} T\left(G; \frac{x_0}{b}, \frac{y_0}{a}\right) \end{aligned}$$

The cases when e is a loop or an isthmus are similar. □

A whole host of other specialisations of T is contained in [BO92] and [Wel93].

1.4.1 The complexity of Tutte invariants

The complexity of evaluating T has been very well studied and it is known that for most fixed points (x, y) computing T is $\#P$-hard even when the class of input graphs is quite restricted, for example, Equation 1.4.5 shows that the number of 3-colourings of a graph, an invariant which is known to be $\#P$-hard, see [Lin86], can easily be computed from $T(G; -2, 0)$.

In contrast to this along H_1 the Tutte polynomial reduces to

$$T(G; x, y) = x^{|E|}(x-1)^{r(E)-|E|}$$

which is easy to compute and there are a few other special points for which there are polynomial time algorithms to evaluate T.

These results are summarised in the following theorems. The first result is due to Jaeger, Vertigan and Welsh [JVW90].

Theorem 1.4.8. *Evaluating the Tutte polynomial of a graph at any fixed point (a, b) of the rational plane is $\#P$-hard except when either*

1. *The point (a, b) lies on the hyperbola H_1,*
2. *The point (a, b) is one of the special points $(0, 0)$, $(-1, 0)$, $(0, -1)$, $(-1, -1)$, $(1, 1)$*

where there is a polynomial time algorithm to evaluate T.

This result has been strengthened by Vertigan and Welsh in [VW92], where the input graph is restricted to being a bipartite planar graph.

Theorem 1.4.9. *It is #P-hard to evaluate the Tutte polynomial of a bipartite planar graph at any fixed point (a, b) of the rational plane except when either*

1. *The point (a, b) lies on one of the hyperbolae H_1 and H_2,*
2. *The point (a, b) is one of the special points $(0, 0)$, $(-1, 0)$, $(0, -1)$, $(-1, -1)$, $(1, 1)$*

where there is a polynomial time algorithm to evaluate T.

1.5 Orientations and Subdigraphs

As the previous section shows there are a vast number of graph invariants which are evaluations of the Tutte polynomial. This section is intended to illustrate some of the techniques used to show that a graph invariant is an evaluation of T. In a recent paper, [GS96], Gessel and Sagan show that a wide range of graph invariants are evaluations of the Tutte polynomial. They prove their results using depth first search, however we give shorter proofs based on Theorem 1.4.7. We begin with some definitions, then state the results and their consequences, and finally give our shorter proofs.

An *orientation* of a graph G is a directed graph obtained from G by assigning to each edge, including loops, one of the two possible directions. A *suborientation* of G is a orientation of a spanning subgraph of G. A directed graph is acyclic if it contains no directed cycles. An oriented loop is considered to be a cycle.

Suppose we have a directed graph D and an ordering of its vertices. D is said to be *initially connected* if there is a directed path from the first vertex to each of the others. Any directed graph can be decomposed into *initially connected components* in the following way. The first component contains the

first vertex and any vertices to which there is a directed path from the first vertex. Delete these vertices and now the second component contains the first vertex remaining and any remaining vertices to which there is a directed path from the first remaining vertex and so on. We denote the number of initially connected components of a suborientation $\mathcal{O}$ by $k(\mathcal{O})$. Note that $k(\mathcal{O})$ depends on the ordering of the vertices. The size of an orientation $|\mathcal{O}|$ is the number of edges that are oriented in $\mathcal{O}$.

The following result is from [GS96].

Theorem 1.5.1.

$$\sum_{\mathcal{O}} x^{k(\mathcal{O})} y^{|\mathcal{O}|} = x^{k(G)} y^{r(E)} (1+y)^{|E|-r(E)} T\left(G; 1+x+\frac{x}{y}, \frac{1}{1+y}\right) \tag{1.5.2}$$

where the sum is over all acyclic suborientations of G.

In particular this result shows that $\sum_{\mathcal{O}} x^{k(\mathcal{O})} y^{|\mathcal{O}|}$ does not depend on the ordering of the vertices. In [GS96] the authors note the following consequences of this theorem. Setting $x = 1$ and $y = 1$ gives that the number of acyclic suborientations of G is $2^{|E|-r(E)} T(G; 3, 1/2)$ and dividing through by x and then setting $x = 0$ and $y = 1$ gives that the number of initially connected suborientations of G is $2^{|E|-r(E)} T(G; 1, 1/2)$. If we divide Equation 1.5.2 by $y^{|E|}$ and let $y \to \infty$ we get

$$\sum_{|\mathcal{O}|=|E|} x^{k(\mathcal{O})} = x^{k(G)} T(G; 1+x, 0).$$

Substituting $x = 1$ gives that $T(G; 2, 0)$ is the number of acyclic orientations of G, a result first proved by Stanley [Sta73]. Dividing through by x and then substituting $x = 0$ shows that for a connected graph G, $T(G; 1, 0)$ is the number of initially connected acyclic orientations of G. This evaluation was found by Greene and Zaslavsky [GZ83].

The second result from [GS96], considered here, concerns subdigraphs, orientations of graphs which have a similar flavour to suborientations. Given a graph G, a *subdigraph* of G is a directed graph D that contains up to one copy of each orientation of every edge of G. Thus in contrast with suborientations, an edge may be oriented in both directions at the same time, so in constructing a subdigraph from a graph there are four possibilities for each edge including loops since an edge can receive no orientation, an orientation in either direction or both orientations. The size of a subdigraph $|D|$ is the total number of orientations which it contains. The following theorem from [GS96], concerning subdigraphs, is similar to Theorem 1.5.1.

Theorem 1.5.3.

$$\sum_D x^{k(D)}y^{|D|} = x^{k(G)}y^{r(E)}(1+y)^{|E|}T\left(G; 1+\frac{x}{y}, 1+y\right) \tag{1.5.4}$$

where the summation is over all subdigraphs of G.

As a special case, if G is connected then dividing Equation 1.5.4 by x, gives that the number of initially connected subdigraphs of G is $2^{|E|}T(G;1,2)$.

The proofs of Theorems 1.5.1 and 1.5.3 are similar and need Lemma 1.5.5 below. We will only prove them for connected graphs since the generalisation to all graphs is straightforward using Equation 1.4.4.

Lemma 1.5.5. *Given a graph and an ordering of the vertices, $\sum_{\mathcal{O}} x^{k(\mathcal{O})-1}y^{|\mathcal{O}|}$ is independent of the choice of ordering of the vertices.*

Proof. Suppose that λ_1, λ_2 are two orderings of the vertices of V. We will show that there is a bijection between the acyclic suborientations of G with ordering λ_1 and the acyclic suborientations of G with ordering λ_2 which preserves the number of oriented edges and the partition of V induced by

initially connected components. Choose an acyclic suborientation $\mathcal{O}_1$ of G and let $C_1, C_2, \ldots, C_l$ be the initially connected components of V with respect to λ_1. We show how to construct a suborientation $\mathcal{O}_2$ of G so that $C_1, C_2, \ldots, C_l$ are the initially connected components with respect to λ_2. We will do this by reversing the orientation of some edges so the number of oriented edges is kept the same.

Consider an oriented edge e between two vertices in different components. There is a natural ordering on $C_1, C_2, \ldots, C_l$ induced by the first vertex in each component. Suppose e is oriented from a vertex in C_a to a vertex in C_b then C_b must precede C_a in the ordering of the components induced by λ_1. If C_a precedes C_b in the ordering of the components induced by λ_2 then reverse the orientation of e.

Now consider the edges within a component C. Let v_1 be the first vertex of C under λ_1 and v_2 be the first vertex of C under λ_2. If $v_1 \neq v_2$ then reverse the orientation of every edge that is on a directed path from v_1 to v_2.

It is not too difficult to check that the initially connected components of $\mathcal{O}_2$ under λ_2 are the same as those of $\mathcal{O}_1$ under λ_1. Consider an initially connected component C of $\mathcal{O}_1$ under λ_1. Let v_2 be the first vertex of this component under λ_2. In $\mathcal{O}_2$ there is a directed path from v_2 to every vertex of C. Also every oriented edge between components is oriented from a later component to an earlier one. No directed cycles are formed. This follows because all edges between components are oriented towards the earlier component in the induced ordering and so any cycles must be within a component. Suppose there is a cycle within a component C. Within C there are edges which have had their orientations reversed. These all lie on paths from v_2 to v_1. Let V_1 be the set of vertices of C which lie on paths from v_2 to v_1. Any cycle must contain both edges whose orientation has been reversed

and edges whose orientation has not. Choose an edge e which has not been reversed but is oriented away from a vertex u_1 of V_1 and is contained in a cycle. Follow the cycle until a vertex u_2 of V_1 is reached. Now in $\mathcal{O}_1$ there are paths from v_1 to u_1, u_1 to u_2 and u_2 to v_2. The union of the edges of these paths either includes a cycle or is a path from v_1 to v_2 using edges which were not reversed giving a contradiction.

To see that this is a bijection note that if we start with λ_2 and let λ_1 be the new ordering that is we interchange λ_1 and λ_2 then the above procedure will map $\mathcal{O}_2$ to $\mathcal{O}_1$. □

Proof of Theorem 1.5.1: Let $f(G) = \sum_{\mathcal{O}} x^{k(\mathcal{O})-1} y^{|\mathcal{O}|}$ where the sum is over all acyclic suborientations. We show that f satisfies Theorem 1.4.7 with $a = 1+y$, $b = y$, $x_0 = x+y+xy$ and $y_0 = 1$. The theorem then follows immediately. Suppose G has an edge that is not a loop. Using Lemma 1.5.5 we can order the vertices arbitrarily so that the first two vertices v_1 and v_2 are adjacent. We now let e be the edge $\{v_1, v_2\}$ and show that the conditions of Theorem 1.4.7 hold.

If e is an isthmus then the acyclic suborientations of G have three types, those with e not oriented, those with e oriented from v_1 to v_2 and those with e oriented from v_2 to v_1. We show how these suborientations are related to those of G''_e where we order vertices so that the vertex coming from identifying v_1 and v_2 comes before all the others which are ordered the same as in G.

If e is not given an orientation then the orientations are the same as those of G''_e but with one extra initially connected component. If e is oriented from v_2 to v_1 then they are the same as those of G''_e but with an extra edge and an extra component and finally if e is oriented from v_1 to v_2 they are the same as those of G''_e but with an extra edge. Hence $x_0 = x + y + xy$.

The case where e is neither a loop or an isthmus is a little more com-

plicated. Suppose we orient a set A of edges of G so that e is unoriented and there are no directed cycles in G. We show how the three possible suborientations obtained by making each possible choice about e correspond to suborientations of G'_e and G''_e with the edges of A oriented as above and the other edges unoriented. The ordering on G'_e is chosen to be the same as that of G and the ordering on G''_e has the vertex formed from v_1 and v_2 coming first and then the others in the same order as in G.

If we do not orient e then this suborientation corresponds to a suborientation of G'_e in the obvious way.

Suppose that once we have chosen the orientations of the edges in A we can now only orient e in one direction and remain acyclic. Then there must be a directed path from v_1 to v_2 or vice versa. Putting an orientation on e without forming a cycle gives a suborientation with one more edge but no more initially connected components than the suborientation on G'_e with just the edges of A oriented as above. Note that G''_e has a directed cycle and so orienting the edges of A as above does not give a suborientation of G''_e.

In the other case there is no directed path from v_1 to v_2 or vice versa and so orienting e from v_2 to v_1 gives a suborientation with one more edge but no more initially connected components than the suborientation of G'_e with the edges of A oriented as above, and orienting e from v_1 to v_2 gives a suborientation with one more edge and the same number of components as the suborientation of G''_e with edges of A oriented as above. Hence $a = 1 + y$ and $b = y$.

If G consists entirely of loops then $f(G) = 1$ and so $y_0 = 1$. □We now prove a version of Lemma 1.5.5 for subdigraphs.

Lemma 1.5.6. *Given a graph and an ordering of the vertices,* $\sum_D x^{k(D)-1}y^{|D|}$ *is independent of the choice of ordering of the vertices.*

Proof. The proof is almost exactly the same as the proof of Lemma 1.5.5 except here we consider subdigraphs we allow edges to have two orientations and we also allow cycles. We establish a bijection as above by reordering edges in almost exactly the same way. We never alter an edge which has both orientations. When we order the edges within a component C, we do nothing if, in the notation of the proof of Lemma 1.5.5, there is already a directed path from w_1 to v_1. This is enough to ensure that the initially connected components remain the same. □

Proof of Theorem 1.5.3: Let $f(G) = \sum_{\mathcal{D}} x^{k(\mathcal{D})-1} y^{|\mathcal{D}|}$ where the sum is over all subdigraphs. We show that f satisfies Theorem 1.4.7 with $a = 1+y$, $b = y(1+y)$, $x_0 = (x+y)(1+y)$ and $y_0 = (1+y)^2$. Suppose that G has an edge that is not a loop. Using Lemma 1.5.6 we can order the vertices of G so that the first two vertices v_1 and v_2 are adjacent. As in the previous proof we let e be the edge $\{v_1, v_2\}$ and show that the conditions of Theorem 1.4.7 hold.

If e is an isthmus then we relate the subdigraphs of G to those of G/e with the vertex ordering defined so that the vertex formed by identifying v_1 and v_2 comes first and then the other vertices in the same way as in G. A subdigraph of G''_e can be extended to a subdigraph of G in four ways with the number of components increasing by one if e is not oriented at all or oriented just from v_2 to v_1 and otherwise staying the same. Therefore $x_0 = (x+y)(1+y)$.

Suppose that e is not a loop or an isthmus. Take a subdigraph of G. If e is not oriented or only oriented from v_2 to v_1 then this subdigraph corresponds in the obvious way by restriction to a subdigraph of G'_e with the same number of components but with possibly one more edge. If e is oriented from v_1 to v_2 or in both directions then the subdigraph corresponds in the same way, by restriction, to a subdigraph of G''_e with the same number of components

but one or two more edges. Hence $a = 1 + y$ and $b = y(1 + y)$.

If G consists entirely of loops then for any e it is easy to see that $f(G) = (1+y)^2 f(G'_e)$ because there are four ways of orienting a loop (including doing nothing) and however we orient the loop makes no difference to the number of initially connected components. Hence $y_0 = (1 + y)^2$. □

1.6 Partitioning a simplicial complex is in NP

This section is essentially the contents of [Nob96]. In [KO95], Kleinschmidt and Onn raise a number of questions concerning the complexity of various problems involving simplicial complexes. We answer one of those questions here.

1.6.1 Definitions

A *simplicial complex*, or just *complex*, consists of a pair $(E, \mathcal{F})$ where E is a finite set and $\mathcal{F}$ is a family of subsets of E such that if $V \in \mathcal{F}$ and $U \subseteq V$ then $U \in \mathcal{F}$. Inclusionwise maximal members of $\mathcal{F}$ are called *facets*. The *dimension* of a complex is the cardinality of its largest facet. A complex with facets $F_1, \ldots, F_n$ is said to be *partitionable* if there exists a sequence, $\phi(F_1), \ldots, \phi(F_n)$, of subsets of E such that for any $U \in \mathcal{F}$ there is a unique facet F satisfying $\phi(F) \subseteq U \subseteq F$. In [KO95] the problem of determining the complexity of recognising a partitionable complex was discussed. It is easy to see that for any fixed d, the language of partitionable complexes of dimension at most d is a member of the complexity class NP. Our result removes the d-dimensional restriction, that is, it shows that the language of partitionable complexes is in NP, and thus answers a question raised in [KO95].

In general membership in P remains open. It is easy to see that partitionable complexes of dimension at most two can be recognised in polynomial time. However the language of partitionable complexes of dimension at most d is not known to be in P for any fixed $d \geq 3$ and is conjectured in [KO95] to be NP-complete for some d. These problems remain open.

1.6.2 Results

Theorem 1.6.1. *Given any simplicial complex $(E, \mathcal{F})$ with facets $F_1, \ldots, F_n$ satisfying $|F_i| \leq d$ and a sequence, $\phi(F_1), \ldots \phi(F_n)$, of subsets of E, we can verify that the intervals $[\phi(F_i), F_i]$ form a partition of $\mathcal{F}$, in time $O(n^2 d)$.*

Corollary 1.6.2. *The language of simplicial complexes which are partitionable is a member of NP.*

Proof of Corollary: Given $F_1, \ldots, F_n$, the sequence $\phi(F_1), \ldots, \phi(F_n)$ is a suitable witness which we can verify in polynomial time. □

Proof of Theorem: Algorithm 1 verifies that the intervals $[\phi(F_i), F_i]$ form a partition of $\mathcal{F}$.

We now show that this algorithm does what we claim. It is clear that the intervals $[\phi(F_i), F_i]$ form an interval partition of $\mathcal{F}$ if and only if the following three conditions hold:

1. $\phi(F_i) \subseteq F_i$ for all i.
2. $[\phi(F_i), F_i] \cap [\phi(F_j), F_j] = \emptyset$ for all i and j with $i \neq j$.
3. Any subset of a facet is contained in some interval.

Condition (2) is equivalent to

$$\phi(F_i) \cup \phi(F_j) \not\subseteq F_i \cap F_j$$

Algorithm 1 Verifying an Interval Partition

Input: Intervals $[\phi(F_i), F_i]$

for $i = 1$ to n **do**
  if $\phi(F_i) \not\subseteq F_i$ **then**
    output: NO
  end if
end for
for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    if $i \neq j$ and $\phi(F_i) \cup \phi(F_j) \subseteq F_i \cap F_j$ **then**
      output: NO
    end if
  end for
end for
for $i = 1$ to n **do**
  if $2^{|F_i|} \neq \sum_{j:\phi(F_j)\subseteq F_i} 2^{|(F_i \cap F_j) - \phi(F_j)|}$ **then**
    output: NO
  end if
end for

Output: YES

for all $i \neq j$.

The intersection of $[\emptyset, F_i]$ and $[\phi(F_j), F_j]$ is $[\phi(F_j), F_i \cap F_j]$ which is empty if $\phi(F_j) \not\subseteq F_i$ and otherwise has cardinality $2^{|(F_i \cap F_j) - \phi(F_j)|}$. So if condition (2) holds then condition (3) is equivalent to

$$2^{|F_i|} = \sum_{j:\phi(F_j) \subseteq F_i} 2^{|(F_i \cap F_j) - \phi(F_j)|}$$

for all i.

It is easy to see that given any $\{F_i\}$ and $\{\phi(F_i)\}$ the algorithm runs in time $O(n^2 d)$. □

1.7 The average size of a member of a complex

In a recent paper [MSS+95] an equation involving the Tutte polynomial is proved. Substituting for the variables in the equation gives a result concerning the average size of a forest. We give a simple proof of a slight strengthening of this result and show how it can be extended to any simplicial complex.

Given a matroid $M = (E, r)$ and $A \subseteq E$ the *closure* of A, denoted by $\bar{A}$, is given by

$$\bar{A} = \{e \in E : r(A \cup e) = r(A)\}.$$

Note that $A \subseteq \bar{A}$. In a matroid, a *loop* e is a member of E such that $r(e) = 0$. The following is the main result from [MSS+95].

Theorem 1.7.1. *If M is a loop free matroid on E, then*

$$|E|\, T(M; x, y) = \sum_{A \subseteq E} \left\{ x|\bar{A}| + y(1-x)|\bar{A} \setminus A| \right\} (x-1)^{r(E)-r(A)} (y-1)^{|A|-r(A)}.$$

Substituting $x = 2$ and $y = 1$ gives a corollary about the independent sets of M.

Corollary 1.7.2. *If M is a loop free matroid on E and $\mathcal{I}$ is the set of independent sets of M then*

$$|E||\mathcal{I}| = \sum_{I \in \mathcal{I}} (|I| + |\bar{I}|).$$

It is this corollary which we consider here. By dividing through by $|\mathcal{I}|$ we can state the result in terms of the average size of an independent set and its closure

$$|E| = \boldsymbol{E}[|I|] + \boldsymbol{E}[|\bar{I}|].$$

Our main result is a generalisation of this corollary. Given a simplicial complex $(E, \mathcal{F})$ we define the *closure* of $F \in \mathcal{F}$ which we denote by $\bar{F}$ by

$$\bar{F} = \{e \in E : F \cup e \notin \mathcal{F}\} \cup F.$$

Note that the independent sets of a matroid form a simplicial complex and the notion of closure that we have defined for simplicial complexes is the same as that for matroids.

Proposition 1.7.3. *Let $(E, \mathcal{F})$ be a simplicial complex such that every element of E is contained in some member of $\mathcal{F}$. Let e be a member of E and F be a member of $\mathcal{F}$ chosen uniformly at random. Then*

$$\Pr(e \in F) + \Pr(e \in \bar{F}) = 1.$$

Corollary 1.7.2 can be deduced from this result.

Proof of Corollary 1.7.2: Since M is loop free, the independent sets of M form a simplicial complex $(E, \mathcal{I})$ such that every member of E is contained

in some member of $\mathcal{I}$. Using Proposition 1.7.3 we get

$$\sum_{e \in E} (\Pr(e \in I) + \Pr(e \in \bar{I})) = |E|$$

where I is an independent set chosen uniformly at random. Hence

$$\boldsymbol{E}[|I|] + \boldsymbol{E}[|\bar{I}|] = |E|.$$

□

Proof of Proposition Fix $e \in E$ and let $F \in \mathcal{F}$ such that $e \notin F$. Now either $F \cup e \in \mathcal{F}$ or $e \in \bar{F}$. Hence

$$\begin{aligned} &|\{F \cup e : F \cup e \in \mathcal{F},\ e \notin F\}| + |\{F : F \in \mathcal{F},\ e \notin F \text{ but } e \in \bar{F}\}| \\ &\qquad = |\{F : F \in \mathcal{F},\ e \notin F\}|. \end{aligned}$$

The first term is just the number of members of $\mathcal{F}$ containing e and so adding the members of $\mathcal{F}$ containing e to each side we get

$$|\{F : F \in \mathcal{F},\ e \in F\}| + |\{F : F \in \mathcal{F},\ e \in \bar{F}\}| = |\{F : F \in \mathcal{F}\}|.$$

Therefore if F is a member of $\mathcal{F}$ chosen uniformly at random then

$$\Pr(e \in F) + \Pr(e \in \bar{F}) = 1.$$

□

Chapter 2

Evaluating the Tutte Polynomial for Graphs of Bounded Tree-Width

2.1 Introduction and Notation

As we have seen, at most fixed points, evaluating the Tutte polynomial for general graphs is $\#P$-hard. Following from this, a natural problem is to find classes of graphs for which there is a polynomial time algorithm to evaluate T. It is easy to see that the Tutte polynomial of any tree can be computed quickly because it is just $x^{|V|-1}$. The class of graphs which we consider here, the graphs with bounded tree-width, are a generalisation of trees. We begin by introducing tree-width.

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $\big(\{X_i | i \in I\}, T = (I, F)\big)$ where $\{X_i | i \in I\}$ is a family of subsets of V, one for each vertex of T, and T is a tree such that

- $\bigcup_{i \in I} X_i = V$.
- for all edges $(v, w) \in E$, there exists $i \in I$ such that $v \in X_i$ and $w \in X_i$.
- for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *tree-width* of a tree-decomposition is $\max_{i \in I} |X_i| - 1$. The *tree-width* of a graph G is the minimum tree-width over all possible tree-decompositions of G. If we give T a root then we can define $Y_i = \{v \in X_j | j = i \text{ or } j \text{ is a descendant of } i\}$.

Many well-studied classes of graphs have bounded tree-width, for instance series-parallel networks are the graphs with tree-width at most two. A large class of graph problems which are thought to be intractable can be solved when the input is restricted to graphs with tree-width at most a fixed constant k. For example, the NP-complete problems, 3-Colouring and Hamiltonian Circuit can be solved in linear time for graphs of bounded tree-width. See [Bod93] for more information on tree-width.

In our algorithm we assume that for some fixed k we are given a graph, G, of tree-width $\leq k$. We first have to compute a tree-decomposition of width $\leq k$ such that $|I| \leq 2|V|$ and T is a binary tree. Let $f(k) = k^5 \cdot (2k+1)^{(2k+1)-2} \cdot ((2(2k+1)+3)^{2(2k+1)+3} \cdot (\frac{8}{3} \cdot 2^{2k+2})^{2(2k+1)+3})^{2(2k+1)-1}$. The algorithm given in [Bod96] will, in time $O(f(k) \cdot |V|)$, produce a tree-decomposition $\big(\{X_i | i \in I'\}, T' = (I', F')\big)$ with $|I'| \leq |V|$ and from this it is easy to construct, in time $O(k|V|)$, a tree-decomposition with $|I| \leq 2|V|$ and T a rooted binary tree.

Our main result is,

Theorem 2.1.1. *For any fixed k, there exists an algorithm $\mathcal{A}$, which will input any graph G, with tree-width at most k, and rational numbers $x = p_x/q_x$*

and $y = p_y/q_y$ *where both* p_x *and* q_x*, and* p_y *and* q_y *are coprime, and evaluate the Tutte polynomial,* $T(G; x, y)$*, using at most* $O(f(k) \cdot (n + M) \cdot (n + m) \cdot \log(n+m) \cdot \log\log(n+m) \cdot l \log l \cdot \log\log l)$ *operations, where* $n = |V|$*,* $m = |E|$*,* $l = \log(|p_x| + |q_x| + |p_y| + |q_y|)$ *and* M *is the largest size of a parallel class of edges.*

Our result extends the work of Arnborg and Proskurowski [AP89] who gave a linear time algorithm to calculate the reliability of a graph and also that of Oxley and Welsh [OW92] who gave a polynomial time algorithm for evaluating the Tutte polynomial for graphs of restricted width, a class which includes series-parallel networks, that is those graphs with tree-width at most two. This contrasts with the situation for general graphs where, as we pointed out earlier, Jaeger, Vertigan and Welsh [JVW90] have shown that the Tutte Polynomial is $\#P$-hard to evaluate except at a few special points and along one special curve, a result which can be extended to bipartite planar graphs [VW92].

Suppose we write $T(G; x, y) = \sum_{i,j} t_{ij} x^i y^j$. A problem motivated by our result is to find an algorithm that will input any graph with tree-width at most k and output a list of the coefficients t_{ij}. We show that an algorithm which does this must have running time $\Omega(|V|^3)$.

We now need to define some notation concerning partitions of a set. We denote the set of partitions of X by $\Pi(X)$ and let $\#\pi$ be the size of a partition, that is the number of non-empty blocks which make up π. Throughout this chapter, the partitions, to which we refer, are partitions of the vertex set of a graph and so if π is a partition of X we say that X is the vertex set of π and refer to elements of X as the vertices of π. We use I_X to denote the partition with vertex set X consisting entirely of singleton blocks. If V is the vertex set of π then for $W \subseteq V$ the *restriction* to W, $\pi|W$, is formed by

deleting all elements in the partition not contained in W and then deleting any empty blocks which are formed. If π_1 and π_2 have the same vertex set, X, then we define their *join* $\pi_1 \vee \pi_2$ to be the partition of X whose blocks are minimal sets such that if u and v are in different blocks of $\pi_1 \vee \pi_2$ then u and v are in different blocks of π_1 and π_2. In other words the operation $\vee$ corresponds to join in the partition lattice. More generally if π_1 and π_2 have vertex sets X_1 and X_2 respectively, we form their join by first adding the vertices of $X_2 \setminus X_1$ to π_1 as singleton blocks giving π_1' and similarly forming π_2' by adding the vertices of $X_1 \setminus X_2$ to π_2 as singleton blocks and finally computing $\pi_1' \vee \pi_2'$. If $A \subseteq E$ and $X \subseteq V$ then $\Pi_X(A)$ denotes the restriction to X of the partition π of V, in which the blocks of π correspond to connected components of $G|A$.

2.2 The Algorithm

From here up until the end of Section 2.2.3 we will assume that we are evaluating T at a point (x, y) with $x \neq 1$. Later we show how to compute T along the line $x = 1$; the method we give here does not work if $x = 1$ because it involves dividing by $x-1$. We first give an informal illustration of the idea behind the algorithm. Suppose we have a graph $G = (V_1 \cup V_2, E_1 \cup E_2)$ with $V_1 \cap V_2 = X$, $E_1 \cap E_2 = \emptyset$ and such that any edge in E_i has both endpoints in V_i. We refer to a set X which occurs in this way as an *intersecting set*. Now suppose that for any partition, π, of X and any i and j we know the number of subsets A of E_1 (E_2) with rank i and cardinality j and satisfying $\Pi_X(A) = \pi$. We denote this number by $N_1(\pi, i, j)$ ($N_2(\pi, i, j)$). Using this information we can calculate $N(\pi, i, j)$, the number of subsets A of $E_1 \cup E_2$ with rank i, cardinality j and satisfying $\Pi_X(A) = \pi$. This is true because if

$A = A_1 \cup A_2$ where $A_1 \subseteq E_1$ and $A_2 \subseteq E_2$ the rank of A depends only on the rank of A_1 and A_2, and on $\Pi_X(A_1)$ and $\Pi_X(A_2)$ but not on the actual edges of A_1 and A_2. More precisely,

$$r(A) = r(A_1) + r(A_2) - |X| - \#\Pi_X(A) + \#\Pi_X(A_1) + \#\Pi_X(A_2).$$

This means that if $\pi_1 \vee \pi_2 = \pi$ then the number of sets A contributing to $N(\pi, i, j)$ and satisfying $\Pi_X(A \cap E_1) = \pi_1$ and $\Pi_X(A \cap E_2) = \pi_2$ is

$$\sum_{i_1=0}^{i+I} \sum_{j_1=0}^{j} N_1(\pi_1, i_1, j_1) N_2(\pi_2, i + I - i_1, j - j_1),$$

where $I = |X| + \#\pi - \#\pi_1 - \#\pi_2$ and so

$$N(\pi, i, j) = \sum_{\substack{(\pi_1, \pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i_1=0}^{i+I} \sum_{j_1=0}^{j} N_1(\pi_1, i_1, j_1) N_2(\pi_2, i + I - i_1, j - j_1),$$

where again $I = |X| + \#\pi - \#\pi_1 - \#\pi_2$.

Let x and y be fixed with $x \neq 1$ and suppose that rather than knowing $N_1(\pi, i, j)$ and $N_2(\pi, i, j)$ explicitly we know the evaluation at the point (x, y) of certain polynomials similar to the Tutte Polynomial; that is we are given

$$\begin{aligned} t_1(\pi) &= \sum_{A \subseteq E_1 : \Pi_X(A) = \pi} (x-1)^{-r(A)} (y-1)^{|A| - r(A)} \\ &= \sum_{i,j} N_1(\pi, i, j) (x-1)^{-i} (y-1)^{j-i} \end{aligned}$$

and

$$\begin{aligned} t_2(\pi) &= \sum_{A \subseteq E_2 : \Pi_X(A) = \pi} (x-1)^{-r(A)} (y-1)^{|A| - r(A)} \\ &= \sum_{i,j} N_2(\pi, i, j) (x-1)^{-i} (y-1)^{j-i} \end{aligned}$$

and we wish to compute for each $\pi \in \Pi(X)$

$$t(\pi) = \sum_{A \subseteq E_1 \cup E_2 : \Pi_X(A) = \pi} (x-1)^{-r(A)} (y-1)^{|A|-r(A)}.$$

Now setting $I = I(\pi_1, \pi_2) = |X| + \#\pi - \#\pi_1 - \#\pi_2$ we have

$$\begin{aligned}
t(\pi) &= \sum_{i,j} N(\pi, i, j)(x-1)^{-i}(y-1)^{j-i} \\
&= \sum_{i,j} N(\pi, i, j)\big((x-1)(y-1)\big)^{-i}(y-1)^{j} \\
&= \sum_{i,j} \sum_{\substack{(\pi_1,\pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i_1, j_1} \Big[N_1(\pi_1, i_1, j_1) N_2(\pi_2, i + I - i_1, j - j_1) \\
&\qquad \cdot \big((x-1)(y-1)\big)^{-i}(y-1)^{j} \Big] \\
&= \sum_{\substack{(\pi_1,\pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i,j,i_1,j_1} \Big[N_1(\pi_1, i_1, j_1)\big((x-1)(y-1)\big)^{-i_1}(y-1)^{j_1} \\
&\qquad \cdot N_2(\pi_2, i + I - i_1, j - j_1)\big((x-1)(y-1)\big)^{(-i-I+i_1)}(y-1)^{j-j_1} \\
&\qquad \cdot \big((x-1)(y-1)\big)^{I} \Big] \\
&= \sum_{\substack{(\pi_1,\pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i_1,j_1,i_2,j_2} \Big[N_1(\pi_1, i_1, j_1)\big((x-1)(y-1)\big)^{-i_1}(y-1)^{j_1} \\
&\qquad \cdot N_2(\pi_2, i_2, j_2)\big((x-1)(y-1)\big)^{-i_2}(y-1)^{j_2} \\
&\qquad \cdot \big((x-1)(y-1)\big)^{I} \Big] \\
&= \sum_{\substack{(\pi_1,\pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} t_1(\pi_1) t_2(\pi_2)\big((x-1)(y-1)\big)^{(|X|+\#\pi-\#\pi_1-\#\pi_2)}. \qquad (2.2.1)
\end{aligned}$$

What this rather messy calculation means is that we can calculate t from t_1 and t_2.

2.2.1 Graphs Without Parallel Edges

We now show how the algorithm works for graphs without parallel edges, although we allow up to one loop at each vertex.

Suppose that we are given G and a tree-decomposition $\big(\{X_i | i \in I\}, T = (I, F)\big)$ of width k such that T is a binary tree with root r and $|I| \leq 2|V|$. We need to associate each edge of G with a particular node of T. To do this we construct for each $i \in I$ a set of edges D_i such that the sets D_i are pairwise disjoint, $\bigcup_{i \in I} D_i = E$ and $\{u, v\} \in D_i \Rightarrow \{u, v\} \subseteq X_i$. We say that D_i is the set of edges *inside* X_i; in fact D_i will be some subset of the edges induced by X_i. There are many ways of constructing the D_i and any of them will do. For any graph of tree-width at most k, we can obtain one such construction in time $O(|E|+|V|)$ just by assigning edges greedily. For each $i \in I$ we define $E_i = \{e \in D_j | j = i$ or j is a descendant of $i\}$ and say that E_i is the set of edges inside Y_i.

For each $i \in I$ and each $\pi \in \Pi(X_i)$, we define $T_i(\pi)$ by

$$T_i(\pi) = \sum (x-1)^{-r(A)} (y-1)^{|A|-r(A)},$$

where the sum is over all sets A satisfying $A \subseteq E_i$ and $\Pi_{X_i}(A) = \pi$; that is all those sets which consist of edges inside Y_i and which partition X_i into connected components given by π.

The algorithm computes all the set of values $\{T_i(\pi) : \pi \in \Pi(X_i)\}$ for each $i \in I$ working upwards from the leaves of T to the root, r. For each i the values $\{T_i(\pi) : \pi \in \Pi(X_i)\}$ are calculated from the values for the children of i using exactly the type of calculation outlined above.

Suppose that we want to calculate T_i where i is a node with two children j and k (this is the harder case). At this stage we will know $T_j(\pi)$ for all $\pi \in \Pi(X_j)$ and $T_k(\pi)$ for all $\pi \in \Pi(X_k)$. We now calculate for each

$\pi \in \Pi(X_i)$, $lift_j(\pi)$ and $lift_k(\pi)$ where these are given by

$$lift_j(\pi) = \sum_{\substack{A \subseteq E_j \\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$$
$$lift_k(\pi) = \sum_{\substack{A \subseteq E_k \\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}.$$

This is easy because $lift_j(\pi)$ will be zero if the vertices of $X_i \setminus X_j$ are not present as singleton blocks in π (there are no edges between vertices of $X_i \setminus X_j$ contained in E_j) and otherwise

$$lift_j(\pi) = \sum_{\pi_j} T_j(\pi_j),$$

where the summation is over all partitions of X_j satisfying $\pi_j|(X_i \cap X_j) = \pi|(X_i \cap X_j)$.

We now calculate for each $\pi \in \Pi(X_i)$

$$mix_i(\pi) = \sum_{\substack{A \subseteq E_j \cup E_k \\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}.$$

This is done using the procedure outlined above and Equation 2.2.1 using the functions $lift_j$ and $lift_k$ in the roles of t_1 and t_2 and X_i as the intersecting set so

$$mix_i(\pi) = \sum_{(\pi_j,\pi_k)} \Big[lift_j(\pi_j) lift_k(\pi_k) \cdot \big((x-1)(y-1)\big)^{(|X_i|+\#\pi_i-\#\pi_j-\#\pi_k)} \Big] \tag{2.2.2}$$

where the summation is over all pairs (π_j, π_k) such that $\pi_j, \pi_k \in \Pi(X_i)$ and $\pi_j \vee \pi_k = \pi$. To compute $T_i(\pi)$ we now just need to take account of the

contribution from edges in D_i, that is the edges inside X_i so we set

$$con_i(\pi) = \sum_{\substack{A \subseteq D_i \\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}.$$

Finally we compute $T_i(\pi)$ for each partition π of X_i using the same procedure as before but with X_i as our intersecting set and con_i and mix_i in place of t_1 and t_2 so

$$T_i(\pi) = \sum_{(\pi',\pi'')} con_i(\pi') mix_i(\pi'') \big((x-1)(y-1)\big)^{(|X|+\#\pi-\#\pi'-\#\pi'')}, \tag{2.2.3}$$

where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi$.

The algorithm that carries out these calculations is shown as Procedures 2–5.

We prove that the algorithm is correct by showing that for each i it calculates T_i correctly; we do this by induction on the height of i in the tree, T. If i is a leaf then it is clear that the procedure $LEAF$ sets $T_i(\pi)$ to the correct value. Otherwise Equations 2.2.2–2.2.3 in the discussion preceding the statement of the algorithm show that the procedure $TWO-CHILDREN$ computes $T_i(\pi)$ correctly given T_j and T_k where j and k are the two children of i. Given that $TWO-CHILDREN$ is correct it is easy to see that $ONE-CHILD$ is correct because it is just the same as $TWO-CHILDREN$ but omitting the calculation of mix. Once we know T_r the algorithm correctly computes the answer by setting $T(G;x,y) = \sum_{\pi\in\Pi(X_r)}(x-1)^{r(E)}T_r(\pi)$.

2.2.2 Parallel Edges

The algorithm given above will return the correct answer for graphs with parallel edges but may not be efficient. With the definition as above it is

Procedure 2 Evaluating the Tutte Polynomial

Input: G where G has tree-width $\leq k$, rational numbers x and y with $x \neq 1$, a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G with width k and such that T is a binary tree with specified root r, and also the partition $\{D_i | i \in I\}$

$T^* \leftarrow T$

while $T^* \neq \emptyset$ **do**

$i \leftarrow$ a leaf of T^*

if i is a leaf of T **then**

CALL $LEAF(i)$

else if i has one child in T **then**

CALL $ONE-CHILD(i)$

else

CALL $TWO-CHILDREN(i)$

end if

Delete i from T^*

end while

$T(G; x, y) \leftarrow (x-1)^{r(E)} \sum_{\pi \in \Pi(X_r)} T_r(\pi)$

Output: $T(G; x, y)$

Procedure 3 Leaf

PROC $LEAF(i)$

while $\pi_i \in \Pi(X_i)$ **do**

$T_i(\pi_i) \leftarrow \sum_A (x-1)^{-r(A)} (y-1)^{|A|-r(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $\Pi_{X_i}(A) = \pi_i$

end while

Procedure 4 One-Child

PROC $ONE-CHILD(i)$

$j \leftarrow$ the child of i in T

while $\pi_i \in \Pi(X_i)$ **do**

if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_j)) + |X_i \setminus X_j|$ **then**

$lift_j(\pi_i) \leftarrow 0$

else

$lift_j(\pi_i) \leftarrow \sum_{\pi_j} T_j(\pi_j)$ where the summation is over all partitions π_j of X_j such that $\pi_j|(X_i \cap X_j) = \pi_i|(X_i \cap X_j)$

end if

end while

while $\pi_i \in \Pi(X_i)$ **do**

$con_i(\pi_i) \leftarrow \sum_A (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$ where the summation is over all sets of edges, A, that are subsets of D_i and satisfy $\Pi_{X_i}(A) = \pi_i$

end while

while $\pi_i \in \Pi(X_i)$ **do**

$T_i(\pi_i) \leftarrow \sum_{(\pi',\pi'')} con_i(\pi') lift_j(\pi'') \big((x-1)(y-1)\big)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$ where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi_i$

end while

Procedure 5 Two-Children

PROC $TWO-CHILDREN(i)$

j and $k \leftarrow$ the children of i in T

while $\pi_i \in \Pi(X_i)$ **do**

while $l \in \{j,k\}$ **do**

if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_l)) + |X_i \setminus X_l|$ **then**

$lift_l(\pi_i) \leftarrow 0$

else

$lift_l(\pi_i) \leftarrow \sum_{\pi_l} T_l(\pi_l)$ where the summation is over all partitions π_l of X_l such that $\pi_l|(X_i \cap X_l) = \pi_i|(X_i \cap X_l)$

end if

end while

end while

while $\pi_i \in \Pi(X_i)$ **do**

$mix_i(\pi_i) \leftarrow \sum_{(\pi_j,\pi_k)} lift_j(\pi_j) lift_k(\pi_k)\big((x-1)(y-1)\big)^{(|X_i|+\#\pi_i-\#\pi_j-\#\pi_k)}$ where the summation is over all pairs (π_j,π_k) such that $\pi_j,\pi_k \in \Pi(X_i)$ and $\pi_j \vee \pi_k = \pi_i$

end while

while $\pi_i \in \Pi(X_i)$ **do**

$con_i(\pi_i) \leftarrow \sum_A (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$ where the summation is over all sets of edges, A, that are subsets of D_i and satisfy $\Pi_{X_i}(A) = \pi_i$

end while

while $\pi_i \in \Pi(X_i)$ **do**

$T_i(\pi_i) \leftarrow \sum_{(\pi',\pi'')} con_i(\pi') mix_i(\pi'')\big((x-1)(y-1)\big)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$ where the summation is over all pairs (π',π'') such that $\pi',\pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi_i$

end while

possible that for some i, $|D_i|$ could be very large and not bounded by a function of k and so the number of operations needed to compute a sum over all subsets of D_i may no longer be polynomially bounded in the size of the input graph. We extend the construction of the sets D_i to graphs with parallel edges by stipulating that $\bigcup_{i\in I} D_i$ contains precisely one edge from each parallel class, that is a maximal set of edges all of which are in parallel with each other. The other conditions, namely that the sets D_i are pairwise disjoint and for all u and v $\{u,v\} \in D_i \Rightarrow \{u,v\} \subseteq X_i$, remain the same. We say a set A is *represented* in D_i if for each $e \in A$, either $e \in D_i$ or D_i contains an edge in parallel with e and denote this by $A \preceq D_i$. We let $m(e)$ be the size of the parallel class containing e.

The only problem comes when we compute *con* and $leaf$. Computing these two functions involves exactly the same calculation so we only show how to modify *con*. Now

$$con_i(\pi) = \sum_{\substack{A\preceq D_i \\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$$

but as we noted, this summation may be too large to compute efficiently. However

$$con_i(\pi) = \sum_{\substack{A\subseteq D_i \\ :\Pi_{X_i}(A)=\pi}} \sum_{(B_1,\ldots,B_{|A|})} (x-1)^{-r(A)}(y-1)^{|B_1|+\cdots+|B_{|A|}|-r(A)}$$

where if $A = \{e_1, \ldots e_l\}$ then the inner summation is over all l-tuples $(B_1, \ldots, B_l)$ such that for all j, B_j is non-empty and contained in the parallel class containing e_j. If $y \neq 1$ then

$$con_i(\pi) = \sum_{\substack{A\subseteq D_i \\ :\Pi_{X_i}(A)=\pi}} \big((x-1)(y-1)\big)^{-r(A)} \prod_{e\in A}(y^{m(e)}-1)$$

and if $y = 1$ then

$$con_i(\pi) = \sum_{\substack{A \subseteq D_i \\ :\Pi_{X_i}(A)=\pi, |A|=r(A)}} (x-1)^{-r(A)} \prod_{e \in A} m(e).$$

To avoid having to calculate y^j repeatedly for the same value of j we can compute the set $\{y, y^2, \ldots, y^M\}$, where $M = \max_{e \in E} m(e)$, before running the algorithm.

2.2.3 Complexity

Here we calculate an upper bound on the running time of our algorithm. We let $t(n, m, k, x, y, M)$ be the maximum number of operations needed to evaluate $T(G; x, y)$ when G is a graph with n vertices, tree-width at most k, m edges and such that the maximum size of any parallel class is M. We let $\alpha(n, m, k, x, y, M)$ be the maximum time needed for one multiplication or addition during the evaluation of $T(G, x, y)$. There are four stages of preprocessing:

1. Finding a tree-decomposition of width at most k which can be done in time $O(f(k) \cdot n)$ using the algorithm given in [Bod96].

2. Constructing a tree-decomposition where T is a binary tree which requires time $O(nk)$.

3. Computing the partition $\{D_i : i \in I\}$ which requires time $O(m + n)$.

4. Computing the numbers $\{y, y^2, \ldots, y^M\}$. The number of operations required for this stage is $O(M\alpha(n, m, k, x, y, M))$.

If π_1 is a partition of X_1 and π_2 is a partition of X_2 with $\max\{|X_1|, |X_2|\} = l$ then deciding whether $\pi_1 = \pi_2$ and computing $\pi_1 \vee \pi_2$ both require $O(l^2)$

operations. Computing $\Pi_X(A)$ where A is a set consisting of edges with both endpoints in X and at most one member from each parallel class takes time $O(|X|^3)$. Recall that $B(k)$ is used to denote the kth Bell Number, that is the number of distinct partitions of a set of size k.

Because $|I| \leq 2n$ the number of operations needed for the main part of the algorithm, that is omitting the preprocessing, is $O(nt'(n, m, N, k, x, y))$ where t' is the maximum time required for one call to the procedure $TWO-CHILDREN$.

The procedure $TWO-CHILDREN$ consists of 4 stages:

1. Calculation of $lift_j$ and $lift_k$ for which we need time $O\big((B(k+1))^2(k^2+\alpha)\big)$.
2. Evaluating mix takes time $O\big((B(k+1))^3(k^2+k\alpha)\big)$.
3. The computation of con needs $O\big(B(k+1)2^{(k+1)^2}(k^3+k^2\alpha)\big)$.
4. The final stage requires $O\big((B(k+1))^3(k^2+k\alpha)\big)$.

This gives a total time for a call to $TWO-CHILDREN$ of $O\big((B(k+1))^3 \cdot 2^{(k+1)^2} \cdot \alpha\big)$.

Finally we need to compute the maximum time for one arithmetical operation. To add, subtract, multiply or divide two l-bit integers takes $O(l \log l \log\log l)$ operations, [AHU74]. $lift$, con, mix, T_i are of the form $\sum_{A\in\mathcal{A}}(x-1)^{-r(A)}(y-1)^{(|A|-r(A))}$, where $\mathcal{A}$ is a subset of power-set of E and so the numbers involved in the calculations are either of this form or $\big((x-1)(y-1)\big)^j$ where $0 \leq j \leq 2k+2$, or y^j where $0 \leq j \leq M$. Now suppose $x = p_x/q_x$ and $y = p_y/q_y$ where p_x, q_x, p_y and q_y are integers such that p_x and q_x are coprime, and p_y and q_y are coprime. Note that $p_x \neq q_x$

since $x \neq 1$. Now

$$\begin{aligned}
&\sum_{A\in\mathcal{A}}(x-1)^{-r(A)}(y-1)^{(|A|-r(A))} \\
&\qquad = \sum_{A\in\mathcal{A}}\left(\frac{p_x-q_x}{q_x}\right)^{-r(A)}\left(\frac{p_y-q_y}{q_y}\right)^{(|A|-r(A))} \\
&\qquad = \sum_{A\in\mathcal{A}}\left(\frac{\left(\frac{(p_x-q_x)}{q_x}\right)^{(r(E)-r(A))}\left(\frac{(p_y-q_y)}{q_y}\right)^{(|A|-r(A))}}{\left(\frac{(p_x-q_x)}{q_x}\right)^{r(E)}}\right) \\
&\qquad = \sum_{A\in\mathcal{A}}\left(\frac{(p_x-q_x)^{r(E)-r(A)}q_x^{r(A)}(p_y-q_y)^{(|A|-r(A))}q_y^{(m-|A|+r(A))}}{(p_x-q_x)^{r(E)}q_y^m}\right).
\end{aligned}$$

Considering the denominator, we have $(p_x-q_x)^{r(E)}q_y^m \leq (|p_x|+|q_x|)^n|q_y|^m$, and for the numerator, we have

$$\begin{aligned}
&\sum_{A\in\mathcal{A}}(p_x-q_x)^{r(E)-r(A)}\cdot q_x^{r(A)}\cdot(p_y-q_y)^{(|A|-r(A))}\cdot q_y^{(m-|A|+r(A))} \\
&\qquad \leq \sum_{A\subseteq E}|p_x-q_x|^{r(E)-r(A)}\cdot|q_x|^{r(A)}\cdot|p_y-q_y|^{(|A|-r(A))}\cdot|q_y|^m \\
&\qquad \leq T(G;|p_x-q_x|+1,|p_y-q_y|+1)\cdot|q_x|^{r(E)}\cdot|q_y|^m \\
&\qquad \leq (|p_x|+|q_x|+|p_y|+|q_y|+2)^m\cdot|q_x|^n\cdot|q_y|^m.
\end{aligned}$$

The second last inequality follows from the definition of T and the final one is obtained using Equations 1.4.1–1.4.3 and induction on the number of edges of the graph. This means that an upper bound on the modulus of any of the numbers occurring in the denominator or numerator of numbers used in the calculations is

$$(|p_x|+|q_x|+|p_y|+|q_y|+2)^{(n+2m+2k+2)}.$$

The calculations within the main part of the algorithm all give quantities of the form $\sum_{A\in\mathcal{A}}(x-1)^{-r(A)}(y-1)^{|A|-r(A)}$, where $\mathcal{A}$ is a subset of the

power-set of E, and so we can avoid problems arising due to the numerator and denominator having a large common factor and thus becoming large themselves, because we can reduce the fraction with 2 integer divisions so that the denominator is $|p_x - q_x|^{r(E)}|q_y|^m$. We have shown that $\alpha(n, m, k, x, y, M) \leq s \log s \log\log s$ where $s = (n + 2m + 2k + 2)\log(|p_x| + |q_x|+|p_y|+|q_y|+2)$ and so the running time for the main part of the algorithm is $O\big(n(B(k+1))^3 \cdot 2^{(k+1)^2} \cdot (n+m+k) \cdot \log(|p_x|+|q_x|+|p_y|+|q_y|) \cdot \log s \cdot \log\log s\big)$ and the running time for the whole algorithm is

$$O\left(f(k) \cdot (n + M) \cdot (n + m) \cdot \log(n + m) \cdot \log\log(n + m) \cdot l \log l \cdot \log\log l\right)$$

where $l = \log(|p_x| + |q_x| + |p_y| + |q_y|)$.

Suppose the input graph has no parallel edges and G has tree-width k. By increasing the size of some of the X_i if necessary it is possible to show that there exists a tree-decomposition for G such that for each i, $|X_i| = k+1$. By adding extra vertices to T we can construct a tree-decomposition $\big(\{X_i | i \in I\}, T = (I, F)\big)$ for G such that T is rooted, for all $i \in I$, $|X_i| = k$ and for any edge $\{i, j\}$ of T, $|X_i \cap X_j| = k$. If our tree-decomposition satisfies both these conditions and G has n vertices then $I = n - k$. There are at most $k(k + 1)/2 + (k + 1)$ edges between vertices in the root of T and for any other node i of T there is precisely one vertex which does not appear in any ancestor of i in T and hence between the vertices of X_i there are at most $k + 1$ edges which have not been previously counted for an ancestor of X_i. Therefore the total number of edges is at most

$$(|I| - 1)(k + 1) + k(k + 1)/2 + (k + 1) = \frac{(k + 1)(2n - k)}{2}$$

so if the input graph has no parallel edges the total running time is at most

$$O\left(f(k) \cdot n^2 k \cdot \log(nk) \cdot \log\log(nk) \cdot l \log l \cdot \log\log l\right).$$

2.3 Case $x = 1$

The algorithm given above will not work when $x = 1$ because it will try to divide by 0. To avoid this problem, we use much the same notation as before but assume without loss of generality that our input graph is connected and for the moment has no parallel edges, although as before we allow up to one loop at each vertex. We define $T'_i(\pi)$ by

$$T'_i(\pi) = \sum_A (y-1)^{|A|-r(A)}$$

where the summation is over sets A of edges contained in E_i and such that each vertex of Y_i is connected to a vertex of X_i in the graph $G|A$. We note that an isolated vertex in X_i is connected to a vertex in X_i since it is connected to itself. The modified algorithm calculates $T'_i(\pi)$ for all π and for each i working upwards from the leaves to the root.

The modified algorithm is shown as Procedures 6–9.

We consider the operation of the procedure $TWO-CHILDREN'$. Suppose we want to compute $T'_i(\pi)$ where i is a node with 2 children j and k, and we know T'_j and T'_k. We first compute for all $\pi \in \Pi(X_i)$, $lift'_j(\pi)$ and $lift'_k(\pi)$ where $lift'_j$ is given by,

$$lift'_j(\pi) = \sum_A (y-1)^{|A|-r(A)}$$

where the summation is over all those sets A contained in E_j such that $\Pi_X(A) = \pi$ and each vertex of Y_j is connected to a vertex of X_i in $G|A$. $lift'_k$ is defined analogously. If the vertices of $X_i \setminus X_j$ are not present as singleton blocks in π then $lift'_j(\pi)$ will be zero and otherwise

$$lift'_j(\pi) = \sum_{\pi_j} T'_j(\pi_j),$$

Procedure 6 Evaluating the Tutte polynomial when $x = 1$

Input: G where G is connected and has tree-width $\leq k$, a rational number y with $x \neq 1$, a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G with width k and such that T is a binary tree with specified root r, and also the partition $\{D_i | i \in I\}$

$T^* \leftarrow T$

while $T^* \neq \emptyset$ **do**

$i \leftarrow$ a leaf of T^*

if i is a leaf of T **then**

CALL $LEAF'(i)$

else if i has one child in T **then**

CALL $ONE - CHILD'(i)$

else

CALL $TWO - CHILDREN'(i)$

end if

Delete i from T^*

end while

$T(G; 1, y) \leftarrow T_r(\pi)$ where $\pi \in \Pi(X_r)$ and $\#\pi = 1$

Output: $T(G; 1, y)$

Procedure 7 Leaf′

PROC $LEAF'(i)$

while $\pi_i \in \Pi(X_i)$ **do**

$T_i'(\pi_i) \leftarrow \sum_A (y-1)^{|A|-r(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $\Pi_{X_i}(A) = \pi_i$

end while

Procedure 8 One-Child′

PROC $ONE-CHILD'(i)$

$j \leftarrow$ the child of i in T

while $\pi_i \in \Pi(X_i)$ **do**

if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_j)) + |X_i \setminus X_j|$ **then**

$lift'_j(\pi_i) \leftarrow 0$

else

$lift'_j(\pi_i) \leftarrow \sum_{\pi_j} T'_j(\pi_j)$ where the summation is over all partitions π_j of X_j such that $\pi_j|(X_i \cap X_j) = \pi_i|(X_i \cap X_j)$ and every block of π_j contains at least one vertex of X_i

end if

end while

while $\pi_i \in \Pi(X_i)$ **do**

$con'_i(\pi_i) \leftarrow \sum_A (y-1)^{|A|-r(A)}$ where the summation is over all sets of edges, A, that are subsets of D_i and satisfy $\Pi_{X_i}(A) = \pi_i$

end while

while $\pi_i \in \Pi(X_i)$ **do**

$T'_i(\pi_i) \leftarrow \sum_{(\pi',\pi'')} con'_i(\pi') lift'_j(\pi'')(y-1)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$ where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi_i$

end while

Procedure 9 Two-Children′

PROC $TWO-CHILDREN'(i)$

j and $k \leftarrow$ the children of i in T

while $\pi_i \in \Pi(X_i)$ **do**

while $l \in \{j,k\}$ **do**

if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_l)) + |X_i \setminus X_l|$ **then**

$lift'_l(\pi_i) \leftarrow 0$

else

$lift'_l(\pi_i) \leftarrow \sum_{\pi_l} T'_l(\pi_l)$ where the summation is over all partitions π_l of X_l such that $\pi_l|(X_i \cap X_l) = \pi_i|(X_i \cap X_l)$ and each block of π_l contains at least one vertex of X_i

end if

end while

end while

while $\pi_i \in \Pi(X_i)$ **do**

$mix'_i(\pi_i) \leftarrow \sum_{(\pi_j,\pi_k)} lift'_j(\pi_j) lift'_k(\pi_k)(y-1)^{(|X_i|+\#\pi_i-\#\pi_j-\#\pi_k)}$ where the summation is over all pairs (π_j,π_k) such that $\pi_j,\pi_k \in \Pi(X_i)$ and $\pi_j \vee \pi_k = \pi_i$

end while

while $\pi_i \in \Pi(X_i)$ **do**

$con'_i(\pi_i) \leftarrow \sum_A (y-1)^{|A|-r(A)}$ where the summation is over all sets of edges, A, that are subsets of D_i and satisfy $\Pi_{X_i}(A) = \pi_i$

end while

while $\pi_i \in \Pi(X_i)$ **do**

$T'_i(\pi_i) \leftarrow \sum_{(\pi',\pi'')} con'_i(\pi') mix'_i(\pi'')(y-1)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$ where the summation is over all pairs (π',π'') such that $\pi',\pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi_i$

end while

where the summation is over those partitions of X_j satisfying $\pi_j|(X_i \cap X_j) = \pi|(X_i \cap X_j)$ and such that each block of π_j contains a vertex of X_i. This last restriction is needed to ensure that $lift'_j$ is a sum over subsets, A, of E_j such that every vertex of Y_j is connected in $G|A$ to a vertex of X_i rather than just to a vertex of X_j. The procedure $TWO-CHILDREN'$ next calculates mix'_i which is given by

$$mix'_i(\pi) = \sum_A (y-1)^{|A|-r(A)}$$

where the summation is over all subsets, A, of $E_j \cup E_k$ such that $\Pi_{X_i}(A) = \pi$ and every vertex of $Y_j \cup Y_k$ is connected to a vertex of X_i. To find an expression for mix' in terms of $lift'$ we have to modify Equation 2.2.1.

Suppose we have a graph $G = (V_1 \cup V_2, E_1 \cup E_2)$ with $V_1 \cap V_2 = X$, $E_1 \cap E_2 = \emptyset$ and such that any edge in E_i has both endpoints in V_i. For each $\pi \in \Pi(X)$ and for each $i \in \{1,2\}$, we define

$$t_i(\pi) = \sum_A (y-1)^{|A|-r(A)}$$

where the summation is over subsets A of E_i satisfying $\Pi_X(A) = \pi$ and such that every vertex of V_i is connected to a vertex of X. Now let $t(\pi)$ be given by

$$t(\pi) = \sum_A (y-1)^{|A|-r(A)}$$

where the summation is over all those subsets A of $E_1 \cup E_2$ which satisfy $\Pi_X(A) = \pi$ and such that every vertex of $V_1 \cup V_2$ is connected to a vertex of X. By modifying the argument preceding Equation 2.2.1 it is possible to show that

$$t(\pi) = \sum_{\substack{(\pi_1,\pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} t_1(\pi_1) t_2(\pi_2) (y-1)^{(|X| + \#\pi - \#\pi_1 - \#\pi_2)}.$$

This means that we can calculate mix'_i and T'_i from $lift'_j$ and $lift'_k$, using this equation, just as in the main algorithm we could calculate mix_i and T_i using Equation 2.2.1.

The procedure $LEAF'$ is the same as the procedure $LEAF$ and the procedure $ONE-CHILD'$ is the same as $TWO-CHILDREN'$ omitting the calculation of mix' just as in the main algorithm. The final stage of the algorithm sets $T(G;1,y) = T_r(\pi)$ where π is the partition of X_r containing all the vertices of X_r in one block. This is correct since $T(G;1,y) = \sum_A (y-1)^{|A|-r(A)}$ where the summation is over all those subsets A of E such that $G|A$ is connected, and we assumed initially that G was connected.

Exactly the same modification as that for the main algorithm must be made to cope with non-simple graphs. The running time for this algorithm satisfies the bound given for the main algorithm.

2.4 Computing T

We can write the Tutte polynomial in the form

$$T(G;x,y) = \sum_{i,j} t_{ij} x^i y^j.$$

A natural problem to consider is that where we input a graph of tree-width at most k and output the list of coefficients t_{ij}. In contrast with the problem of evaluating T, this problem has complexity $\Omega(n^3)$.

It is easy to construct a family of graphs $\{G_r\}$ with tree-width k such that there are $\Omega(n^2)$ coefficients exceeding $2^{\Omega(n)}$ and hence $\Omega(n^3)$ time is required to list the coefficients. One way to do this is to take a complete graph on k vertices $v_1, \ldots, v_k$ and add vertices $v_{k+1}, \ldots, v_{k+r}$ so that each one is connected by a single edge to each of $v_1, \ldots, v_k$ and now add vertices

$v_{k+r+1}, \ldots, v_{k+2r}$ so that for each s with $k + r + 1 \leq s \leq k + 2r$, v_s is connected to $v_2, \ldots, v_k$ and v_{s-r} by a single edge. T can be calculated using Equations 1.4.1–1.4.3.

We can examine the size of the coefficients of $T(G)$ by ordering the edges and considering the binary tree where each node is labelled with a graph obtained when 1.4.1–1.4.3 are applied recursively. The root is labelled with G and if a node is labelled with H then, providing H has at least one edge, the children of H correspond to the graphs obtained from H by deleting and contracting the lexicographically first edge, e, remaining, providing e is not a loop or an isthmus. If e is a loop (isthmus) then H has one child corresponding to deleting (contracting) e. The leaves correspond to graphs with $k(G)$ vertices and no edges and each leaf also corresponds to a term in the expansion of $T(G)$. If in obtaining a leaf L from G we contract i isthmuses and delete j loops then L corresponds to a term $x^i y^j$.

Now consider G_r. We order the edges so that if $i < j$ those edges adjacent to v_{i+k} and v_{i+k+r} come before those adjacent to v_{j+k} and v_{j+k+r} and the edges of the K_k come last. The edges adjacent to v_{i+k} and v_{i+k+r} are ordered so that

$$\{v_{i+k}, v_{i+k+r}\} \prec \{v_k, v_{i+k+r}\} \prec \ldots \prec \{v_2, v_{i+k+r}\}$$
$$\prec \{v_k, v_{i+k}\} \prec \ldots \prec \{v_1, v_{i+k}\}$$

where $e \prec f$ means that edge e precedes f. It is easy to see there are ways of deleting or contracting the edges adjacent to v_{k+1} and v_{k+r+1} in order so that the rest of the graph is unchanged and we have either deleted one loop, contracted one isthmus or neither. In other words there are nodes at depth $2k$ in the binary tree defined above which correspond to $G : (V(G) \setminus \{v_{r+k}, v_{2r+k}\})$ and are reached by deleting one loop or contracting one isthmus

or neither. Similarly for any i we can delete and contract the edges adjacent to v_{k+i} and v_{k+r+i} so that we delete one loop, contract one isthmus or neither and leave the K_k intact. We can delete and contract the K_k so that we contract one isthmus and delete no loops. This means that the coefficient of $x^{i+1}y^j$ is at least $r!/(i!j!(r-i-j)!)$ and so if $\lfloor r/4 \rfloor \leq i, j \leq \lfloor r/2 \rfloor$, this coefficient is at least $4^{\lfloor r/4 \rfloor}$.

This means that the running time of an algorithm to list all the coefficients of T must be $\Omega(n^3)$ because it takes this long to write them out and hence even when we restrict our input graphs to have tree-width at most k, more time is needed to list the coefficients of T than to evaluate it at a point. One approach to this problem is to evaluate T at several points using our algorithm and then use Lagrangian interpolation to find the coefficients.

2.5 Conclusion

As we mentioned in the introduction, evaluations of the Tutte Polynomial correspond to a wide variety of counting problems. In the case where the input graph has tree-width at most k, algorithms for some of these problems already exist, see for example [AP89]. We have shown that for any of these problems, if we restrict the input to graphs of tree-width at most k, for any fixed k, then there is a polynomial time algorithm. The methods we use can be extended to the Tutte Polynomial on signed graphs.

Chapter 3

Evaluating the Rank Generating Function of a Graphic 2-Polymatroid

3.1 Introduction

We now consider two natural problems concerning a two variable graph polynomial $S(G; u, v)$ that is closely related to the Tutte polynomial and was introduced in [OW93]. Like the Tutte polynomial, S contains a large variety of well studied specialisations, for instance the number of perfect matchings of a graph. We show that the complexity of S is very similar to that of the Tutte polynomial in that it is $\#P$-hard to evaluate at almost every point, a result well known for the Tutte polynomial [JVW90]. We go on to apply the techniques of the previous chapter to show that we can evaluate S if the input graph has tree-width at most a constant k. The polynomial S is essentially the rank generating function of a particular class of 2-polymatroids

which we now consider. The previous chapter gives some information on the theory of graphs of bounded tree-width and we follow the notation that we established there.

3.2 Integer polymatroids

An *integer polymatroid* (E, f) consists of a finite edge set, E, and an integer valued rank function, f, defined on all subsets of E and satisfying:

1. $f(\emptyset) = 0$,
2. if $X \subseteq Y$ then $f(X) \leq f(Y)$,
3. if $X, Y \subseteq E$ then $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$.

A k-polymatroid is a polymatroid (E, f) such that for all $e \in E$ $f(e) \leq k$. Polymatroids are a natural generalisation of the well-studied class of matroids, which correspond to 1-polymatroids, see for instance [Wel76]. For the rest of this chapter we will only be concerned with 2-polymatroids.

We need to consider two operations on a 2-polymatroid (E, f) which are defined in [OW93]. The *deletion* of a set A of edges, denoted by $(E, f) \setminus A$, is the 2-polymatroid $(E - A, f'_A)$ where for any $X \subseteq E \setminus A$,

$$f'_A(X) = f(X).$$

The *contraction* of a set A of edges, denoted by $(E, f)/A$ is the 2-polymatroid $(E - A, f''_A)$ where for any $X \subseteq E \setminus A$,

$$f''_A(X) = f(X \cup A) - f(A). \tag{3.2.1}$$

It is straightforward to check that, with these definitions, the two operations do actually produce 2-polymatroids. We will often just write $f \setminus A$ and f/A instead of $(E, f) \setminus A$ and $(E, f)/A$ respectively.

Any graph gives rise to a 2-polymatroid (E, f_G) by taking $E = E(G)$ and for any $A \subseteq E$ setting $f_G(A) = |V(G : A)|$. It is easy to check that this satisfies the definition of a 2-polymatroid. Moreover it is noted in [OW93] that such a 2-polymatroid (E, f_G) uniquely determines G up to the addition of isolated vertices. We say (E, f) is *induced* by G if it is isomorphic to (E, f_G). This polymatroid derived from graphs is the one that will interest us in the rest of this chapter.

The 2-polymatroid rank generating function was introduced in [OW93] and is defined by

$$S(f; u, v) = \sum_{A \subseteq E} u^{f(E)-f(A)} v^{2|A|-f(A)}.$$

When we consider the rank generating function of the 2-polymatroid derived from a graph G we will usually write $S(G; u, v)$. It is easy to see that adding isolated vertices to G will not affect S, consequently we will assume that all our graphs have no isolated vertices.

The following specialisations of S are stated in [OW93].

- $S(G; 1, 0)$ is the number of matchings of G.
- If G has no isolated vertices then $S(G; 0, 0)$ is the number of perfect matchings of G and $S(G; 0, 1)$ is the number of subsets of E spanning every vertex of G.
- If $u \neq 0$ then $u^{f_G(E)/2} S(G; u^{-1/2}, 0)$ is the polynomial $\sum_{k \geq 0} m_k u^k$ where m_k is the number of matchings of size k in G.
- $S(f; -u, -v) = (-1)^{f(E)} S(f; u, v)$.
- $S(f; \frac{1}{u}, u) = (1 + u^2)^{|E|} u^{-f(E)}$ for $u \neq 0$.

- For a graph G with no isolated vertices

$$(1-p)^{(|E|-f_G(E)/2)}\, p^{(f_G(E)/2)}\, S(G;0,p^{1/2}(1-p)^{-1/2})$$

  is the probability that G_p has no isolated vertices; where G_p is the random graph formed by deleting all the edges of G independently with probability $1-p$.

- $u^{f(E)}S(G;1/u,1) = \sum_{k\geq 0} r_k u^k$ where r_k is the number of subsets of E spanning k vertices. This polynomial can be thought of as a one-variable rank generating function.

This chapter examines the complexity of the following problems:

Problem 3.2.2 $\pi_1(u,v)$: RANK GENERATING FUNCTION EVALUATION AT (u,v)
Input A graph, G.
Output The evaluation at (u,v) of the rank generating function of the 2-polymatroid, induced by G.

Problem 3.2.3 $\pi_2(k)$: RANK GENERATING FUNCTION(k)
Input A graph, G, with tree-width at most k and rational numbers u and v.
Output The rank generating function of the 2-polymatroid, induced by G, evaluated at (u,v).

We show that $\pi_1(u,v)$ is $\#P$-hard for almost all rational values of (u,v). In contrast to this we show that for any fixed value of k there is a polynomial time algorithm which will compute RANK GENERATING FUNCTION(k). This behaviour is very similar to the complexity of the Tutte polynomial which is $\#P$-hard to evaluate at almost all points in the plane, [JVW90], a result which also remains true if we restrict the input to bipartite planar graphs

[VW92]. Conversely if we fix k we show in the previous chapter that there is a polynomial time algorithm which will evaluate at any rational point, the Tutte polynomial of any graph with tree-width at most k.

3.3 The Unrestricted Case

The main result in this section is

Theorem 3.3.1. *The problem $\pi_1(u, v)$ is $\#P$-hard to compute unless $uv = 1$ when we have a polynomial time algorithm to evaluate it.*

The rest of this section is devoted to the proof of this result. The case when $uv = 1$ is easy because if $u \neq 0$ then

$$S\left(f; \frac{1}{u}, u\right) = (1 + u^2)^{|E|} u^{-f(E)}$$

and obviously this can be evaluated very quickly.

All the hardness proofs rely on the following which is a restatement of a result of Valiant [Val79].

Theorem 3.3.2. *Computing $\pi_1(0, 0)$ is $\#P$-hard.*

Proof. If G has no isolated vertices then $S(G; 0, 0)$ is the number of perfect matchings of G which is $\#P$-hard to compute. □

It is convenient to consider a slightly larger class of polymatroids than just the ones which are induced by graphs because later we will need to consider a class of polymatroids that contains those induced by graphs and which is also closed under both deletion and contraction. We consider the effect of deletion and contraction later but for the moment note that contracting an edge in a polymatroid may create an edge with rank zero. Clearly these do not occur

in graphs because they would correspond to edges with no endpoints. We call such an edge a *circle* and say that a polymatroid is graphic if it is of the form $M = (E_1 \cup E_2, f)$ where $M \setminus E_2$ is graphic and for any $e \in E_2$, $f(e) = 0$, in other words M is induced by a graph except for the addition of some circles. In some places we go a little further and abuse our notation by allowing graphs, rather than just graphic polymatroids, to have circles.

A set, X, of edges is a *separator* for a 2-polymatroid (E, f) if $f(X) + f(E \setminus X) = f(E)$.

Single element separators can have rank zero, one or two, and for a graphic polymatroid these correspond to circles, a loop on a vertex that is incident with no other edges and an edge joining two vertices that are incident with no other edges. The 2-polymatroids $U_{0,1}$, $U_{1,1}$ and $U_{2,1}$ are the graphic polymatroids with precisely one edge e which is respectively a circle, loop or edge between two vertices.

If e is not a separator of f then [OW93] note that one of the following must occur.

1. $f(E \setminus e) = f(E)$ and $f(e) = 1$.
2. $f(E \setminus e) = f(E) - 1$ and $f(e) = 2$.
3. $f(E \setminus e) = f(E)$ and $f(e) = 2$.

For graphic 2-polymatroids the first case corresponds to e being a loop but not a separator, the second to e being a non-loop edge with precisely one of its endpoints having degree one, that is a pendant edge, and the third to any non-loop edge for which both endpoints have degree at least two.

It is worthwhile noting the effect of the operations of deletion and contraction on a graphic 2-polymatroid. Deletion is easy

$$(E, f_G) \setminus e = (E \setminus e, f_{G \setminus e}),$$

where $f_{G\setminus e}$ denotes the restriction of the rank function to $E \setminus e$ and so it just corresponds to normal deletion in the graph. Equation 3.2.1 shows that contracting a separator is equivalent to deleting it but generally contraction is more difficult. Suppose we contract the non-separating edge uv where we allow $v = u$. Then

$$(E, f_G)/uv = (E \setminus e, f_{G\sim e})$$

where $G \sim e$ is formed from G by deleting uv, replacing any loop attached at u or v or edge parallel to uv by a circle and replacing any edge uw (vw) for $w \neq u$ (v) by a loop at w and $f_{G\sim e}$ is the graphic 2-polymatroid induced by $G \sim e$.

The proof of the hardness result for the Tutte polynomial makes use of the *tensor product* construction of Brylawski [Bry82]. The tensor product of a matroid M with a pointed matroid N, that is a matroid with a distinguished element e, is formed by taking the 2-sum of M and N about each point of M. In [JVW90] the matroid $U_{1,k+1}$ which is the graphic matroid induced by the graph consisting of k parallel edges was used in the role of N in order to prove the complexity results. This particular tensor product is known as a k-thickening because each edge of the graph is replaced by k parallel edges. We have not constructed a general 2-sum for a polymatroid but we define the *k-thickening* of a graphic 2-polymatroid induced by G with l edges of rank zero to be the 2-polymatroid induced by G^k together with kl edges of rank zero where G^k is formed by replacing each edge in G, including loops, by k parallel edges.

ϕ is said to be a *generalised Tutte invariant (for graphic 2-polymatroids)*

if there exist constants a, b, c, d, m, n, x, y and $z \in \mathbb{C}$ such that

$$\phi(U_{2,1}) = x,$$
$$\phi(U_{0,1}) = y,$$
$$\phi(U_{1,1}) = z$$

and for any graphic 2-polymatroid (E, f_G)

$$\phi(f) = \phi(f \setminus (E - e))\phi(f \setminus e) \qquad \text{if } e \text{ is a separator of } f;$$

and if e is not a separator,

$$\phi(f) = \begin{cases} a\phi(f \setminus e) + b\phi(f/e) & \text{if } f(E \setminus e) = f(E) \text{ and } f(e) = 1, \\ c\phi(f \setminus e) + d\phi(f/e) & \text{if } f(E \setminus e) = f(E) - 1 \text{ and } f(e) = 2, \\ m\phi(f \setminus e) + n\phi(f/e) & \text{if } f(E \setminus e) = f(E) \text{ and } f(e) = 2. \end{cases}$$

The following theorem is from [OW93]:

Theorem 3.3.3. *Let ϕ be a generalised Tutte invariant on graphic 2-polymatroids and suppose that at most two of x, y, z, a, b, c, m and n are zero. Then one of the following occurs:*

1. *$a = m$; $d = n$; $mx = mn + c^2$; $ny = mn + b^2$; $z = b + c$; $m \neq 0$; $n \neq 0$; and for all 2-polymatroids f,*

$$\phi(f) = m^{|E| - f(E)/2} n^{f(E)/2} S\left(f; \frac{c}{(mn)^{1/2}}, \frac{b}{(mn)^{1/2}}\right);$$

2. *$z^2 = xy = ax = cz + dy = mx + ny$; $yz = az + by$; $xz = cx + dz$; and, for all 2-polymatroids f, $\phi(f) = Q(f)$ where*

$$Q(f) = \begin{cases} y^{(|E| - f(E))} z^{f(E)} & \text{if } f(E) \leq |E|; \\ x^{(f(E) - |E|)} z^{(2|E| - f(E))} & \text{otherwise.} \end{cases}$$

It is easy to show that S is a generalised Tutte invariant on 2-polymatroids with

$$x = 1 + u^2, \quad y = 1 + v^2, \quad z = u + v, \quad m = n = 1, \tag{3.3.4}$$
$$a = d = 1, \quad b = v \text{ and } c = u.$$

The family of hyperbolae H_α defined by

$$H_\alpha = \{(u, v) : uv = \alpha\}$$

seems to play an important role in the theory. If $\alpha \neq 0$ then along the hyperbola H_α, we can write

$$S(G; u, v) = S_\alpha(G; v) = \sum_{i=0}^{|E|} c_i v^{2i - f(E)}$$

for certain coefficients c_i. It is convenient to consider the H_0 in two separate parts corresponding to the u and v axes which we denote respectively by H_0^u and H_0^v but in either case it is obvious that the restriction of S to either part is a one variable polynomial. All this motivates the following problem which has a crucial part in the proof of hardness of π_1:

Problem 3.3.5 $\pi_3(\alpha)$: H_α RANK GENERATING FUNCTION
Input A graph G.
Output The polynomial $S_\alpha(G; v)$.

It is clear that for any u and v,

$$\pi_1(u, v) \propto \pi_3(uv).$$

We use $\pi_3(0^u)$ and $\pi_3(0^v)$ in the obvious way, to denote the problem of computing the restriction of S to H_0^u and H_0^v respectively.

Our first result relates the rank generating function of a graphic polymatroid with that of its k-thickening.

Proposition 3.3.6. *If $v \neq 0$ then*

$$S\left(G^k; u, v\right) = \left(\frac{(1+v^2)^k - 1}{v^2}\right)^{f(E)/2} \cdot S\left(G; \frac{uv}{\sqrt{(1+v^2)^k - 1}}, \sqrt{(1+v^2)^k - 1}\right).$$

If $v = 0$ then

$$S\left(G^k; u, 0\right) = k^{f(E)/2} S\left(G; \frac{u}{\sqrt{k}}, 0\right).$$

To shorten the proof of this Proposition we first prove the following lemma. We let R_k be the graph consisting of just k circles, L_k be the graph with just one vertex and k loops and M_k be the graph with 2 vertices and k edges between them.

Lemma 3.3.7. *If $k \geq 2$*

$$S(R_k; u, v) = (1+v^2)^k.$$
$$S(L_k; u, v) = v(1+v^2)^{k-1} + \ldots + v(1+v^2) + u + v.$$
$$S(M_k; u, v) = (1+v^2)^{k-1} + \ldots + (1+v^2) + 1 + u^2.$$

Proof. The first equation is simple to check because each circle is a separator and $S(R_1; u, v) = 1 + v^2$. We prove the second by induction. If $k = 2$ then $S(L_k; u, v) = v(1+v^2) + u + v$. Otherwise using induction

$$\begin{aligned} S(L_k; u, v) &= S(L_{k-1}; u, v) + vS(R_{k-1}; u, v) \\ &= v(1+v^2)^{k-2} + \ldots + v(1+v^2) + u + v + v(1+v^2)^{k-1}. \end{aligned}$$

The third is also proved using induction. If $k = 1$ then $S(M_k; u, v) = 1 + u^2$. Otherwise using induction

$$\begin{aligned} S(M_k; u, v) &= S(M_{k-1}; u, v) + S(R_{k-1}; u, v) \\ &= (1+v^2)^{k-2} + \ldots + (1+v^2) + 1 + u^2 + (1+v^2)^{k-1}. \end{aligned}$$

□

Proof of Proposition: We let $\phi(G;u,v) = S\left(G^k;u,v\right)$. To prove this result it is just necessary to show that ϕ is a generalised Tutte invariant on 2-polymatroids and then use Theorem 3.3.3. Let e be an edge of G. There are three cases to consider where e is a separator and three cases where e is not. In the following we make repeated use of Equation 3.3.4.

In each of the cases when e is a separator, e will be replaced by k edges in G^k which together form a separator of G^k. Consequently

$$\phi(G;u,v) = \phi(G \setminus e;u,v)\phi(G|e;u,v).$$

So when e is a circle

$$\phi(G;u,v) = (1+v^2)^k\phi(G \setminus e;u,v),$$

when e is an isolated loop

$$\phi(G;u,v) = \left(v(1+v^2)^{k-1} + \ldots v(1+v^2) + u + v\right)\phi(G \setminus e;u,v),$$

and when e is an isolated (non-loop) edge

$$\phi(G;u,v) = \left((1+v^2)^{k-1} + \ldots (1+v^2) + 1 + u^2\right)\phi(G \setminus e;u,v).$$

We now consider the cases where e is not a separator in G. All three cases are quite similar. We use Equation 3.3.4, and contract and delete one of the edges replacing e in G^k, to leave respectively $(G \sim e)^k$ with $k-1$ circles and the k-thickening of G but with only $k-1$ edges replacing e. We repeat the procedure with this second graph. First suppose that e is a non-isolated loop of G.

$$\begin{aligned}\phi(G;u,v) &= vS(R_{k-1};u,v)S\left((G\sim e)^k;u,v\right)\\ &\quad + \ldots + vS(R_1;u,v)S\left((G\sim e)^k;u,v\right)\\ &\quad + vS\left((G\sim e)^k;u,v\right) + S\left((G\setminus e)^k;u,v\right)\\ &= (v(1+v^2)^{k-1} + \ldots + v(1+v^2) + v)S\left((G\sim e)^k;u,v\right)\\ &\quad + S\left((G\setminus e)^k;u,v\right).\end{aligned}$$

Secondly if e is a pendant edge of G then

$$\begin{aligned}\phi(G;u,v) &= S(R_{k-1};u,v)S\left((G\sim e)^k;u,v\right)\\ &\quad + \ldots + S(R_1;u,v)S\left((G\sim e)^k;u,v\right)\\ &\quad + S\left((G\sim e)^k;u,v\right) + uS\left((G\setminus e)^k;u,v\right)\\ &= ((1+v^2)^{k-1} + \ldots + (1+v^2) + 1)S\left((G\sim e)^k;u,v\right)\\ &\quad + uS\left((G\setminus e)^k;u,v\right).\end{aligned}$$

Finally if e is an edge with both endpoints having degree at least two then

$$\begin{aligned}\phi(G;u,v) &= S(R_{k-1};u,v)S\left((G\sim e)^k;u,v\right)\\ &\quad + \ldots + S(R_1;u,v)S\left((G\sim e)^k;u,v\right)\\ &\quad + S\left((G\sim e)^k;u,v\right) + S\left((G\setminus e)^k;u,v\right)\\ &= ((1+v^2)^{k-1} + \ldots + (1+v^2) + 1)S\left((G\sim e)^k;u,v\right)\\ &\quad + S\left((G\setminus e)^k;u,v\right).\end{aligned}$$

If $uv \neq 1$ and u, v are not both zero then it is easy to check that the first case of Theorem 3.3.3 applies and so

$$\begin{aligned}\phi(G;u,v) &= S(G^k;u,v)\\ &= \left((1+v^2)^{k-1}+\cdots+(1+v^2)+1\right)^{f(E)/2}\\ &\quad\cdot S\left(G;\frac{u}{\sqrt{(1+v^2)^{k-1}+\ldots+(1+v^2)+1}},\right.\\ &\quad\left.\frac{v(1+v^2)^{k-1}+\ldots+v(1+v^2)+v}{\sqrt{(1+v^2)^{k-1}+\ldots+(1+v^2)+1}}\right).\end{aligned}$$

If $v=0$ this simplifies to

$$\phi(G;u,0) = k^{f(E)/2}S\left(G;\frac{u}{\sqrt{k}},0\right)$$

and otherwise we have

$$\begin{aligned}\phi(G;u,v) &= \left(\frac{(1+v^2)^k-1}{v^2}\right)^{f(E)/2}\\ &\quad\cdot S\left(G;\frac{uv}{\sqrt{(1+v^2)^k-1}},\sqrt{(1+v^2)^k-1}\right).\end{aligned}$$

□

We now give a result that is halfway to the main theorem of this section.

Theorem 3.3.8. *If $\alpha \neq 1$ then $\pi_3(\alpha) \propto \pi_1(u,v)$ for any rational u and v not both zero such that $uv=\alpha$.*

Proof. We begin by proving the result in the case when $\alpha \neq 0$. The idea is as follows. Assume we have an oracle to compute $S(G;u,v)$ for any graph G. Hence for a given graph G and for any positive integer k, we know $S(G^k;u,v)$, and so using Proposition 3.3.6 we can compute

$$S\left(G;\frac{uv}{\sqrt{(1+v^2)^k-1}},\sqrt{(1+v^2)^k-1}\right).$$

All these points lie on H_α so if we do this for enough values of k we can compute the polynomial $S_\alpha(G;v)$ using Lagrange interpolation. More precisely, suppose we input a graph G and that for some rationals u, v with $uv = \alpha$ we have an oracle to compute $S(H;u,v)$ for every graph H. We write $S_\alpha(G;v) = \sum_{i=0}^{|E|} c_i v^{2i-f(E)}$. For each k such that $1 \leq k \leq |E|+1$, we compute $S(G^k;u,v)$, and then

$$\begin{aligned}\sum_{i=0}^{|E|} c_i \left(\sqrt{(1+v^2)^k - 1}\right)^{2i-f(E)} &= S\left(G; \frac{uv}{\sqrt{(1+v^2)^k-1}}, \sqrt{(1+v^2)^k-1}\right) \\ &= \left(\frac{v}{\sqrt{(1+v^2)^k-1}}\right)^{f(E)} S(G^k;u,v).\end{aligned}$$

Rearranging all this gives

$$\sum_{i=0}^{|E|} c_i \left((1+v^2)^k - 1\right)^i = v^{f(E)} S(G^k;u,v).$$

The square roots have been eliminated so this is an expression containing only rationals. To compute c_i for $0 \leq i \leq |E|$ we solve the $|E|+1$ equations resulting from substituting $k = 1, 2 \ldots, |E|+1$, using Gaussian elimination. To see that this gives a polynomial time algorithm to compute each c_i we need to note two facts. Firstly the $|E|+1$ equations are linearly independent. This is because the determinant of the coefficients of the equations is a Vandermonde determinant which is well known to be non-zero. Secondly each of the coefficients is polynomially bounded in terms of the input size, that is the length of the description of G. Gaussian elimination on an $n \times n$ matrix requires $O(n^3)$ arithmetical operations and can be done in such a way that the length of the description of the entries of the matrix remains polynomially bounded in terms of the original length of the description of

the entries. See [GLS80] for a discussion of this. Thus we can recover the coefficients c_i in polynomial time.

Along H_0^v we have

$$S(G;u,v) = \sum_{A\subseteq E: f(A)=f(E)} v^{2|A|-f(E)} = \sum_{i=0}^{|E|} c_i v^{2i-f(E)}$$

and so we can use exactly the same procedure as above to show that $\pi_3(0^v) \propto \pi_1(0,v)$ for any non-zero rational v.

The final case is slightly different, just because the form of the tensor product is different when $v = 0$. Suppose we have an oracle to evaluate $S(G;u,0)$ for any non-zero rational u. We write $S(G;u,0) = \sum_{i=0}^{f(E)} c_i u^{f(E)-i}$. Since

$$S(G;u,0) = \sum_{A\subseteq E: f(A)=2|A|} u^{f(E)-f(A)}$$

we have that c_i is zero unless i is even.

Using Proposition 3.3.6, we have for $k \geq 1$

$$S\left(G; \frac{u}{\sqrt{k}}, 0\right) = \left(\frac{1}{\sqrt{k}}\right)^{f(E)} S(G^k;u,0)$$

and so

$$\sum_{i=0}^{f(E)} c_i k^{i/2} u^{-i} = \left(\frac{1}{u}\right)^{f(E)} S(G^k;u,0).$$

This is a rational expression because c_i is zero unless i is even. Hence we can solve for $c_0,\ldots,c_{f(E)}$ by computing the values $S(G^1;u,0),\ldots,S(G^{f(E)+1};u,0)$ and solving for $c_0,\ldots,c_{f(E)}$ which is possible because the determinant of the matrix of coefficients is non-zero since it has the form of a Vandermonde determinant. □

Theorem 3.3.9.

1. *The problems $\pi_3(0^u)$ and $\pi_3(0^v)$ are both $\#P$-hard.*

2. *For any rationals u, v such that $uv = 0$, we have that $\pi_1(u, v)$ is $\#P$-hard.*

Proof. 1. $\pi_1(0, 0) \propto \pi_3(0^u)$ and $\pi_1(0, 0) \propto \pi_3(0^v)$.

2. If $u \neq 0$ then $\pi_3(0^u) \propto \pi_1(u, v)$ and if $v \neq 0$ then $\pi_3(0^v) \propto \pi_1(u, v)$ and so using the first half of this theorem we have the required result. If u and v are both zero then we just have Theorem 3.3.2.

□

We now move on to consider another specialisation of S which plays an important role in the proof of hardness. Let $s(G; u)$ be the one-variable polynomial given by

$$s(G; u) = S(G; u, 1) = \sum_{A \subseteq E} u^{f(E)-f(A)}.$$

This polynomial seems to be a fairly natural object to consider as it is just a one-variable rank generating function. The second half of the proof of our main result is contained in the following theorem.

Theorem 3.3.10. *For any rational u, $\pi_1(0, 1) \propto \pi_1(u, 1)$.*

Proof. We can obviously assume that $u \neq 0$. We write

$$s(G; u) = \sum_{i=0}^{f(E)} r_i u^{f(E)-i}$$

where r_i is the number of subsets of $E(G)$ which are incident with i vertices. Let G_k be the graph formed from G by adding k loops at each non-isolated vertex. We have that

$$r_i(G_k) = \sum_{j=0}^{i} r_j \binom{f(E) - j}{i - j} 2^{jk}(2^k - 1)^{i-j}. \tag{3.3.11}$$

This follows because each set A of edges that is incident with j vertices in G can be extended to give a set incident with i vertices by adding any number of loops at $i-j$ vertices which were isolated in $G|A$ and possibly adding loops at the vertices of $G:A$.

The idea is to use an oracle for $\pi_1(u,1)$ to calculate $s(G;u)$, $s(G_1;u)$, ..., $s(G_{f(E)};u)$ and then solve for the r_i. Using Equation 3.3.11 gives for $k \geq 1$,

$$\begin{aligned}
s(G_k;u) &= \sum_{i=0}^{f(E)} \sum_{j=0}^{i} r_j \binom{f(E)-j}{i-j} 2^{jk} (2^k-1)^{i-j} u^{f(E)-i} \\
&= \sum_{j=0}^{f(E)} \sum_{i=j}^{f(E)} r_j \binom{f(E)-j}{i-j} 2^{jk} (2^k-1)^{i-j} u^{f(E)-i} \\
&= \sum_{j=0}^{f(E)} \sum_{i=0}^{f(E)-j} r_j \binom{f(E)-j}{i} 2^{jk} (2^k-1)^{i} u^{f(E)-i-j} \\
&= \sum_{j=0}^{f(E)} r_j 2^{jk} \left(u + (2^k-1)\right)^{f(E)-j}.
\end{aligned}$$

This means that we can solve for $r_0, \ldots, r_{f(E)}$ using the values $s(G;u)$, $s(G_1;u)$, ..., $s(G_{f(E)};u)$ because the determinant of the matrix of coefficients has the form of a Vandermonde determinant and is non-zero. The sizes of the entries of the matrix are polynomially bounded in terms of the size of the input graph G and so solving for $r_0, \ldots, r_{f(E)}$ can be done in polynomial time, as shown in [GLS80]. □

Proof of Main Theorem: The only remaining case is when u and v are rationals such that $uv \neq 0$ and $uv \neq 1$. The $\#P$-hardness of $\pi_1(u,v)$ follows because

$$\pi_1(0,0) \propto \pi_3(0^v) \propto \pi_1(0,1) \propto \pi_1(uv,1) \propto \pi_3(uv) \propto \pi_1(u,v).$$

and $S(G;0,0)$ is the number of perfect matchings of G. □

3.4 The Bounded Tree Width Case

The main result of this section is the following.

Theorem 3.4.1. *For any fixed k, there exists an algorithm $\mathcal{A}$, which will input any graph G, with tree-width at most k, and rational numbers $x = p_x/q_x$ and $y = p_y/q_y$ where both p_x and q_x, and p_y and q_y are coprime, and evaluate $S(G; u, v)$, using at most $O((n + M) \cdot m \cdot \log m \cdot \log\log m \cdot l \log l \cdot \log\log l)$ operations, where $n = |V|$, $m = |E|$, $l = \log(|p_x| + |q_x| + |p_y| + |q_y|)$ and M is the largest size of a parallel class of edges.*

The methods used in this section are very similar to those used in the previous chapter where it was shown that there is a polynomial time algorithm to evaluate the Tutte polynomial for graphs with bounded tree-width. We refer to the previous chapter for information on tree-width but some of the definitions and constructions are repeated here in order to make the chapter reasonably self-contained. From here until the end of Section 3.4.3 we will assume that we are evaluating S at a point (u, v) with $u \neq 0$. The algorithm we give here will not work if $u = 0$ because it requires division by u. We show later how to overcome this difficulty. The method of computation is very similar to that in the previous chapter but where we were previously interested in partitions of a set X, we are now interested in subsets of X.

We first give an illustration of the main idea involved in the computation. Let $G = (V_1 \cup V_2, E_1 \cup E_2)$ be a graph with $V_1 \cap V_2 = X$, $E_1 \cap E_2 = \emptyset$ and such that any edge in E_i has both endpoints in V_i. As in the previous chapter, we refer to a set X which occurs in this way as an *intersecting set*. Suppose that for any subset, Z, of X and any i and j we know the number of subsets A of E_1 (E_2) with rank i, cardinality j and satisfying $A \cap X = Z$. We denote this number by $N_1(Z, i, j)$ ($N_2(Z, i, j)$). With this information we can calculate

$N(Z,i,j)$, the number of subsets A of $E_1 \cup E_2$ with rank i, cardinality j and satisfying $A \cap X = Z$ because if $A_1 \subseteq E_1$, $A_2 \subseteq E_2$ then

$$f(A_1 \cup A_2) = f(A_1) + f(A_2) - |V(G : A_1) \cap X| - |V(G : A_2) \cap X| + |V(G : (A_1 \cup A_2)) \cap X|.$$

Hence

$$N(Z,i,j) = \sum_{\substack{(Z_1,Z_2) \\ : Z_1 \cup Z_2 = Z}} \sum_{i_1=0}^{i} \sum_{j_1=0}^{j} N(Z_1, i_1, j_1) N(Z_2, i - i_1 + I, j - j_1)$$

where $I = |V(G : A_1) \cap X| + |V(G : A_2) \cap X| - |V(G : (A_1 \cup A_2)) \cap X|$.

Now suppose that rather than knowing $N_1(Z,i,j)$ and $N_2(Z,i,j)$ explicitly, we know the evaluation at the point (u,v) where $u \neq 0$ of some polynomials similar to the rank generating function. More precisely we know

$$\begin{aligned} s_1(Z) &= \sum_{A \subseteq E_1 : V(G:A) \cap X = Z} u^{-r(A)} v^{2|A| - r(A)} \\ &= \sum_{i,j} N_1(Z,i,j) u^{-i} v^{2j-i} \end{aligned}$$

and

$$\begin{aligned} s_2(Z) &= \sum_{A \subseteq E_2 : V(G:A) \cap X = Z} u^{-r(A)} v^{2|A| - r(A)} \\ &= \sum_{i,j} N_2(Z,i,j) u^{-i} v^{2j-i} \end{aligned}$$

and we want to compute for each $Z \subseteq X$

$$s(Z) = \sum_{A \subseteq E_1 \cup E_2 : V(G:A) \cap X = Z} u^{-r(A)} v^{2|A| - r(A)}.$$

We let $I = I(Z_1, Z_2) = |Z_1| + |Z_2| - |Z_1 \cup Z_2|$ and then

$$\begin{aligned}
s(Z) &= \sum_{i,j} N(Z,i,j) u^{-i} v^{2j-i} \\
&= \sum_{i,j} \sum_{\substack{(Z_1,Z_2) \\ : Z_1 \cup Z_2 = Z}} \sum_{i_1,j_1} \Big[N_1(Z_1,i_1,j_1) N_2(Z_2, i+I-i_1, j-j_1) \cdot u^{-i} v^{2j-i} \Big] \\
&= \sum_{\substack{(Z_1,Z_2) \\ : Z_1 \cup Z_2 = Z}} \sum_{i,j,i_1,j_1} \Big[N_1(Z_1,i_1,j_1) u^{-i_1} v^{2j_1-i_1} \cdot N_2(Z_2, i+I-i_1, j-j_1) \\
&\quad \cdot u^{(-i-I+i_1)} v^{(2j-2j_1-i-I+i_1)} \cdot (uv)^I \Big] \\
&= \sum_{\substack{(Z_1,Z_2) \\ : Z_1 \cup Z_2 = Z}} \sum_{i_1,j_1,i_2,j_2} \Big[N_1(Z_1,i_1,j_1) u^{-i_1} v^{2j_1-i_1} \\
&\quad \cdot N_2(Z_2,i_2,j_2) u^{-i_2} v^{2j_2-i_2} \cdot (uv)^I \Big]
\end{aligned}$$

(3.4.2)

$$= \sum_{\substack{(Z_1,Z_2) \\ : Z_1 \cup Z_2 = Z}} s_1(Z_1) s_2(Z_2) (uv)^{(|Z_1|+|Z_2|-|Z|)}.$$

3.4.1 Graphs Without Parallel Edges

As in the previous chapter, we first state and explain the algorithm for graphs without parallel edges, although we allow up to one loop at each vertex.

Suppose that we are given G and a tree-decomposition $\big(\{X_i | i \in I\}, T = (I, F)\big)$ of width k. By the discussion in the previous chapter we may assume that T is a binary tree with root r and $|I| \leq 2n$. We can also construct, for each $i \in I$, a set of edges D_i such that the sets D_i are pairwise disjoint, $\bigcup_{i \in I} D_i = E$ and $\{u, v\} \in D_i \Rightarrow \{u, v\} \subseteq X_i$. As before, in the previous chapter, for each $i \in I$ we define $E_i = \{e \in D_j | j = i$ or j is a descendant of $i\}$.

For each $i \in I$ and each $Z \subseteq X_i$, we define $S_i(Z)$ by

$$S_i(Z) = \sum_A u^{-f(A)} v^{2|A|-f(A)},$$

where the sum is over all sets A satisfying $A \subseteq E_i$ and $V(G:A) \cap X_i = Z$.

The algorithm works in exactly the same way as the algorithm in the previous chapter by computing all the set of values $\{S_i(Z) : Z \subseteq X_i\}$ for each $i \in I$ working upwards from the leaves of T to the root, r. At each stage we compute the functions $lift$, mix, con which are defined in an analogous way to those in the previous chapter. The computations are similar to those in the previous chapter but we now use use Equation 3.4.2.

Suppose that we want to compute S_i where i is a node with two children j and k. At this point we will know $S_j(Z_j)$ for all $Z_j \subseteq X_j$, and $S_k(Z_k)$ for all $Z_k \subseteq X_k$. We first calculate for each $Z \subseteq X_i$, $lift_j(Z)$ and $lift_k(Z)$ which are given by

$$lift_j(Z) = \sum_{\substack{A \subseteq E_j \\ :V(G:A) \cap X_i = Z}} u^{-f(A)} v^{2|A|-f(A)}$$

$$lift_k(Z) = \sum_{\substack{A \subseteq E_k \\ :V(G:A) \cap X_i = Z}} u^{-f(A)} v^{2|A|-f(A)}.$$

These are just given by

$$lift_j(Z) = \sum_Z S_j(Z),$$

where the summation is over all subsets Z of X_j such that $X_i \cap X_j = Z$. $lift_k$ is defined similarly.

Next we calculate for every $Z \subseteq X_i$

$$mix_i(Z) = \sum_{\substack{A \subseteq E_j \cup E_k \\ :V(G:A) \cap X_i = Z}} u^{-f(A)} v^{2|A|-f(A)}.$$

To do this we use Equation 3.4.2 with the functions $lift_j$ and $lift_k$ in the roles of s_1 and s_2 and X_i as the intersecting set. We then compute for each $Z \subseteq X_i$

$$con_i(Z) = \sum_{\substack{A \subseteq D_i \\ :V(G:A) \cap X_i = Z}} u^{-f(A)} v^{2|A| - f(A)}$$

and finally compute $S_i(Z)$ for every $Z \subseteq X_i$ by using Equation 3.4.2 with con_i and mix_i in place of s_1 and s_2 and X_i as the intersecting set. The algorithm that carries out these calculations is shown as Procedures 10–13.

The proof that this algorithm is correct follows using exactly the same induction as in the previous chapter and the preceding discussion.

3.4.2 Parallel Edges

We have exactly the same problem with parallel edges as we had in the previous chapter in that if parallel edges are present the algorithm will return the correct answer but may not be efficient. It is possible that for some i, $|D_i|$ could be very large and not bounded by a function of k and as the algorithm attempts to sum over all subsets of D_i, the running time may no longer be polynomially bounded in the size of the input of the graph. As in the previous chapter we extend the construction of the sets D_i to graphs with parallel edges by stipulating that $\bigcup_{i \in I} D_i$ contains precisely one edge from each parallel class. The other conditions, namely that the sets D_i are pairwise disjoint and for all u and v $\{u, v\} \in D_i \Rightarrow \{u, v\} \subseteq X_i$, remain the same. We say a set A is *represented* in D_i if for each $e \in A$, either $e \in D_i$ or D_i contains an edge in parallel with e and denote this by $A \preceq D_i$. We let $m(e)$ be the size of the parallel class containing e.

Procedure 10 Evaluating the Rank Generating Function

Input: G where G has tree-width $\leq k$, rational numbers u and v with $u \neq 0$, a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G with width k and such that T is a binary tree with specified root r, and also the partition $\{D_i | i \in I\}$

$T^* \leftarrow T$

while $T^* \neq \emptyset$ **do**

$i \leftarrow$ a leaf of T^*

if i is a leaf of T **then**

CALL $LEAF(i)$

else if i has one child in T **then**

CALL $ONE-CHILD(i)$

else

CALL $TWO-CHILDREN(i)$

end if

Delete i from T^*

end while

$S(G; u, v) \leftarrow u^{f(E)} \sum_{Z \subseteq X_r} S_r(Z)$

Output: $S(G; u, v)$

Procedure 11 Leaf

PROC $LEAF(i)$

while $Z_i \subseteq X_i$ **do**

$S_i(Z_i) \leftarrow \sum_A u^{-f(A)} v^{2|A|-f(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $V(G : A) = Z_i$

end while

Procedure 12 One-Child

PROC $ONE-CHILD(i)$

$j \leftarrow$ the child of i in T

while $Z_i \subseteq X_i$ **do**

$lift_j(Z_i) \leftarrow \sum_{Z_j} S_j(Z_j)$ where the summation is over all subsets Z_j of X_j such that $Z_j \cap X_i = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$con_i(Z_i) \leftarrow \sum_A u^{-f(A)} v^{2|A|-f(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $V(G:A) = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$S_i(Z_i) \leftarrow \sum_{Z,Z'} con_i(Z) lift_i(Z')(uv)^{|Z|+|Z'|-|Z \cup Z'|}$ where the summation is over all pairs (Z, Z') such that $Z, Z' \subseteq X_i$ and $Z \cup Z' = Z_i$

end while

Procedure 13 Two-Children

PROC TWO-CHILDREN(i)

j and $k \leftarrow$ the children of i in T

while $Z_i \subseteq X_i$ **do**

while $l \in \{j, k\}$ **do**

$lift_l(Z_i) \leftarrow \sum_{Z_l} S_l(Z_l)$ where the summation is over all subsets Z_l of X_l such that $Z_l \cap X_i = Z_i$

end while

end while

while $Z_i \subseteq X_i$ **do**

$mix_i(Z_i) \leftarrow \sum_{Z_j, Z_k} lift_j(Z_j) lift_k(Z_k)(uv)^{|Z_j|+|Z_k|-|Z_j \cup Z_k|}$ where the summation is over all pairs (Z_j, Z_k) such that $Z_j, Z_k \subseteq X_i$ and $Z_j \cup Z_k = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$con_i(Z_i) \leftarrow \sum_A u^{-f(A)} v^{2|A|-f(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $V(G : A) = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$S_i(Z_i) \leftarrow \sum_{Z,Z'} con_i(Z) mix_i(Z') uv^{|Z|+|Z'|-|Z \cup Z'|}$ where the summation is over all pairs (Z, Z') such that $Z, Z' \subseteq X_i$ and $Z \cup Z' = Z_i$

end while

The only differences this makes occur in the computation of *con* and $leaf$. Computing these two functions amounts to exactly the same computations so we only show how to modify *con*. Now

$$\begin{aligned} con_i(Z) &= \sum_{\substack{A \prec D_i \\ :V(G:A)\cap X_i = Z}} u^{-f(A)} v^{2|A|-f(A)} \\ &= \sum_{\substack{A \subseteq D_i \\ :V(G:A)\cap X_i = Z}} \sum_{(B_1,\ldots,B_{|A|})} u^{-f(A)} v^{2|B_1|+\cdots+2|B_{|A|}|-f(A)} \end{aligned}$$

where if $A = \{e_1, \ldots e_l\}$ then the inner summation is over all l-tuples $(B_1, \ldots B_l)$ such that for all j, B_j is non-empty and contained in the parallel class containing e_j. If $v \neq 0$ then

$$con_i(Z) = \sum_{\substack{A \subseteq D_i \\ :V(G:A)\cap X_i = Z}} (uv)^{-f(A)} \prod_{e \in A} ((v^2+1)^{m(e)} - 1)$$

and if $v = 0$ then

$$con_i(Z) = \sum_{\substack{A \subseteq D_i \\ :V(G:A)\cap X_i = Z, 2|A| = f(A)}} u^{-f(A)} \prod_{e \in A} m(e).$$

To avoid having to calculate $(v^2+1)^j$ repeatedly for the same value of j we can compute the set $\{(v^2+1), (v^2+1)^2, \ldots, (v^2+1)^M\}$, where $M = \max_{e \in E} m(e)$, before running the algorithm.

3.4.3 Complexity

We calculate an upper bound on the running time of our algorithm. We let $s(n, m, k, u, v, M)$ be the maximum number of operations needed to evaluate $S(G; u, v)$ when G is a graph with n vertices, tree-width at most k, m edges and with the maximum size of a parallel class equal to M. We let

$\alpha(n, m, k, u, v, M)$ be the maximum time needed for one arithmetical operation (addition, subtraction, multiplication or division) during the evaluation of $S(G; u, v)$. We let $g(k) = k^5 \cdot (2k+1)^{(2k+1)-2} \cdot ((2(2k+1)+3)^{2(2k+1)+3} \cdot (\frac{8}{3} \cdot 2^{2k+2})^{2(2k+1)+3})^{2(2k+1)-1}$. There are the same four stages of preprocessing as in the previous chapter:

1. Finding a tree-decomposition of width at most k which can be done in time $O(g(k) \cdot n)$ using the algorithm given in [Bod96].

2. Constructing a tree-decomposition, T where T is a binary tree, which requires time $O(nk)$.

3. Computing the partition $\{D_i : i \in I\}$ which requires time $O(m+n)$.

4. Computing the numbers $\{v^2+1, (v^2+1)^2, \ldots, (v^2+1)^M\}$. This needs $O(M\alpha(n, m, k, u, v, M))$.

Because $|I| \leq 2n$ the number of operations needed for the main part of the algorithm, that is omitting the preprocessing, is $O(nt'(n, m, k, u, v, M))$ where t' is the maximum time required for one call to the procedure $TWO-CHILDREN$. It is possible to compute the intersection of two sets each with size at most l in time $O(l^2)$. If X is a set of l vertices of a graph G and A is a set of edges all having both endpoints in X then it is possible to compute $V(G : A) \cap X$ in time $O(l^3)$.

The procedure $TWO - CHILDREN$ consists of 4 stages:

1. Calculation of $lift_j$ and $lift_k$ which requires time $O\left(2^{2(k+1)}(k^2+\alpha)\right)$.

2. Evaluating mix takes time $O\left(2^{3(k+1)}(k^2+k\alpha)\right)$.

3. The computation of con needs $O\left(2^{2(k+1)}(k^3+k^2\alpha)\right)$.

4. The final stage requires time $O\left(2^{3(k+1)}(k^2+k\alpha)\right)$.

Therefore the time for a call to $TWO{-}CHILDREN$ is $O\left(2^{3(k+1)} \cdot k^2 \cdot \alpha\right)$. It now remains to calculate a bound on the total time needed for any arithmetic operation. Time $O(l \log l \log\log l)$ is needed to add, subtract, multiply or divide two numbers with at most l bits [AHU74]. The functions $lift$, mix, con and S_i are all of the form $\sum_{A\in\mathcal{A}} u^{-f(A)}v^{(2|A|-f(A))}$, where $\mathcal{A}$ is a subset of the power-set of E and so the numbers involved in the calculations are either of this form or $(uv)^j$ where $0 \le j \le 2k+2$, or $(v^2+1)^j$ where $0 \le j \le M$. Now let $u = p_u/q_u$ and $v = p_v/q_v$ where p_u, q_u, p_v and q_v are integers such that p_u and q_u are coprime, and p_v and q_v are coprime. Since $u \neq 0$, we have that $p_u \neq 0$. Now

$$\begin{aligned}
\sum_{A\in\mathcal{A}} u^{-f(A)}v^{(2|A|-f(A))} &= \sum_{A\in\mathcal{A}} \left(\frac{p_u}{q_u}\right)^{-f(A)} \left(\frac{p_v}{q_v}\right)^{(2|A|-f(A))} \\
&= \sum_{A\in\mathcal{A}} \left(\frac{\left(\frac{p_u}{q_u}\right)^{(f(E)-f(A))} \left(\frac{p_v}{q_v}\right)^{(2|A|-f(A))}}{\left(\frac{p_u}{q_u}\right)^{f(E)}} \right) \\
&= \sum_{A\in\mathcal{A}} \left(\frac{p_u^{f(E)-f(A)} q_u^{f(A)} p_v^{(2|A|-f(A))} q_v^{(2m-2|A|+f(A))}}{p_u^{f(E)} q_v^{2m}} \right).
\end{aligned}$$

Considering the denominator of this expression, we have

$$p_u^{f(E)} q_v^{2m} \le |p_u|^n |q_v|^{2m},$$

and for the numerator, we have

$$\begin{aligned}
&\sum_{A\in\mathcal{A}} p_u^{f(E)-f(A)} \cdot q_u^{f(A)} \cdot p_v^{(2|A|-f(A))} \cdot q_v^{(2m-2|A|+f(A))} \\
&\qquad \le \sum_{A\subseteq E} |p_u|^{f(E)-f(A)} \cdot |q_u|^{f(A)} \cdot |p_v|^{(2|A|-f(A))} \cdot |q_v|^{2m} \\
&\qquad \le S(G; |p_u|, |p_v|) \cdot |q_u|^{f(E)} \cdot |q_v|^{2m}
\end{aligned}$$

with the last equation following from the definition of S. Using Equation 3.3.4 and induction on the number of edges of G we can show that $|S(G;u,v)| \leq (|u|+|v|+1)^{2m}$ and so

$$S(G;|p_u|,|p_v|)\cdot|q_u|^{f(E)}\cdot|q_v|^{2m} \leq (|p_u|+|p_v|+|q_u|+|q_v|+1)^{4m}.$$

The calculations in the main part of the algorithm all give quantities of the form $\sum_{A\in\mathcal{A}} u^{-f(A)}v^{2|A|-f(A)}$, where $\mathcal{A}$ is a subset of the power-set of E, and so we can avoid problems arising due to the numerator and denominator having a large common factor and thus becoming large themselves, because we can reduce the fraction by 2 integer divisions so that the denominator is $|p_u|^{f(E)}|q_v|^{2m}$.

Hence $\alpha(n,m,k,u,v,M) = O(w\log w\log\log w)$ where $w = 4m\log(|p_u|+|p_v|+|q_u|+|q_v|+1)$ and so the running time for the main part of the algorithm is $O\left(n2^{3(k+1)}\cdot k^2\cdot 4m\cdot\log(|p_u|+|p_v|+|q_u|+|q_v|)\cdot\log w\cdot\log\log w\right)$ and the running time for the whole algorithm is

$$O\left(g(k)\cdot(n+M)\cdot m\cdot\log m\cdot\log\log m\cdot l\log l\log\log l\right)$$

where $l = \log(|p_u|+|q_u|+|p_v|+|q_v|)$.

As we noted in the previous chapter, if a graph with tree-width k has no parallel edges then it has at most $\frac{(k+1)(2n-k)}{2}$ edges and so in this case the total running time is at most

$$O\left(g(k)\cdot n^2k\cdot\log(nk)\cdot\log\log(nk)\cdot l\log l\cdot\log\log l\right).$$

3.4.4 Case $u = 0$

As we remarked earlier, the algorithm given above will not work when $u = 0$ because it will try to divide by zero. We assume to begin with that the input

graph has no parallel edges although as before we allow up to one loop at each vertex. We define $S'_i(Z)$ by

$$S'_i(Z) = \sum_A v^{2|A|-f(A)}$$

where the summation is over all sets A, consisting of edges in E_i such that each vertex of $Y_i \setminus X_i$ is an endpoint of an edge in E_i. The modified algorithm works in much the same way as the main algorithm and is shown as Procedures 14–17.

Procedure 14 Evaluating the Rank Generating Function when $u = 0$

Input: G where G has tree-width $\leq k$, rational number v, a tree-decomposition $(\{X_i | i \in I\}, T = (I, F)\}$ of G with width k and such that T is a binary tree with specified root r, and also the partition $\{D_i | i \in I\}$

$T^* \leftarrow T$

while $T^* \neq \emptyset$ **do**

$i \leftarrow$ a leaf of T^*

if i is a leaf of T **then**

CALL $LEAF'(i)$

else if i has one child in T **then**

CALL $ONE-CHILD'(i)$

else

CALL $TWO-CHILDREN'(i)$

end if

Delete i from T^*

end while

$S(G; 0, v) \leftarrow S'_r(X_r)$

Output: $S(G; 0, v)$

Procedure 15 Leaf′

PROC $LEAF'(i)$

while $Z_i \subseteq X_i$ **do**

$S'_i(Z_i) \leftarrow \sum_A v^{2|A|-f(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $V(G:A) = Z_i$

end while

Procedure 16 One-Child′

PROC $ONE-CHILD'(i)$

$j \leftarrow$ the child of i in T

while $Z_i \subseteq X_i \cap X_j$ **do**

$lift'_j(Z_i) \leftarrow S'_j(Z_i \cup (X_j \setminus X_i))$

end while

while $Z_i \subseteq X_i$ **do**

$con'_i(Z_i) \leftarrow \sum_A v^{2|A|-f(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $V(G:A) = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$S'_i(Z_i) \leftarrow \sum_{Z,Z'} con'_i(Z) lift'_i(Z') u^{|Z|+|Z'|-|Z\cup Z'|}$ where the summation is over all pairs (Z, Z') such that $Z, Z' \subseteq X_i$ and $Z \cup Z' = Z_i$

end while

Procedure 17 Two-Children′

PROC TWO-CHILDREN'(i)

j and $k \leftarrow$ the children of i in T

while $l \in \{j, k\}$ **do**

while $Z_i \subseteq X_i$ **do**

$lift'_l(Z_i) \leftarrow S'_l(Z_i \cup (X_l \setminus X_i))$

end while

end while

while $Z_i \subseteq X_i$ **do**

$mix'_i(Z_i) \leftarrow \sum_{Z_j, Z_k} lift'_j(Z_j) lift'_k(Z_k) v^{|Z_j|+|Z_k|-|Z_j \cup Z_k|}$ where the summation is over all pairs (Z_j, Z_k) such that $Z_j, Z_k \subseteq X_i$ and $Z_j \cup Z_k = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$con'_i Z_i \leftarrow \sum_A v^{2|A|-f(A)}$ where the summation is over all sets, A, of edges contained in D_i and satisfying $V(G : A) = Z_i$

end while

while $Z_i \subseteq X_i$ **do**

$S_i(Z_i) \leftarrow \sum_{Z, Z'} con'_i(Z) mix'_i(Z') v^{|Z|+|Z'|-|Z \cup Z'|}$ where the summation is over all pairs (Z, Z') such that $Z, Z' \subseteq X_i$ and $Z \cup Z' = Z_i$

end while

As before we just consider the crux of the algorithm, namely the procedure $TWO-CHILDREN'$. We show how the procedure computes $S'_i(Z)$ where i is a node with 2 children j and k, when we know S'_j and S'_k. We first compute for all $Z \subseteq X_i$, $lift'_j(Z)$ and $lift'_k(Z)$ where for $l \in \{j,k\}$, $lift'_l$ is given by

$$lift'_l(Z) = \sum_A v^{2|A|-f(A)}$$

where the summation is over all subsets A of E_l such that $V(G : A) = Z \cup (Y_l \setminus X_i)$. This is zero if $Z \not\subseteq X_l$ and otherwise

$$lift'_l(Z) = S'_l(Z \cup (X_l \setminus X_i)).$$

The procedure next calculates mix'_i which is given by

$$mix'_i(Z) = \sum_A v^{2|A|-f(A)}$$

where the summation is over all subsets, A, of $E_j \cup E_k$ such that $V(G : A) = Z \cup (Y_i \setminus X_i)$. We modify Equation 3.4.2 to obtain an expression for mix' in terms of $lift'$.

Suppose we have a graph $G = (V_1 \cup V_2, E_1 \cup E_2)$ with $V_1 \cap V_2 = X$, $E_1 \cap E_2 = \emptyset$ and such that any edge in E_i has both endpoints in V_i. For each $Z \subseteq X$ and for each $i \in \{1,2\}$, we define

$$s_i(Z) = \sum_A v^{2|A|-f(A)}$$

where the summation is over subsets A of E_i satisfying $V(G : A) = Z \cup (V_i \setminus X)$. Now let $s(Z)$ be defined by

$$s(Z) = \sum_A v^{2|A|-f(A)}$$

where the summation is over all subsets A of $E_1 \cup E_2$ such that $V(G : A) = Z \cup ((V_1 \cup V_2) \setminus X)$. Using a modification of the argument preceding Equation 3.4.2 it is straightforward to show that

$$s(Z) = \sum_{\substack{(Z_1,Z_2) \\ :Z_1 \cup Z_2 = Z}} s_1(Z_1) s_2(Z_2) v^{(|Z_1|+|Z_2|-|Z|)}.$$

Therefore we can calculate mix'_i from $lift'_j$ and $lift'_k$, using this equation. The definition of con'_i is unchanged and so we use this equation again to calculate S_i from con'_i and mix'_i. This shows that $TWO-CHILDREN'$ works as claimed and it is easy to check that the rest of the algorithm is correct.

Exactly the same modification as that for the main algorithm must be made to cope with non-simple graphs and it is easy to show that the running time for this algorithm satisfies the bound given for the main algorithm.

Chapter 4

Reidemeister Moves on Graphs

4.1 Introduction

We now move away from complexity and investigate some problems motivated by knot theory. We consider various classes of graphs which are defined so that members of a particular class can be obtained from a graph with no edges using a sequence of allowed local moves such as deleting a loop. Many natural questions arise. For instance, deciding whether a graph belongs to a particular class or finding all containments between the classes. We have results answering some of these problems but there remain many open questions. The connection with graph polynomials is that an evaluation of the Tutte polynomial is an invariant of one of the sets of moves.

The problems discussed here stem from the paper of Schwärzler and Welsh [SW93] where an attempt is made to decide how well the Jones polynomial, or equivalently the bracket polynomial, of a knot diagram detects whether the diagram represents the unknot. As far as we know this problem is still open and is one of the foremost unsolved questions in the theory of link

polynomials. Most of [SW93] is concerned with *signed graphs*, that is ordinary undirected graphs, in which each edge is given a plus or minus sign. Here we are primarily interested in the unsigned case and unless otherwise specified, graphs will be unsigned. We use O_k to denote the graph with k vertices and no edges; a graph with no edges is called an *empty graph*. The class of all graphs is denoted by $\mathcal{G}$. A graph is *simple* if it contains no loops or parallel edges; it is *cosimple* if it contains no isthmuses or vertices of degree 2.

In [SW93] a class of graphs is studied which are called *Reidemeister graphs*. These are the $\mathcal{R}$-graphs defined below. We believe that making a slight change in the definition results in a closer approximation to the actual Reidemeister knot moves and gives the $\mathcal{Q}$-graphs. The two classes are closely related with $\mathcal{Q} \subseteq \mathcal{R}$, though we cannot decide if the containment is strict. The graphic versions of Reidemeister moves are clearly related to the relatively well studied delta-wye moves (see Robertson, Seymour and Thomas [RST93]) which define the class $\mathcal{D}$, the *delta-wye* graphs. Much of this chapter is concerned with transformations related to Reidemeister and delta-wye moves on unsigned graphs. In particular the class $\mathcal{S}$ consists of those graphs which are reducible to an empty graph by the unsigned version of the Reidemeister graph moves. We show that delta-wye graphs form a proper subset of $\mathcal{S}$. We believe, but are unable to prove that $\mathcal{S} \neq \mathcal{G}$. The containments that we know are shown as a partial order in Figure 4.1. We show in Section 4.4, a relationship between the span of the bracket polynomial of a knot and $\mathcal{S}$-equivalence and it is tempting to believe that $\mathcal{S}$-reducibility might be related to linkless or flat embeddings in $\mathbb{R}^3$ as described in [RST93]. However any connection cannot be too straightforward. A prime reason for this is that each of these geometric properties is closed under minors, whereas $\mathcal{S}$-reducibility certainly is not, unless every graph is $\mathcal{S}$-reducible, see Propo-

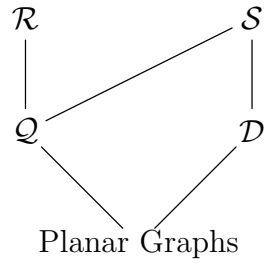


Figure 4.1: Known containments.

sition 4.5.1. However what we do believe is that the "smallest" graphs which are not $\mathcal{S}$-reducible are the seven graphs of the Petersen family.

4.2 Reidemeister moves

In [SW93] the allowable moves on signed graphs consist of the following moves and their inverses.

A Delete any loop, contract any isthmus (or coloop).

B Delete any pair of edges e, f with opposite signs and such that $\{e, f\}$ is a circuit; contract any pair of edges e, f with opposite signs and such that $\{e, f\}$ is a cocircuit.

C Replace a triangle which has edges of both signs as shown in Figure 4.2.

Two signed graphs G, H are called *Reidemeister equivalent*, $G \overset{R}{\sim} H$, if it is possible to transform G to an isomorphic copy of H by some finite sequence of moves (A),(B),(C) and their inverses.

For two *unsigned* graphs G and H we say that G is *Reidemeister reducible, ($\mathcal{R}$-reducible)* to H, denoted by $G \overset{\mathcal{R}}{\to} H$ if there exist signings ω_1, ω_2 such

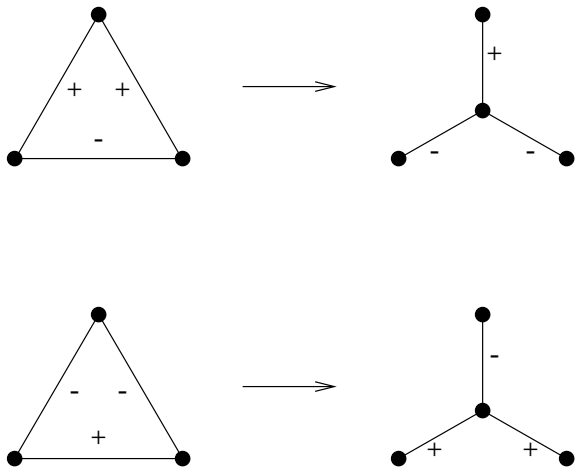


Figure 4.2: Moves of type C.

that $\omega_1(G) \overset{R}{\sim} \omega_2(H)$. We denote by $\mathcal{R}_k$ the class of unsigned graphs which are $\mathcal{R}$-reducible to O_k and let $\mathcal{R} = \bigcup_{k\geq 1} \mathcal{R}_k$. Although it is easy to see that $\mathcal{R}$-equivalence is an equivalence relation on signed graphs, we do not know whether $\mathcal{R}$-reducibility is an equivalence relation on unsigned graphs.

As pointed out in [SW93], the moves (A) – (C) are not the exact equivalents of the Reidemeister moves on knots. We now define a weaker notion of equivalence which is closer to the spirit of the Reidemeister moves. Replace the set of moves (B) by (D).

D Delete any pair of parallel edges e, f with opposite signs; contract any pair of edges e, f with opposite signs and e, f incident with a common vertex of degree 2.

Call two signed graphs G and H, *$\mathcal{Q}$-equivalent*, $G \overset{Q}{\sim} H$ if G can be reduced to an isomorphic copy of H by some finite sequence of the moves (A),(C),(D) and their inverses.

For unsigned graphs, G is $\mathcal{Q}$-*reducible* to H, $G \stackrel{Q}{\to} H$ if there are signings ω_1, ω_2 such that $\omega_1(G)$ and $\omega_2(G)$ are Q-equivalent. Again we do not know whether $\mathcal{Q}$-reducibility is an equivalence relation.

Since (D) is a special case of (B) it is clear that the following is true.

Observation 4.2.1. *If G is $\mathcal{Q}$-reducible to H then G is $\mathcal{R}$-reducible to H.*

Similarly, $\mathcal{Q}_k$, defined as the class of graphs $\mathcal{Q}$-reducible to O_k, is contained in $\mathcal{R}_k$.

For our next result we need a few basic facts from knot theory. A *link* with k components is a subset of $\mathbb{R}^3$ which is homeomorphic to the disjoint union of k circles. Two links K, L are *ambient isotopic* if there exists a homotopy $h_t : \mathbb{R}^3 \to \mathbb{R}^3 \, (0 \leq t \leq 1)$ such that $h_0 = 1$, each h_t is a homeomorphism and $h_1(K) = L$. A knot is usually described by a projection onto a plane. A projection of K is a *regular projection* if it contains only finitely many multiple points and all multiple points are double points and these are crossing points. A regular projection and the specification of which is the under/over crossing for each crossing point determines the knot.

Given any link diagram it is easy to prove that we can colour the faces black and white so that no two faces with a common edge receive the same colour. Conventionally the outer face is coloured black and this colouring is known as a *Tait colouring*. From this we can get a canonical signed graph $S(D)$, its vertices are the black faces of the Tait colouring and two vertices are joined by a signed edge if they share a crossing. The sign of the edge is determined by the rule shown in Figure 4.3. Given any signed plane graph we can construct a link diagram in a canonical way. Given a graph G, the *medial graph* $m(G)$ has vertices at the midpoint of each edge of G and two vertices are adjacent in $m(G)$ if they are on consecutive edges of

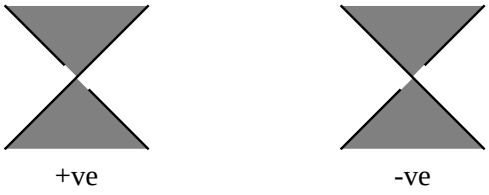


Figure 4.3: Positive and negative crossings.

a face in G. Therefore $m(G)$ is 4-regular and the vertices are the crossings of the link diagram D. The sign of the edge of G determines whether the corresponding crossing is under/over in D. This construction determines a 1-1 correspondence between link diagrams and plane signed graphs. The fundamental theorem of Reidemeister, [Rei48], can be stated in terms of plane signed graphs.

Theorem 4.2.2. *Two plane signed graphs G_1, G_2, represent links which are ambient isotopic if and only if G_1 can be transformed to a graph isomorphic to G_2 by a finite sequence of moves (A), (C), (D) and their inverses.*

A link is said to be *descending* if for each component we can find a point on the string and trace along the string and always take the under crossing the first time we arrive at a particular crossing. It is easy to see that each component of this link is ambient isotopic to the unknot and that the link diagram of the unknot is just O_1. See [Wel93] for more information on knot theory.

Proposition 4.2.3. *$\mathcal{Q}$ contains all planar graphs.*

Proof. Given a planar graph G give it an arbitrary signing ω. Let $L(G)$ be the associated link. Now successively change the under/over crossings of $L(G)$ so that it becomes the descending link L' with say k components.

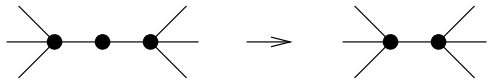


Figure 4.4: Move 2.

Change the signs of the edges of G to get a graph G' such that $L' = L(G')$ and then G' is the signed version of G which is Q-equivalent to O_k. □

However $\mathcal{Q}$ is a larger class than this, for example $K_5 \in \mathcal{Q}$. In fact, as pointed out in [SW93] there are 260 signings of K_5 which show this. In the next section we relate $\mathcal{Q}$ and $\mathcal{R}$ with classes that do not rely on signings.

4.3 ΔY-Equivalence and $\mathcal{S}$-graphs.

We start by recalling the notion of ΔY-equivalence from [RST93]. Let G be a graph with a vertex v of degree 3 which has distinct neighbours. Let H be obtained from G by deleting v and its incident edges and adding an edge between each pair of neighbours of v. We say that H is obtained from G by a $Y\Delta$-*exchange* and that G is obtained from H by a ΔY-*exchange*. Two graphs are ΔY-*equivalent* if one can be obtained from a graph isomorphic to the other by a finite sequence of the following operations and their inverses.

1. Deleting a vertex of degree 1.
2. Suppressing a vertex of degree 2 (that is contracting an edge incident with a vertex of degree 2) as in Figure 4.4.
3. Deleting a parallel edge or a loop.
4. $Y\Delta$-exchange.

We define the class of graphs $\mathcal{D}_k$ by $G \in \mathcal{D}_k$ if G is ΔY-equivalent to O_k. We let $\mathcal{D} = \bigcup_{k=1}^{\infty} \mathcal{D}_k$ and if $G \in \mathcal{D}$ we say G is a *delta-wye graph*. Applying any of moves 1 – 4 does not affect the number of connected components and so a delta-wye graph is in $\mathcal{D}_k$ if and only if it has k components.

We now define the unsigned graphs G and H to be $\mathcal{S}$-*equivalent* if one can be obtained from an isomorphic copy of the other by a finite sequence of the following moves and their inverses.

I. Contract an isthmus.

II. Delete a loop.

III. Delete a pair of parallel edges.

IV. Contract a pair of edges which are not parallel if both are incident with the same vertex of degree 2.

V. $Y\Delta$-exchange.

We write $G \overset{\mathcal{S}}{\sim} H$ if G and H are $\mathcal{S}$-equivalent. Clearly this is an equivalence relation. As we said earlier, the motivation for this is that this set of moves can be regarded as the unsigned graphical version of the Reidemeister moves on knots. For each positive integer k, we define the class $\mathcal{S}_k$ by, $G \in \mathcal{S}_k$ if and only if $G \overset{\mathcal{S}}{\sim} O_k$. Then $\mathcal{S} = \bigcup_{k=1}^{\infty} \mathcal{S}_k$.

A sequence of moves reducing G to an empty graph is called a *reduction sequence*. To make it clear that the moves in a reduction sequence are for instance the $\mathcal{S}$ moves we will sometimes refer to a reduction sequence as an $\mathcal{S}$-reduction sequence.

Observation 4.3.1. *If G,H belong to $\mathcal{S}_j$,$\mathcal{S}_k$ respectively then $G \cup H$ belongs to $\mathcal{S}_{j+k}$.*

Our first results link $\mathcal{S}$ with existing classes.

Proposition 4.3.2. *$\mathcal{S}$ contains $\mathcal{Q}$.*

This is easy and follows from

Proposition 4.3.3. *For each positive integer k, $\mathcal{Q}_k \subseteq \mathcal{S}_k$.*

Proof. Suppose $G \in Q_k$. Then there exists some signing ω, such that $\omega(G) \overset{\mathcal{Q}}{\sim} O_k$. Now follow the $\mathcal{Q}$-reduction sequence $\omega(G) \overset{\mathcal{Q}}{\sim} \cdots \overset{\mathcal{Q}}{\sim} O_k$ replacing each signed move from the set $\{(A), (C), (D)\}$ together with their inverses with its unsigned equivalent from moves I – V and their inverses giving an $\mathcal{S}$-reduction from G to O_k. □

Every planar graph is contained in $\mathcal{Q}$ by Proposition 4.2.3 and so we have the following corollary.

Corollary 4.3.4. *$\mathcal{S}$ contains all planar graphs.*

Moreover this containment is strict as we have the following result.

Proposition 4.3.5. *$\mathcal{S}$ contains all delta-wye graphs.*

The proof of this result needs the following lemmata which may be known but we have been unable to find in the literature, although the proof ideas are certainly contained in [Tru89].

Lemma 4.3.6. *Suppose G is in $\mathcal{D}$, and there is a $\mathcal{D}$-reduction sequence for G in which the number of edges never increases. Now let H be a minor of G, then H is in $\mathcal{D}$ and there exists a $\mathcal{D}$-reduction sequence for H in which the number of edges never increases.*

Proof. The proof is by induction on the number of edges in H plus the number of moves in the $\mathcal{D}$-reduction sequence for G. The result is clearly true if H has no edges. Otherwise we may assume that H is simple and cosimple, if not we could apply one of moves 1 – 4 to remove an edge from H and we could then use induction. Let G' be the first graph obtained in the $\mathcal{D}$-reduction sequence of G. If G' is obtained from G by one of moves 1 – 4 then H is a minor of G' because H is simple and cosimple and H is a minor of G. Suppose the first move in the $\mathcal{D}$-reduction sequence of G is a $Y\Delta$-exchange. If the three edges involved are contained in H then because H is simple and cosimple they join a vertex of degree three to three distinct vertices so apply the same $Y\Delta$-exchange to H giving H'. Now H' is a minor of G'. If any of the three edges involved is not present in H then H is a minor of G'. The case when the first move in the $\mathcal{D}$-reduction sequence of G is a ΔY-exchange is similar. Either one of the three edges involved is not present in H in which case H is a minor of G' or the three edges form a triangle and so applying a ΔY-exchange to H gives a graph H' which is a minor of G'. In each case we have obtained a graph with the same number of edges as H and which is a minor of G', and so by induction H can be reduced to the empty graph without increasing the number of edges at any stage. □

Lemma 4.3.7. *If $G \in \mathcal{D}$ then there is a $\mathcal{D}$-reduction sequence for G in which the number of edges never increases.*

Proof. If the lemma is not true for G then any $\mathcal{D}$-reduction sequence for G must contain a move where the number of edges increases. Consider a $\mathcal{D}$-reduction sequence for G where the number of moves which increase the number of edges is minimised. Suppose this $\mathcal{D}$-reduction sequence is of the form $G \to \ldots \to H \to H' \to \ldots \to O_{k(G)}$ and the transition $H \to H'$

is the last move which increases the edge number. Hence H is a minor of H' and there is a decreasing $\mathcal{D}$-reduction sequence for H' so applying the preceding lemma implies that there is a $\mathcal{D}$-reduction sequence for H in which the number of edges never increases. This contradicts the choice of $\mathcal{D}$-reduction sequence for G. □

The following lemma is well known folklore and is immediate from the two preceding lemmata.

Lemma 4.3.8. *The class $\mathcal{D}$ is closed under minors.*

Hence from Robertson-Seymour theory we know that there is a finite collection of forbidden minors. This set is known to include K_6 and the Petersen family, that is those graphs which can be obtained from K_6 by a finite sequence of $Y\Delta$ and ΔY moves but seems to be much larger.

Proof of Proposition: We use induction on the number of edges in G. If G has no edges then clearly $G \in \mathcal{S}$ so suppose G has k edges and $G \in \mathcal{D}$. Find any $\mathcal{D}$-reduction sequence for G in which the number of edges never increases. This sequence will begin with a possibly empty sequence of $Y\Delta$ and ΔY-exchanges to form H and then make a move that reduces the number of edges. Suppose the next move is to delete a loop e from H. Now $H \backslash e \in \mathcal{D}$ and has $k-1$ edges and so by induction $H \backslash e \in \mathcal{S}$ but this means that $H \in \mathcal{S}$. The same argument works if the next move is to contract a pendant edge. Now suppose the next move is to delete e where e and f are a pair of parallel edges. We have that $H \setminus \{e, f\} \in \mathcal{D}$ because $\mathcal{D}$ is closed under minors and $H \backslash \{e, f\}$ is a minor of H. Now $H \backslash \{e, f\}$ has $k-2$ edges and so by induction it is in $\mathcal{S}$ which means that H and hence G belong to $\mathcal{S}$. The case where the next move is to suppress a vertex of degree two is similar.

□

However $\mathcal{S}$ is much larger than $\mathcal{D}$ for the following reasons. The *2-thickening* of G, denoted by $G^{(2)}$, is formed by replacing every edge of G by two parallel edges and the *2-stretch* of G is formed by replacing every edge of G by two edges in series. An obvious fact is

Proposition 4.3.9. *For any graph, both its 2-thickening and 2-stretch are in* $\mathcal{S}$.

Example. For any $k \geq 6$, $K_k^{(2)}$ belongs to $\mathcal{S}$ and is not a delta-wye graph.

What seems to be more difficult is to prove that there is a graph which does not belong to $\mathcal{S}$.

Problem 4.3.10. *Does* $\mathcal{S}$ *contain all graphs ?*

We believe not, indeed we believe that K_6 and all other members of the Petersen family are not in $\mathcal{S}$, see Problem 4.5.6 at the end of the chapter.

4.4 The bracket and Tutte polynomials

We start by recalling the definition of the *Tutte polynomial*. This is the two-variable polynomial defined by

$$T(G; x, y) = \sum_{A \subseteq E} (x-1)^{r(E)-r(A)} (y-1)^{|A|-r(A)}$$

where E is the edge set of G. One evaluation of T which is particularly relevant here is

$$|T(G; -1, -1)| = 2^{\dim(\mathcal{C} \cap \mathcal{C}^*)}$$

where $\mathcal{C}$ and $\mathcal{C}^*$ are respectively the cycle and cocycle spaces of G, (see [SW93]). The point $(-1, -1)$ is one of the very few points in the (x, y)

plane at which there exists a polynomial time algorithm to evaluate T, see [JVW90].

In [SW93] the definition of the bracket polynomial of a knot is extended to any matroid and hence to any graph. As far as this chapter is concerned for a graph G the *bracket polynomial* is a Laurent polynomial in the variable A given by

$$< G; A > = A^{|E^-|-|E^+|-2r(E)}(-A^2 - A^{-2})^{k(G)-1} \cdot \sum_{X \subseteq E} A^{4(r(X)-|X^-|)}(-A^4 - 1)^{r(E)+|X|-2r(X)}$$

where X^+ (X^-) denotes the set of positive (negative) edges of X.

When G is planar, along the hyperbola $xy = 1$, T evaluates the bracket polynomial of the alternating link $L(G)$ determined by G, (see [SW93]).

First we give a necessary condition for G to belong to $\mathcal{S}_k$ in terms of the Tutte polynomial. For any G define

$$\mu(G) = \dim(\mathcal{C} \cap \mathcal{C}^*) + k(G).$$

Theorem 4.4.1. *The integer μ is an invariant of $\mathcal{S}$-equivalence.*

Corollary 4.4.2. *For any positive integer k a necessary condition for a graph G to belong to $\mathcal{S}_k$ is that $k = \log_2(|T(G; -1, -1)|) + k(G)$.*

Proof. This is immediate from Theorem 4.4.1 because $T(O_k; -1, -1) = 1$. □

Corollary 4.4.3. *Given a graph $G \in \mathcal{S}$, we have a polynomial time algorithm to decide which class $\mathcal{S}_k$ contains G.*

Proof of Theorem: Let $t(G) = T(G; -1, -1)$. We will show that $\mu(G)$ is preserved under moves I – V and their inverses. Suppose we are applying

one of moves I – V to G and the move acts on the component H. Repeated use of Equations 1.4.1–1.4.3 gives the following.

Move I If e is an isthmus of H then $t(H/e) = -t(H)$.

Move II If e is a loop then $t(H \setminus e) = -t(H)$.

Move III If e and f are parallel edges, there are two cases depending on whether deleting e and f disconnects H. Suppose first that $H \setminus \{e, f\}$ is connected.

$$\begin{aligned} t(H) &= t(H \setminus e) + t(H/e) \\ &= t(H \setminus \{e, f\}) + t(H \setminus e/f) - t(H/e \setminus f) \\ &= t(H \setminus \{e, f\}). \end{aligned}$$

Now suppose that $H \setminus \{e, f\}$ is disconnected and that H_1 and H_2 are the connected components of $H \setminus \{e, f\}$.

$$\begin{aligned} t(H) &= t(H \setminus e) + t(H/e) \\ &= -t(H \setminus e/f) - t(H/e \setminus f) \\ &= -2t(H_1)t(H_2). \end{aligned}$$

Move IV If e and f are incident on a vertex of degree two and not in parallel.

$$\begin{aligned} t(H) &= t(H \setminus e) + t(H/e) \\ &= -t(H \setminus e/f) + t(H/e \setminus f) + t(H/\{e, f\}) \\ &= t(H/\{e, f\}). \end{aligned}$$

Move V Suppose v is a vertex of degree 3, with distinct neighbours x, y and z. Let e, f and g be the edges $\{v, x\}$, $\{v, y\}$ and $\{v, z\}$ respectively.

Let H' denote the graph formed from H by applying the $Y\Delta$-exchange. Let e', f' and g' be the edges $\{y,z\}$, $\{x,z\}$ and $\{x,y\}$ respectively, that is those edges that are added in the $Y\Delta$-exchange. There are 3 cases to consider depending on how many of e, f and g are isthmuses. In each of the cases the expressions are not asymmetric because various terms have been cancelled, for instance in the first case $H \setminus \{e,g\}/f = H \setminus \{e,f\}/g$. First suppose none of e, f and g are isthmuses. Then

$$\begin{aligned} t(H) &= t(H \setminus \{e,f\}) + t(H \setminus e/f) + t(H \setminus f/e) + t(H/\{e,f\}) \\ &= t(H \setminus \{e,g\}/f) + t(H \setminus e/\{f,g\}) + t(H \setminus f/\{e,g\}) \\ &\quad + t(H \setminus g/\{e,f\}) + t(H/\{e,f,g\}) \end{aligned}$$

and

$$\begin{aligned} t(H') &= t(H' \setminus \{e',f'\}) + t(H' \setminus e'/f') + t(H' \setminus f'/e') + t(H/\{e',f'\}) \\ &= t(H' \setminus \{e',f',g'\}) + t(H' \setminus \{e',f'\}/g') + t(H' \setminus \{e',g'\}/f') \\ &\quad + t(H' \setminus \{f',g'\}/e') + t(H' \setminus f'/\{e',g'\}). \end{aligned}$$

So $t(H) = t(H')$. The second case is when e, say, is an isthmus but f and g are not. Then

$$\begin{aligned} t(H) &= -t(H/e) \\ &= t(H \setminus f/\{e,g\}) - t(H \setminus g/\{e,f\}) - t(H/\{e,f,g\}) \\ &= -t(H/\{e,f,g\}), \end{aligned}$$

and

$$\begin{aligned} t(H') &= t(H' \setminus f') + t(H'/f') \\ &= -t(H' \setminus f'/g') + t(H'/\{f',g'\}) + t(H' \setminus g'/f') \\ &= -t(H' \setminus e'/\{f',g'\}). \end{aligned}$$

So again $t(H) = t(H')$. It is not possible for exactly two of $\{e, f, g\}$ to be isthmuses so suppose they all are. In this case,

$$t(H) = -t(H/\{e, f, g\})$$

and

$$\begin{aligned} t(H') &= t(H' \setminus e') + t(H'/e') \\ &= t(H' \setminus e'/\{f', g'\}) - t(H' \setminus f'/\{e', g'\}) - t(H' \setminus g'/\{e', f'\}). \end{aligned}$$

This shows that $\mu = \log_2(|t(G)|) + k(G)$ is preserved under moves I – V and their inverses. □

It is possible that the converse of Corollary 4.4.2 holds. It is clearly related to Problem 4.3.10.

A key question which we are unable to answer completely is which if any of these classes contain K_6. It is well known that $K_6 \notin \mathcal{D}$. We are fairly certain that $K_6 \notin \mathcal{S}$ but all we can show is the following:

Proposition 4.4.4. *$K_6 \notin \mathcal{R}$.*

Proof. A graph $G \in \mathcal{R}_k$ if and only if there exists a signing ω such that $\omega(G) \overset{R}{\sim} O_k$. It is not difficult to show that if $G \overset{R}{\to} H$ then $\mu(G) = \mu(H)$ because the only extra move that must be checked in addition to the $\mathcal{S}$ moves is when we apply Move B to contract a pair of edges which form a cocircuit and checking this is essentially the same as showing that μ is preserved under Move II.

The *span* of a signed graph, G, is the difference between the largest and smallest powers with non-zero coefficients in the bracket polynomial of G. Since by [SW93] span is preserved under $\mathcal{R}$-moves we have that $\omega(G) \overset{R}{\sim} O_k$ implies $\operatorname{span} \omega(G) = \operatorname{span} O_k$. The bracket polynomial of O_k is

$$< O_k; A >= (-A^2 - A^{-2})^{k-1}$$

so that $\operatorname{span} O_k = 4(k-1)$. In [SW93], it was reported that a computer search of all possible signings of K_6 showed that no signing had span less than 24. However

$$T(K_6; -1, -1) = 16$$

and so if $K_6 \overset{R}{\sim} O_k$, then $k = 5$ but $\operatorname{span} O_5 = 16 < 24$ □

At the moment we know of no graphs which would show that any of the sets $\mathcal{S} \setminus \mathcal{Q}$, $\mathcal{S} \setminus \mathcal{R}$, $\mathcal{R} \setminus \mathcal{S}$ or $\mathcal{R} \setminus \mathcal{Q}$ is non-empty.

4.5 How large is $\mathcal{S}$?

A basic question which we are unable to answer is whether $\mathcal{S} = \mathcal{G}$. A first slightly surprising result is

Proposition 4.5.1. *If $\mathcal{S}$ is closed under either deletion or contraction then $\mathcal{S} = \mathcal{G}$.*

Proof. Suppose $\mathcal{S}$ is closed under deletion. Let G be any graph. We know by Proposition 4.3.9 that its 2-thickening, $G^{(2)}$ belongs to $\mathcal{S}$. We can obtain G from $G^{(2)}$ by deleting edges and so G belongs to $\mathcal{S}$.

Now suppose $\mathcal{S}$ is closed under contraction. The 2-stretch of any graph G belongs to $\mathcal{S}$ and from this we can obtain G by contracting edges. □

The following refinement of this argument shows that the somewhat "cheating" nature of just doubling up edges is not relevant.

Proposition 4.5.2.

1. *If for every simple and cosimple G that is in $\mathcal{S}$ and for every edge e of G, $G \setminus e$ is a member of $\mathcal{S}$ then $\mathcal{S} = \mathcal{G}$.*

2. *If for every simple and cosimple G that is in $\mathcal{S}$ and for every edge e of G, G/e is a member of $\mathcal{S}$ then $\mathcal{S} = \mathcal{G}$.*

Proof.

1. Suppose that for every simple and cosimple G that is in $\mathcal{S}$ and for every edge e of G, $G \setminus e$ is a member of $\mathcal{S}$. Every graph is $\mathcal{S}$-equivalent to a simple, cosimple graph. It is enough to show that for any $n \geq 4$, K_n is a member of $\mathcal{S}$ because then we can successively delete edges from K_n to show that any simple, cosimple and connected graph with n vertices is contained in $\mathcal{S}$ and so using Observation 4.3.1 any simple and cosimple graph is a member of $\mathcal{S}$. Let H be the wheel with n vertices, $\{v_1, \ldots, v_n\}$ where v_1 is at the centre of the wheel. H is planar and so by Corollary 4.3.4 $H \in \mathcal{S}$. We now show that we can successively add each edge between vertices on the rim of the wheel and always obtain a graph in $\mathcal{S}$. See Figure 4.5 for an illustration of the following procedure. Suppose we wish to add the edge $\{v_i, v_j\}$ where $i, j \geq 2$ and that H_1, the first graph in Figure 4.5, is the graph we have obtained so far. First use the inverse of move IV to split v_1 into three vertices v_1^1, v_1^2 and v_1^3 so that v_1^1 is joined to v_i, v_j and v_1^2, v_2^2 is joined to v_1^1 and v_1^3, and v_1^3 is joined to every vertex except v_i, v_j and v_1^1. Now v_1^1 has degree 3 so we can apply move V obtaining H_2, the third graph in Figure 4.5. There is now an edge between v_i and v_j and H_2 is simple and cosimple so by our hypothesis $H_2 \setminus \{v_i, v_j\}$ is in $\mathcal{S}$. In $H_2 \setminus \{v_i, v_j\}$, v_1^2 has degree 3 and so we can apply move V again which gives H_1 with $\{v_i, v_j\}$ added and so H_1 with the additional edge $\{v_i, v_j\}$ is a member of $\mathcal{S}$.

2. Now suppose that for every simple, cosimple member G of $\mathcal{S}$, and for

every edge e of G, G/e is in $\mathcal{S}$. We can again show that for any n, K_n is in $\mathcal{S}$. Start from the wheel on n vertices and successively add edges as in the first half of the proof. Suppose we have H_1 and wish to add $\{v_i, v_j\}$. We follow the first two moves in the procedure shown in Figure 4.5 to give H_2, which is simple and cosimple, and so by our initial assumption $H_2/\{v_1^2, v_1^3\}$ is contained in $\mathcal{S}$, but $H_2/\{v_1^2, v_1^3\}$ is just H_1 with $\{v_i, v_j\}$ added. Having obtained K_n we successively delete some of the edges $\{v_i, v_j\}$, with $i \neq 1$ and $j \neq 1$ and so that the graph obtained at each stage is simple and cosimple, by applying the procedure shown in Figure 4.6 to delete an edge. In the first three steps we apply the inverse of move IV, move V and then move IV, giving a graph which is either simple or cosimple or in which the edge $\{v_1^3, v_1^4\}$ is an isthmus. Thus we can contract $\{v_1^3, v_1^4\}$ either by our assumption or by using move I. This means that any simple, cosimple, connected graph with a vertex that is joined to every other vertex, is a member of $\mathcal{S}$.

Now suppose G is a simple, cosimple, connected graph on n vertices which has a vertex v of degree at most $n-3$. Let H be formed from G by deleting the edges incident with v, adding the edge $\{v, w\}$ for each w such that $\{v, w\}$ was not an edge of G, and finally adding a new vertex u adjacent to every vertex of G. Now H is simple, cosimple and has a vertex that is incident with every other vertex and so under our assumption about $\mathcal{S}$, $H \in \mathcal{S}$. Also $H/\{u, v\}$ is a member of $\mathcal{S}$ and after deleting some pairs of parallel edges from H we obtain G. The only simple, cosimple connected graphs that we have not yet shown to belong to $\mathcal{S}$ are those which can be formed from K_n by deleting a matching. For these we can apply the technique used to delete edges, and shown in Figure 4.6 directly and delete each edge in turn. Now

Observation 4.3.1 and the fact that every graph is $\mathcal{S}$-equivalent to a simple and cosimple graph, imply that if for every simple, cosimple graph G and for every edge e of G, $G/e \in \mathcal{S}$ then $\mathcal{S} = \mathcal{G}$.

□

Note. It is easy to find examples which show that for each positive integer k, $\mathcal{S}_k$ is not closed under either deletion or contraction.

It is obvious that there exist pairs of graphs which are ΔY-equivalent but not $\mathcal{S}$-equivalent, for example any two empty graphs with a different number of vertices. To avoid this triviality we say that G and H are $\mathcal{T}$*-equivalent*, if one can be obtained from an isomorphic copy of the other by a finite sequence of moves of type I – V and their inverses together with the additional move VI.

VI. Add or delete an isolated vertex.

We define $\mathcal{T}$ to be the class of graphs which are $\mathcal{T}$-equivalent to a single vertex.

Note. Clearly $\mathcal{S} \subseteq \mathcal{T}$ but again we do not know whether the containment is proper.

There is a result for $\mathcal{T}$ similar to Proposition 4.5.1, namely,

Proposition 4.5.3. *If $\mathcal{T}$ is closed under either deletion or contraction then $\mathcal{T} = \mathcal{G}$.*

The proof of this result is the same as the proof of Proposition 4.5.1.

A question which is very close to Problem 4.3.10 is the following.

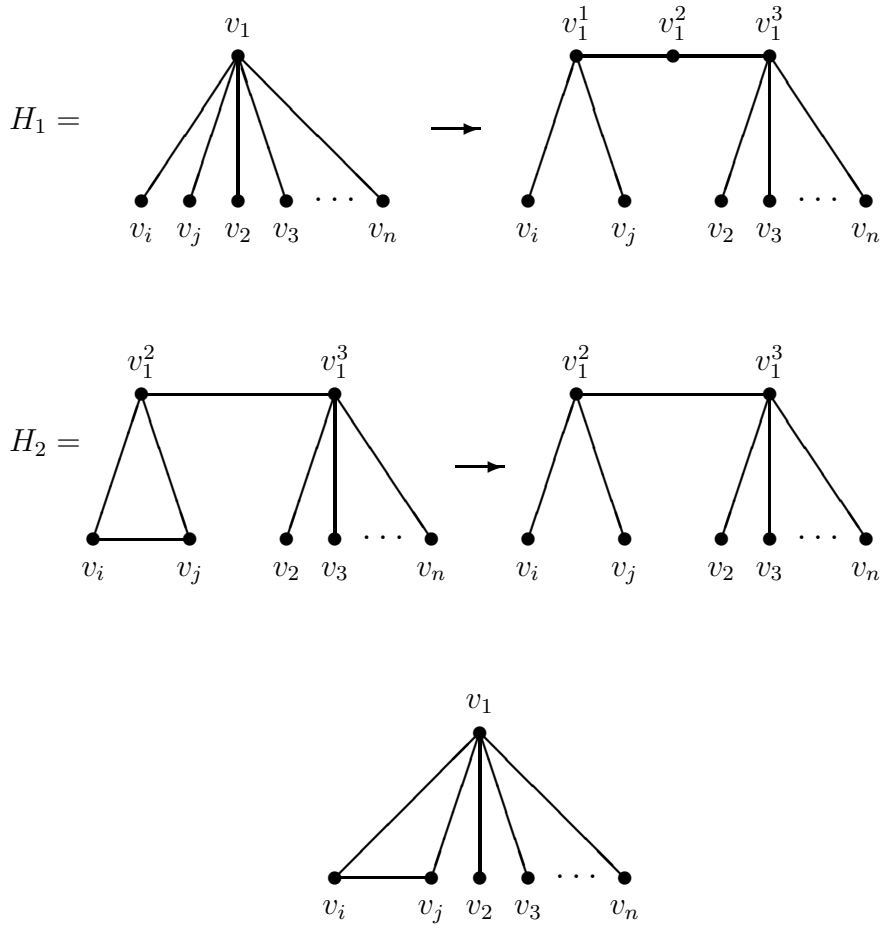


Figure 4.5: Adding an Edge.

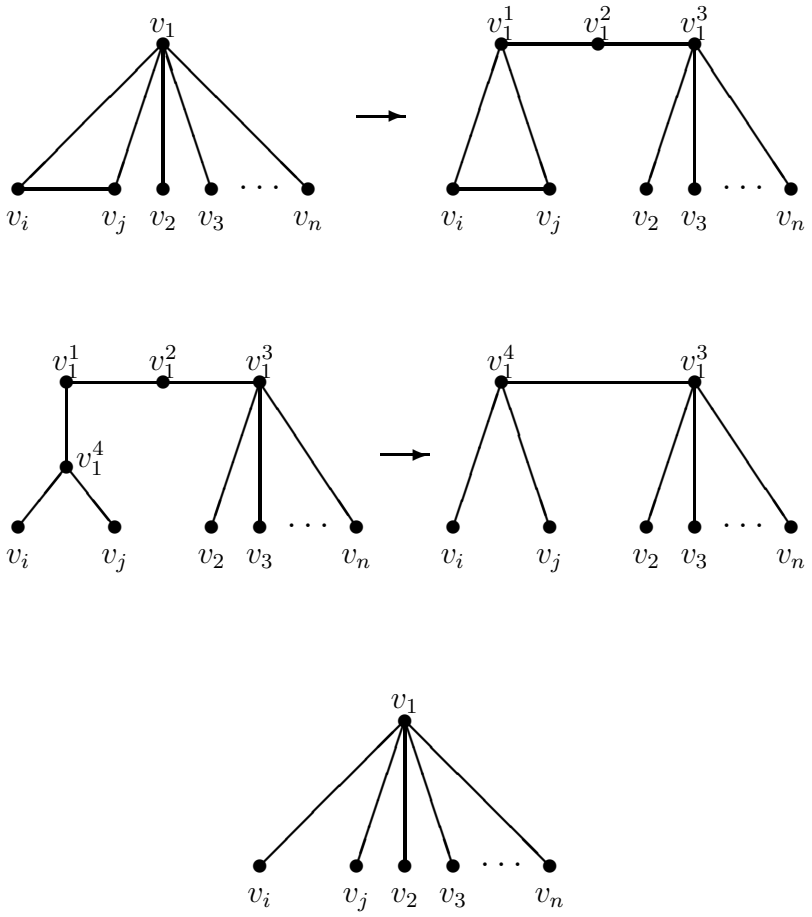


Figure 4.6: Removing an Edge.

Problem 4.5.4. *Is it true that any pair of graphs are $\mathcal{T}$-equivalent ?*

The only slight progress we have made on this is the following.

Proposition 4.5.5. *If $\mathcal{T} \neq \mathcal{G}$ then there exist non-isomorphic graphs, G and H, which are ΔY-equivalent but which are not $\mathcal{T}$-equivalent.*

Proof. Suppose $\mathcal{T} \neq \mathcal{G}$. We will show that if all pairs of ΔY-equivalent graphs are $\mathcal{T}$-equivalent then $\mathcal{T}$ is closed under deletion and so $\mathcal{T} = \mathcal{G}$, a contradiction.

Suppose that all ΔY-equivalent graphs are $\mathcal{T}$-equivalent and let $G \in \mathcal{T}$. Let e be any edge of G. If e is a loop then $G \setminus e$ is related to G using move II. Otherwise G is ΔY-equivalent to the graph H formed from G by adding an edge parallel to e and so G is $\mathcal{T}$-equivalent to H. Now H contains two parallel edges e and f and so we can delete them, using move III, to obtain $G \setminus e$. Therefore we have shown that $\mathcal{T}$ is closed under deletion. □

A natural hypothesis is that $\mathcal{S}$ consists of those graphs G which are obtainable from some delta-wye graph by a finite sequence of moves I – IV and their inverses. However this is not true as the following example shows.

Example. Let G be the graph in Figure 4.7. If G is obtainable from some $H \in \mathcal{D}$ by a finite sequence of moves I – IV and their inverses then because G is simple and cosimple G must be a minor of H. It is easy to see that G contains K_6 as a minor and so H must also contain K_6 as a minor which contradicts the assumption that $H \in \mathcal{D}$ because the class $\mathcal{D}$ is closed under minors and K_6 is not contained in $\mathcal{D}$. However it is not difficult to show that G is contained in $\mathcal{S}$.

A key question which we are unable to settle is the following.

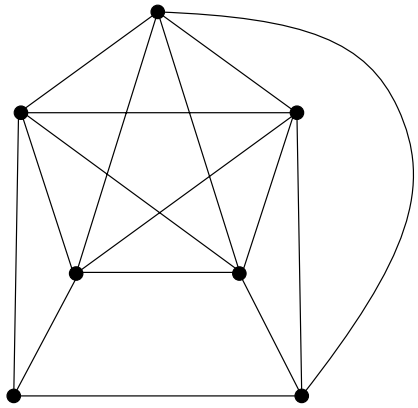


Figure 4.7: The graph G.

Problem 4.5.6. *Do any of the graphs in the Petersen family belong to $\mathcal{S}$?*

We believe not. Since any member of the Petersen family can be obtained from any other by ΔY and $Y\Delta$-exchanges we know that if any one of the Petersen family belongs to $\mathcal{S}$ then they all do. Using Corollary 4.4.2 we know that if a member of the Petersen family belongs to $\mathcal{S}$ then it belongs to $\mathcal{S}_5$.

4.6 Conclusion

One of the main purposes of this chapter has been to sort out the relationships between the classes of graphs defined by closely related families of moves. At the risk of repetition it may be useful to list again the open problems which are particularly frustrating us.

1. Does $\mathcal{S} = \mathcal{G}$? We believe not.

2. Do any of the Petersen family belong to $\mathcal{S}$?

Computationally it would be very useful to have answers to the following questions.

3 If G is $\mathcal{S}$-equivalent to the empty graph does there exist an $\mathcal{S}$-reduction sequence which achieves this and never increases the number of edges? If this is true then $\mathcal{S}$ can contain no simple, cosimple, triangle free graph with minimum degree 4. It would also show that $\mathcal{S}$ contains no member of the Petersen family.

4 Is membership in $\mathcal{S}$ decidable? The answer is clearly yes if (3) is true.

Chapter 5

Weighted Graphs and Vassiliev Invariants

5.1 Introduction

Our final chapter is motivated by the recent work of Chmutov, Duzhin and Lando [CDL94a, CDL94b, CDL94c] on Vassiliev invariants. These are knot invariants that have recently received much attention. We begin with a brief summary of the results which motivate our work and then describe the contents of this chapter.

It is known from Kontsevich [Kon93] that over a field of characteristic zero, the study of Vassiliev invariants of knots is equivalent to the study of weight functions on a combinatorially defined Hopf algebra of chord diagrams. Although, regarded as a Hopf algebra, this algebra is relatively easy to describe, it is in fact a very complicated object, for example as far as we are aware, the number of primitive generators is unknown for degrees greater than nine. In a series of three papers [CDL94a, CDL94b, CDL94c],

the authors examine the combinatorial aspect of this algebra.

A key concept in the treatment in [CDL94c] is the Hopf algebra homomorphism associating a chord diagram with its intersection graph, and this turns out to be just a map on weighted graphs modulo a weighted chromatic relation. This is essentially the same as the study of a chromatic type polynomial on weighted graphs. However, as the authors point out, the treatment given in [CDL94c] does not allow an extension to more general Tutte Grothendieck invariants.

This work originated in an attempt to clarify the reasons for this and we show that a fairly simple modification of the definition of [CDL94c] does allow such an extension. This gives a polynomial of weighted graphs which contains the invariants of [CDL94c] as a specialisation but moreover has a wide range of other specialisations in combinatorics. These include the Tutte polynomial, stability polynomial and matching polynomial of ordinary graphs and a polymatroid polynomial studied by Oxley and Whittle [OW93].

The plan of this chapter is as follows. In Sections 5.2–5.4 we give a brief overview of the relationship between Vassiliev invariants, chord diagrams and weight functions. In Section 5.5 we introduce weighted graphs and our weighted polynomial W. In Section 5.6 we study it as an invariant of ordinary graphs and in Section 5.7 relate this specialisation to a symmetric function generalisation of the chromatic polynomial introduced by Stanley [Sta95]. We close with a discussion of complexity issues and prove hardness results for very restricted classes of graphs.

5.2 Vassiliev invariants and chord diagrams

We assume familiarity with the basics of classical combinatorial knot theory, at a level equivalent to say [Wel93]. A key notion in the Vassiliev theory is that of a *singular knot* and its associated diagram. In a singular knot certain crossings are called *double points*. Geometrically they are points where the knot intersects itself and the tangent lines cross as shown in Figure 5.1. An

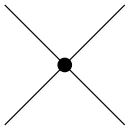


Figure 5.1: A double point.

example of a singular knot with three double points is shown in Figure 5.2. Equivalence of singular knot diagrams can be characterised by an extension

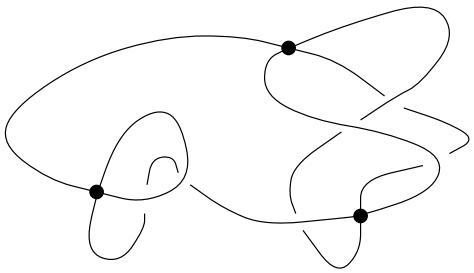


Figure 5.2: A knot with three double points.

of Reidemeister's theorem to allow the moves in Figure 5.3.

Given an orientation of a singular knot K a double point can be eliminated or unfolded to create either a positive or negative crossing, K_+ or K_-, as shown in Figure 5.4.

If $\mathcal{K}^n$ denotes the collection of singular knots with n double points, and $\mathcal{K} = \bigcup_{n\geq 0} \mathcal{K}^n$, then a *Vassiliev invariant* is any function V on $\mathcal{K}$ which takes

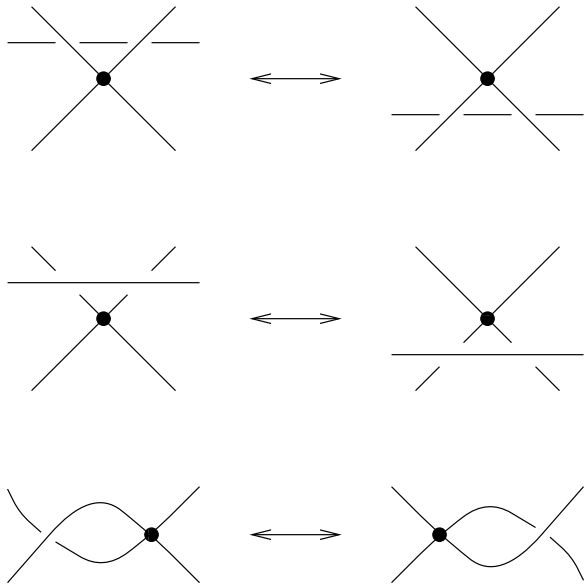


Figure 5.3: Additional Reidemeister moves.

values in some abelian group A, which is invariant under isotopy and satisfies

$$V(K) = V(K_+) - V(K_-).$$

The invariant V is said to be of *order* n if V vanishes on any knot of more than n double points.

A useful proposition is the following.

Proposition 5.2.1. *Let V be a Vassiliev invariant and let the double points of $K \in \mathcal{K}^n$ be $s_1, s_2, \ldots, s_n$. For each i, $0 \leq i \leq 2^n - 1$ let K_i be the (ordinary) knot diagram obtained from K by unfolding the double points of K positively if $\epsilon_r = 0$ and negatively if $\epsilon_r = 1$, for $1 \leq r \leq n$, and*

$$i = \epsilon_1 + \epsilon_2 2 + \epsilon_3 2^2 + \cdots + \epsilon_n 2^{n-1},$$

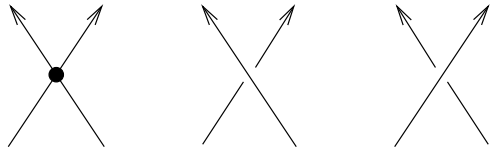


Figure 5.4: Unfolding a double point.

then

$$V(K) = \sum_{i=0}^{2^n-1} (-1)^{c(i)} V(K_i)$$

where $c(i)$ is the number of ϵ_j which are one in its binary expansion above.

Using this, any ordinary (classical) knot invariant taking numerical values can be extended to a Vassiliev invariant on singular knots.

A *chord diagram* of *degree m* is a circle, oriented anti-clockwise, together with m chords marked on it which we regard as orientation preserving diffeomorphisms of the circle. Thus what we are really considering is arrangements of pairs of distinct points on the circle. For example, as shown in Figure 5.5, there are exactly two chord diagrams of degree 2.

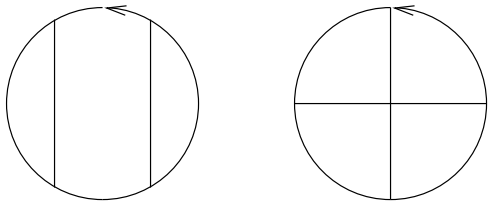


Figure 5.5: Chord diagrams of degree two.

If $u : S_1 \to \mathbb{R}^3$ is a singular knot with n double points $s_1, \ldots, s_n \in \mathbb{R}^3$ then the chord diagram of the singular knot u is the circle with n pairs of

points, which are the pre-images $\{u^{-1}(x_1), \ldots, u^{-1}(x_n)\}$, shown as chords. For example the chord diagrams of the two knots in Figure 5.6 are those shown in Figure 5.5.

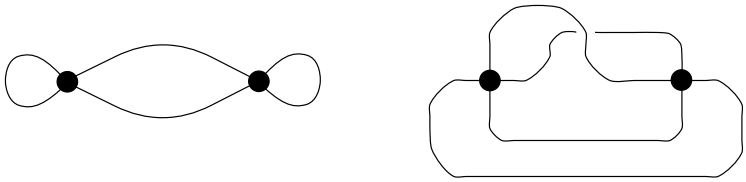


Figure 5.6: Singular knots with two double points.

The theory of Kontsevich says that if V is a Vassiliev invariant of order n then its value on any $K \in \mathcal{K}^n$ is determined by the chord diagram of K.

A *weight function of degree n* is a function ω, mapping all chord diagrams with n chords into an abelian group A and satisfying the two relations (1T) and (4T) listed below.

(1T) The *one term relation* says that if chord diagram D has a chord which does not meet another then $\omega(D) = 0$.

(4T) The *four term relation* is more complicated and says that if D_N,D_S,D_E,D_W differ only as shown in Figure 5.7 then

$$\omega(D_N) - \omega(D_S) = \omega(D_W) - \omega(D_E).$$

Also it is assumed that no chord, other than those shown, has an endpoint in the shaded areas.

The two key results are the following

Theorem 5.2.2. *If V is a Vassiliev invariant of order n with values in the abelian group A then there is a naturally defined weight function $\omega(V)$ on chord diagrams of degree n and taking values in A.*

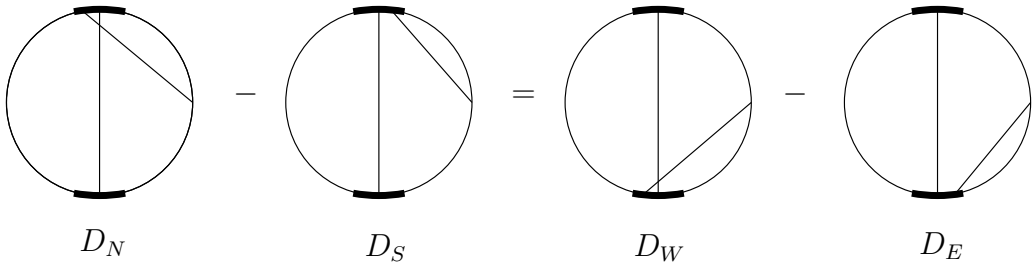


Figure 5.7: Four term relation.

Conversely, Bar-Natan and Kontsevich show

Theorem 5.2.3. *Given a weight function f of degree n and taking values in a field of characteristic zero there is an associated naturally defined Vassiliev invariant $V(f)$ of order n such that*

$$\omega(V(f)) = f$$

and this ω is essentially unique.

To see how a Vassiliev invariant V of order m gives a weight function, W_m, of degree m, we proceed as follows.

Given V of order m and D a chord diagram of degree m, let $W_m(V)$ be the linear functional defined by

$$W_m(V)[D] = V(K_D)$$

where K_D is any embedding of D as a singular knot with m double points. It is shown in Bar-Natan [BN95] that the result is independent of which K_D is chosen and why it vanishes on any 4T relation or 1T relation. The converse result that over $\mathbb{R}$ the degree m weight function can be integrated to give a Vassiliev invariant is much harder, and uses Kontsevich's transcendental formula. For good accounts of the relationship between Vassiliev invariants and the Hopf algebra of chord diagrams see [CDL94a] and [Wil96b].

5.3 Chord diagrams and intersection graphs

Given a chord diagram D we can associate with it a graph $\mathcal{G}(D)$, called its intersection graph. We can then define the operations corresponding to the 4T and 1T relations on chord diagrams in terms of operations in the algebra of all graphs.

The *intersection graph* of a chord diagram is obtained by taking a vertex for each chord of the diagram and an edge between two vertices if the corresponding (straight line) chords intersect.

The *Intersection Graph Conjecture* made in [CDL94c] is that weight functions depend only on the intersection graph of diagrams. The Intersection Graph Conjecture has been verified in [CDL94a] for weight functions up to degree 8. It has also [CDL94b] been shown to be true when the intersection graph is a tree and hence a forest. In other words, if two chord diagrams have isomorphic intersection graphs and this graph is a tree then the diagrams are equivalent modulo the 4T and 1T relations. It is important to recognise that the intersection graph $\mathcal{G}(D)$ is a much rougher concept than a chord diagram. For example each of the distinct chord diagrams in Figure 5.8 has intersection graph isomorphic to the path P_5. The theory of [CDL94b] shows

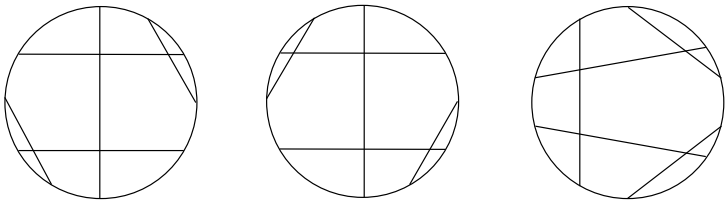


Figure 5.8: Chord diagrams with the same intersection graph.

that since P_5 is a tree, these three diagrams are equal modulo the 4T and 1T relations.

However it has recently been reported in [Wil96a] that a counterexample to the Intersection Graph Conjecture has been constructed by Le Tu using a pair of mutant knots which can be distinguished by a finite order invariant. Despite this, as pointed out in [Wil96a] the results of [CDL94b, CDL94c] are a step towards a better understanding of the Hopf algebra of chord diagrams.

The module generated by all chord diagrams modulo the 4T relation has a natural structure as a Hopf algebra. In the following our notation follows that of Milnor and Moore [MM65]. Let K be a fixed commutative ring. Tensor products are taken over K and $A \oplus B$ will denote the tensor product of the two K-modules A and B. $\hom(A, B)$ denotes the set of morphisms of A into B in the category of K-modules. A *graded* K-module A is a family of K-modules $\{A_n\}$, $n = 1, 2, \ldots$. If A, B are graded K-modules, then a morphism $f : A \to B$ is a family of morphisms $\{f_n\}$ such that $f_n : A_n \to B_n$ is a morphism of K-modules.

$A \oplus B$ is the graded K-module defined by

$$(A \oplus B)_n = \bigoplus_{i+j=n} A_i \oplus B_j.$$

The *dual* A^* is the graded K-module such that

$$(A^*)_n = \hom(A_n, K).$$

The operations on graphs that correspond to multiplication and comultiplication of chord diagrams are as follows. The *product* of two graphs is just their disjoint union. The *coproduct* of a graph G is the sum of all terms of the form $G_J \otimes G_{V \setminus J}$ where $G_J, G_{V \setminus J}$ are the two full subgraphs of G with vertex sets J and $V \setminus J$. This gives the free module generated by all graphs the structure of a Hopf algebra $\mathcal{G}$.

As it stands the mapping defined above has not taken into account the additional structure due to the 4-term relations. In order to obtain a correctly defined homomorphism we replace the target algebra $\mathcal{G}$ by a quotient $\mathcal{G}/\mathcal{S}$ where $\mathcal{S}$ contains $\mathcal{S}_0$, the ideal generated by all 4-term relations. To do this [CDL94c] introduce the *algebra of weighted graphs* and denote it by $\mathcal{W}$. This has one primitive generator in each degree and it yields a series of preinvariants for chord diagrams through the notion of *weighted chromatic invariants*.

Let $\mathcal{A}^c$ be the algebra of chord diagrams modulo the 4-term relation. Let $\mathcal{A}^r$ be the algebra of chord diagrams modulo both the 4T and 1T relations. Clearly, since the relations are homogeneous these algebras inherit gradings from the algebra of chord diagrams.

In [CDL94b] the authors study the subalgebra of chord diagrams $\mathcal{A}^c$ whose intersection graph is a forest. These generate a Hopf subalgebra called the *forest algebra* $\mathcal{F}^c$. They prove the Intersection Graph Conjecture for pairs of diagrams belonging to this forest subalgebra $\mathcal{F}^c$.

In [CDL94c] they show that this forest algebra is isomorphic to the Hopf algebra of weighted graphs.

5.4 Weighted graphs

A *weighted graph* consists of a finite graph G with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set E, together with a weight function $\omega : V \to \mathbb{Z}^+$. We call $\omega(v_i)$ the *weight* of vertex v_i. More generally, if $U \subseteq V$, we define the *weight* of U, $\omega(U)$, to be $\sum_{v \in U} \omega(v)$.

We need to introduce the notion of deletion and contraction in weighted graphs.

If e is an edge of (G, ω) then let (G'_e, ω) denote the graph obtained from G by deleting e and leaving ω unchanged.

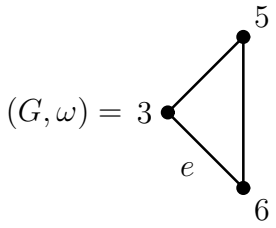
If e is not a loop of (G, ω) then let (G''_e, ω) be the graph obtained from G by contracting e, that is deleting e, identifying its endpoints v, v' into a single vertex v'' and setting

$$\omega(v'') = \omega(v) + \omega(v'),$$

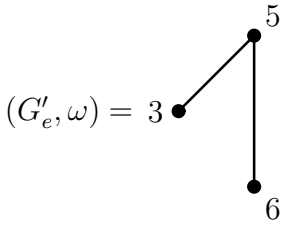
the other vertices of (G''_e, ω) having the same weight as in (G, ω). If e is a loop then we regard (G''_e, ω) as (G'_e, ω).

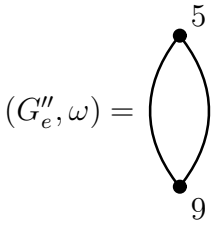
If e is an edge of a simple weighted graph, (G, ω), then (G^c_e, ω) is the graph formed from (G''_e, ω) by replacing every parallel class by a single edge. This is the notion of contraction for weighted graphs used in [CDL94a] where they denote it by G''_e.

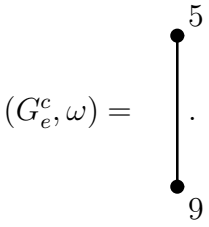
Example. Suppose



then







From here onwards, we will often write just G instead of (G, ω). Now let us examine the forest algebra which is shown in [CDL94c] to be isomorphic to the Hopf algebra of weighted graphs. In the forest algebra $\mathcal{F}^c$ shown, the 4T relation on chord diagrams reduces to the following operation on the

associated intersection graphs. For any forest F, edge e and endpoint u of e, we get the algebraic equation

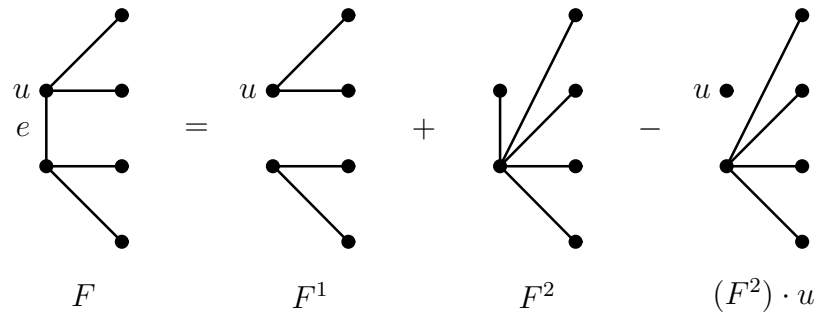


Figure 5.9: The four term relation for forests.

$$F = F^1 + (1 - x_u)F^2$$

shown in Figure 5.9, where x_u is an indeterminate. This is clearly similar to the basic recipe theorem for Tutte-Grothendieck invariants, see [OW79]. Hence it is not surprising that an analogue of the Tutte polynomial can be defined in these algebras. We now proceed to do this.

We first sketch the argument in [CDL94c] which constructs Vassiliev invariants from weighted chromatic invariants on the algebra of weighted graphs. A function f defined on the set $\mathcal{W}_n$ of weighted graphs with total weight n is called a *weighted chromatic invariant of order n* if it satisfies the weighted chromatic relation

$$f(G) - f(G') - f(G^c) = 0. \tag{5.4.1}$$

Given a chord diagram D, form its intersection graph $\mathcal{I}(D)$ which is a weighted graph with *all weights set to* 1.

Theorem 5.4.2 ([CDL94c]: Theorem 3)**.** *If $\mathcal{M}$ denotes the Hopf algebra of chord diagrams modulo the 4T-relation and $\mathcal{W}$ denotes the algebra of weighted graphs then the map $\mathcal{I} : \mathcal{M} \to \mathcal{W}$ is a homomorphism of Hopf algebras.*

A corollary of Theorem 5.4.2 shows that the dual map $\mathcal{I}^* : \mathcal{W}^* \to \mathcal{M}^*$ should give a family of easily calculable weight functions on chord diagrams for each weighted chromatic invariant. As pointed out in [CDL94c], exactly what these are does not seem to be known.

The proof of Theorem 5.4.2 is by showing that if f is a weighted chromatic invariant on intersection graphs then it will satisfy the more complicated condition demanded by the 4T-relation, and this is straightforward combinatorics (graph theory).

5.5 A new polynomial on weighted graphs

We associate with any weighted graph (G, ω) a multivariate polynomial $W_G(\mathbf{x}, y)$ which is defined as follows. Let $y, x_1, x_2, \ldots$ be commuting indeterminates.

Now let $W_G(\mathbf{x}, y)$ be defined recursively by the following rules which are reminiscent of the Tutte-Grothendieck decomposition well-known in combinatorics.

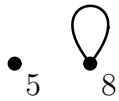
- If G consists of m isolated vertices with weights $a_1, \ldots a_m$ then $W_G(\mathbf{x}, y) = x_{a_1} \ldots x_{a_m}$.
- If G has a loop e then

$$W_G(\mathbf{x}, y) = yW_{G'_e}(\mathbf{x}, y).$$

- If G has an edge e which has distinct endpoints then W satisfies the deletion / contraction rule

$$W_G(\mathbf{x}, y) = W_{G'_e}(\mathbf{x}, y) + W_{G''_e}(\mathbf{x}, y).$$

Thus, as a trivial example, if

$(G, \omega) =$ 

then $W_G(\mathbf{x}, y) = x_5 x_8 y$.

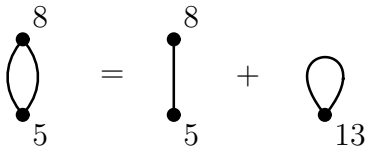
For a slightly less simple example consider the following.

Example. Let

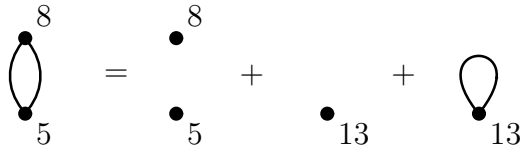
$(G, \omega) =$ 

.

Now



therefore



and so $W_G(\mathbf{x}, y) = x_5 x_8 + x_{13} + x_{13} y$.

We will often write $W(G)$ or just W instead of $W_G(\mathbf{x}, y)$.

Proposition 5.5.1. *The polynomial W is well defined in the sense that the resulting multivariate polynomial is independent of the order in which the edges are deleted / contracted.*

Proof. First note that if we have a weighted graph G and edges e and f then $G/e \setminus f = G \setminus f/e$ so the order in which we contract or delete edges does not affect the graph which we obtain. We prove the result by induction on the number of edges. If G has at most one edge then there is nothing to be proved. So suppose G has at least two edges. Choose any two edges e and f which we will assume are not loops and not in parallel. We will show that computing W in any way that starts with deleting and contracting e leads to the same answer as starting with f. Starting with e gives us $W(G'_e) + W(G''_e)$ which by induction is not dependent on the order in which we delete or contract the rest of the edges but

$$\begin{aligned} W(G'_e) + W(G''_e) &= W((G'_e)'_f) + W((G'_e)''_f) + W((G''_e)'_f) + W((G''_e)''_f) \\ &= W((G'_f)'_e) + W((G'_f)''_e) + W((G''_f)'_e) + W((G''_f)''_e) \\ &= W(G'_f) + W(G''_f) \end{aligned}$$

which is what we get if we delete and contract f first. It is easy to modify this if either e or f or both is a loop, or if they are in parallel. □

The following properties of W are easily verified.

- If G consists of connected components $G_1, \ldots G_k$ then
$$W(G) = \prod_{i=1}^{k} W(G_i).$$
- For any G, each monomial in the polynomial $W_G(\mathbf{x}, y)$ is of the form $x_{a_1} x_{a_2} \cdots x_{a_m} y^t$ where $a_1 + \cdots + a_m$ equals the sum of the weights of G.
- $W_G(\mathbf{x}, y)$ has no term in y if and only if G is a forest.

We first show that W contains as a specialisation the weighted chromatic invariants of [CDL94c] as an evaluation of the form $W_G(\mathbf{x}, 0)$.

Theorem 5.5.2. *Let f be a weighted chromatic invariant with the property that on any graph consisting just of k isolated vertices with weights $a_1, \ldots, a_k$, f takes the value $x_{a_1} \cdots x_{a_k}$, then for any simple weighted graph G, $f(G) = W_G(\mathbf{x}, 0)$.*

Proof. Let G be a graph with no loops. Suppose e_1, e_2 are parallel edges. We claim $W_G(\mathbf{x}, 0) = W_{G'_{e_1}}(\mathbf{x}, 0)$. This follows from the recursive definition of W, $W_G(\mathbf{x}, 0) = W_{G'_{e_1}}(\mathbf{x}, 0) + W_{G''_{e_1}}(\mathbf{x}, 0)$ but $W_{G''_{e_1}}(\mathbf{x}, 0) = 0$ because G''_{e_1} contains a loop e_2. Hence, for any e, $W_{G''_e}(\mathbf{x}, 0) = W_{G^c_e}(\mathbf{x}, 0)$. Now the result follows by induction on the number of edges of G using the recursive definition of W and Equation 5.4.1. □

Theorem 5.5.3. *$W_G(\mathbf{x}, y)$ has a states model representation of the form*

$$W_G(\mathbf{x}, y) = \sum_{A \subseteq E} x_{c_1} x_{c_2} \cdots x_{c_k} (y-1)^{|A| - r(A)}$$

where c_i, $1 \leq i \leq k$ is the total weight of the ith component of the weighted subgraph $(G|A, \omega)$.

Note that in this and some later theorems, the number of components k is not constant and depends on A. Setting $y = 0$ gives a corollary concerning the form of any weighted chromatic invariant.

Corollary 5.5.4. *Every weighted chromatic invariant is of the form*

$$\sum_{A \subseteq E} x_{c_1} \cdots x_{c_k} (-1)^{|A| - r(A)}.$$

Proof of Theorem: We let $c_i(A)$ be the total weight of the ith component of $G|A$.

Let $S(G) = \sum_{A \subseteq E} x_{c_1(A)} x_{c_2(A)} \cdots x_{c_k(A)} (y-1)^{|A| - r(A)}$. We show that $S(G) = W(G)$ by induction on the number of edges of G. If G has no

edges then $S(G) = W(G)$ so suppose e is an edge of G that is not a loop.

$$\begin{aligned}
S(G) &= \sum_{A\subseteq E} x_{c_1(A)}x_{c_2(A)}\cdots x_{c_k(A)}(y-1)^{|A|-r(A)} \\
&= \sum_{A\subseteq E\setminus e} x_{c_1(A)}x_{c_2(A)}\cdots x_{c_k(A)}(y-1)^{|A|-r(A)} \\
&+ \sum_{A: e\in A\subseteq E} x_{c_1(A)}x_{c_2(A)}\cdots x_{c_k(A)}(y-1)^{|A|-r(A)} \\
&= S(G'_e) + \sum_{A\subseteq E\setminus e} x_{c_1(A\cup e)}x_{c_2(A\cup e)}\cdots x_{c_k(A\cup e)}(y-1)^{|A\cup e|-r(A\cup e)} \\
&= S(G'_e) + S(G''_e),
\end{aligned}$$

which by induction is equal to $W(G'_e) + W(G''_e) = W(G)$. If e is a loop then

$$\begin{aligned}
S(G) &= \sum_{A\subseteq E} x_{c_1(A)}x_{c_2(A)}\cdots x_{c_k(A)}(y-1)^{|A|-r(A)} \\
&= \sum_{A\subseteq E\setminus e} x_{c_1(A)}x_{c_2(A)}\cdots x_{c_k(A)}\left((y-1)^{|A|-r(A)} + (y-1)^{|A\cup e|-r(A\cup e)}\right) \\
&= \sum_{A\subseteq E\setminus e} x_{c_1(A)}x_{c_2(A)}\cdots x_{c_k(A)}(y-1)^{|A|-r(A)}y \\
&= yS(G'_e)
\end{aligned}$$

which by induction equals $yW(G'_e) = W(G)$. □As an immediate consequence, we have by putting $y = 1$:

Corollary 5.5.5. *The polynomial $W_G(\mathbf{x}, 1)$ is a homogeneous polynomial in $(x_1, x_2, \ldots)$ and is a weighted sum over all forests of G, namely*

$$W_G(\mathbf{x}, 1) = \sum_{F\in\mathcal{F}(G)} x(F),$$

where if F is a forest with connected components having vertex sets $V_1, \ldots, V_k$, $x(F)$ is the monomial $x_{\omega(V_1)}x_{\omega(V_2)}\ldots x_{\omega(V_k)}$ and $\mathcal{F}(G)$ is the set of all forests of G.

By setting each x_i equal to θ we see that the Tutte polynomial is an evaluation of W.

Corollary 5.5.6. *If $x_i = \theta$ for each i, then the resulting 2-variable polynomial $W_G(x_i = \theta, y)$ is independent of its weights and is given by*

$$W(x_i = \theta, y) = \theta^{k(G)} T(G; 1 + \theta, y),$$

where $k(G)$ is the number of connected components of G.

Note. Here and throughout this paper we slightly abuse notation by letting $W(x_i = \theta, y)$ denote the polynomial (function) obtained by substituting θ for each x_i.

Proof. Putting $x_i = \theta$ for all i in Theorem 5.5.3 gives

$$\begin{aligned} W_G(x_i = \theta, y) &= \sum_{A \subseteq E} \theta^{k(G|A)} (y-1)^{|A| - r(A)} \\ &= \sum_{A \subseteq E} \theta^{|V(G)| - r(A)} (y-1)^{|A| - r(A)} \\ &= \theta^{k(G)} T(G; 1 + \theta, y). \end{aligned}$$

□

This instantly gives a host of specialisations of W, see for example [Wel93].

A second interpretation of W in terms of the Tutte polynomial is contained in the following theorem.

If $\pi = (V_1, \ldots, V_k)$ denotes a partition of the vertex set V it determines subgraphs $G_1, \ldots, G_k$ where G_i denotes the subgraph of G induced by V_i. We say a partition is *connected* if each of $G_1, \ldots, G_k$ is connected. The partition π of V also determines the monomial defined by

$$x(\pi) = x_{\omega(V_1)} \cdots x_{\omega(V_k)}.$$

Then we have:

Theorem 5.5.7. *A states model for W is given by*

$$W_G(\mathbf{x}, y) = \sum_{\pi} x(\pi) T(G_1; 1, y) \cdots T(G_k; 1, y) \tag{5.5.8}$$

where the sum is over all connected partitions π of $V(G)$.

Again, by putting $y = 0$ we obtain a corollary concerning chromatic invariants.

Corollary 5.5.9. *Any weighted chromatic invariant has the form*

$$\sum_{\pi} x(\pi) T(G_1; 1, 0) \ldots T(G_k; 1, 0)$$

where the summation is over all connected partitions of $V(G)$.

Proof of Theorem: Given a set A of edges, we define $\Pi(A)$ to be the partition with ground set V whose blocks correspond to the connected components of $G|A$. We have from Theorem 5.5.3 that

$$W_G(\mathbf{x}, y) = \sum_{A \subseteq E} x_{c_1} x_{c_2} \cdots x_{c_k} (y-1)^{|A| - r(A)}$$

where c_i, $1 \leq i \leq k$ is the total weight of the ith component of the weighted subgraph $(G|A, \omega)$. Now

$$W_G(\mathbf{x}, y) = \sum_{\pi} \sum_{A : \Pi(A) = \pi} x_{\omega(V_1)} x_{\omega(V_2)} \cdots x_{\omega(V_k)} (y-1)^{|A| - r(A)}$$

where π is the partition $(V_1, \ldots, V_k)$. If $\Pi(A) = \pi$ then we can write $A = A_1 \cup \ldots \cup A_k$ where for each i, the edges in A_i have both endpoints in V_i and $G : V_i$ is connected. Hence $r(A) = r(A_1) + \cdots + r(A_k)$. This gives

$$\begin{aligned} W_G(\mathbf{x}, y) &= \sum_{\pi} x_{\omega(V_1)} x_{\omega(V_2)} \cdots x_{\omega(V_k)} \\ &\quad \cdot \sum_{A_1} (y-1)^{|A_1| - r(A_1)} \cdots \sum_{A_k} (y-1)^{|A_k| - r(A_k)} \\ &= \sum_{\pi} x_{\omega(V_1)} x_{\omega(V_2)} \cdots x_{\omega(V_k)} T(G_1; 1, y) \cdots T(G_k; 1, y), \end{aligned}$$

where the final summation is over all connected partitions of $V(G)$. □

Suppose we are given a connected graph G and a total ordering on its edges. Consider a spanning tree T of G. An edge e in $G \setminus T$ is *externally active* with respect to T if it is the largest edge in the unique cycle contained in $T \cup e$. Let $ext(T)$ be the number of externally active edges with respect to T. A theorem of Tutte [Tut54] implies that

$$T(G; 1, y) = \sum_{T} y^{ext(T)}$$

where the summation is over all spanning trees. This implies that the number of spanning trees with a given external activity is a graph invariant and not dependent on the choice of ordering. We extend the definition of external activity to forests in the obvious way: given a graph G and a forest F of G, an edge e of G is *externally active* with respect to F if $F \cup e$ contains a unique cycle and e is the largest edge in that cycle. An edge is externally active with respect to F if both its endpoints are incident with vertices in the same component T of F and e is externally active with respect to the spanning tree T of $G : V(T)$. Given a weighted graph (G, ω) and a forest F with connected components having vertex sets $V_1, \ldots V_k$ the term $x(F)$ is given by

$$x(F) = x_{\omega(V_1)} x_{\omega(V_2)} \cdots x_{\omega(V_k)}.$$

This leads to yet another states model for W over the set of all forests.

Theorem 5.5.10. *For any weighted graph* (G, ω)

$$W_G(\mathbf{x}, y) = \sum_{F \in \mathcal{F}(G)} x(F) y^{ext(F)}$$

where the summation is over all forests of G *and* $ext(F)$ *is the number of edges externally active with respect to* F*.*

Corollary 5.5.11. *Any weighted chromatic invariant can be written in the form*

$$\sum_F x(F)$$

where the summation is over all forests of external activity zero.

Proof of Theorem: For a forest F we let $\Pi(F)$ be the partition of V induced by the connected components of F. Using Theorem 5.5.7 we have

$$W_G(\mathbf{x}, y) = \sum_\pi x(\pi) T(G_1; 1, y) \cdots T(G_k; 1, y)$$

where the summation is over all connected partitions $\pi = (V_1, \ldots, V_k)$ of the vertex set and for each i, G_i is the subgraph induced by V_i. This gives

$$W_G(\mathbf{x}, y) = \sum_\pi x(\pi) \sum_{T_1} y^{ext(T_1)} \cdots \sum_{T_k} y^{ext(T_k)}$$

where for each i the summation over T_i is over all spanning trees of G_i. The union of these trees is just a forest and so

$$\begin{aligned} W_G(\mathbf{x}, y) &= \sum_\pi x(\pi) \sum_{F:\Pi(F)=\pi} y^{ext(F)} \\ &= \sum_{F \in \mathcal{F}(G)} x(F) y^{ext(F)} \end{aligned}$$

where the summation is over all forests of G. □

Proposition 5.5.12. *If G has n vertices each with weight one or weight two and A is the set of vertices of weight two then the coefficient of $x_2^{k+|A|} x_1^{n-2k-|A|}$ in $W_G(\mathbf{x}, 1)$ gives the number of matchings of size k in which no edge has an endpoint in A.*

Proof. Theorem 5.5.10 shows that $W_G(\mathbf{x}, 1) = \sum_F x(F)$ where the summation is over all forests of G. The forests such that $x(F) = x_2^{k+|A|} x_1^{n-2k-|A|}$ are

precisely the matchings with k edges in which no edge has an endpoint in A. □

We now move to a version of Theorem 1.4.6, the Recipe Theorem of [OW79].

Theorem 5.5.13. *Let f be a function on weighted graphs defined recursively by the following conditions.*

1. *For any non-loop edge e,*

$$f(G) = af(G'_e) + bf(G''_e)$$

   where a, b are any two non-zero scalars.

2. *If e is a loop then $f(G) = yf(G'_e)$.*

3. *If G consists of isolated vertices with weights $a_1, \ldots a_m$ then $f(G) = x_{a_1} \cdots x_{a_m}$.*

Then

$$f(G) = a^{|E|-|V|}b^{|V|}W_G\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right).$$

Proof. We prove the result by induction on the number of edges in G. If G consists of isolated vertices with weights $a_1, \ldots a_m$ then $f(G) = x_{a_1} \cdots x_{a_m} = a^{|E|-|V|}b^{|V|}W\left(G; \frac{a\mathbf{x}}{b}, \frac{y}{a}\right)$ so the result is true if G has no edges. Let e be an edge of G and suppose first that e is a loop. Using induction we have that

$$\begin{aligned} f(G) &= yf(G'_e) \\ &= ya^{|E(G'_e)|-|V(G'_e)|}b^{|V(G'_e)|}W_{G'_e}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \\ &= \frac{y}{a}a^{|E(G)|-|V(G)|}b^{|V(G)|}W_{G'_e}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \\ &= a^{|E(G)|-|V(G)|}b^{|V(G)|}W_G\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \end{aligned}$$

so the result is true in this case. Now let e be any non-loop edge of G. Using induction we have that

$$\begin{aligned}
f(G) &= af(G'_e) + bf(G''_e) \\
&= a^{1+|E(G'_e)|-|V(G'_e)|}b^{|V(G'_e)|}W_{G'_e}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \\
&\quad + a^{|E(G''_e)|-|V(G''_e)|}b^{1+|V(G''_e)|}W_{G''_e}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \\
&= a^{|E(G)|-|V(G)|}b^{|V(G)|}W_{G'_e}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \\
&\quad + a^{|E(G)|-|V(G)|}b^{|V(G)|}W_{G''_e}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right) \\
&= a^{|E(G)|-|V(G)|}b^{|V(G)|}W_{G}\left(\frac{a\mathbf{x}}{b}, \frac{y}{a}\right)
\end{aligned}$$

hence the result is true in general. □

5.6 An invariant of ordinary graphs

We now consider the natural invariant of graphs obtained from W by treating an ordinary, unweighted graph G as a weighted graph in which each vertex has weight equal to unity. In this case we write W as $U_G(\mathbf{x}, y)$.

Apart from a certain intrinsic interest as a combinatorial invariant it is also the case that in the original motivating situation (weighted chord diagrams) the weights are initially all one but the recursive definition does not permit remaining in this "unweighted category". First we summarise the basic properties of U which follow from what we know about W.

Proposition 5.6.1. *If G is a graph on n vertices then $U_G(\mathbf{x}, y)$ is a polynomial in $x_1, \ldots, x_n, y$ such that:*

1. *U_G is independent of y if and only if G is a forest.*

2. *Each monomial in U_G is of the form $x_{a_1} \cdots x_{a_m} y^t$ where $\sum a_i = n$.*

3. $$U_G(\mathbf{x}, y) = \sum_{A \subseteq E} x_{n_1} \cdots x_{n_k} (y-1)^{|A|-r(A)}$$

   where $n_1, \ldots n_k$ are the number of vertices of the different components of $G|A$.

We now turn to specific evaluations of U in terms of known combinatorial polynomials. From Corollary 5.5.6 we know that the Tutte polynomial is given by

$$T(G; x, y) = (x-1)^{-k(G)} U_G(x_i = x - 1, y).$$

In particular, the chromatic polynomial $P_G(\lambda)$ is given by

$$P_G(\lambda) = (-1)^{|V|} U_G(x_i = -\lambda, y = 0).$$

Despite its apparent similarity to the Tutte polynomial, U is a much stronger invariant. Any two graphs with the same matroid share a common Tutte polynomial whereas

$$U(P_4) = x_1^4 + 3x_1^2 x_2 + 2x_1 x_3 + x_2^2 + x_4,$$
$$U(St_4) = x_1^4 + 3x_1^2 x_2 + 3x_1 x_3 + x_4$$

where P_4 and St_4 are respectively the path and star with four vertices.

Another specialisation of U gives the *stability polynomial*, $A(G; p)$, of a loopless graph. This was introduced by Farr in [Far93] and is the one-variable polynomial given by

$$A(G; p) = \sum_{U \subseteq V(G) : U \in \mathcal{S}(G)} p^{|U|} (1-p)^{|V(G) \setminus U|},$$

where $\mathcal{S}(G)$ is the set of all stable sets of G.

Theorem 5.6.2. *If G is loopless then $A(G;p)$ is given by*

$$A(G;p) = U_G(x_1 = 1, x_j = -(-p)^j \text{ if } j \geq 2, y = 0).$$

Proof. Theorem 13 in [Far93] shows that

$$A(G;p) = \sum_{A \subseteq E(G)} (-1)^{|A|} p^{f(A)}$$

where $f(A)$ is the number of vertices incident with an edge of A. In the case when all weights are one and $y = 0$, Theorem 5.5.3 reduces to

$$W_G(\mathbf{x}, 0) = \sum_{A \subseteq E} x_{|V(G_1)|} x_{|V(G_2)|} \cdots x_{|V(G_k)|} (-1)^{|A| - r(A)}$$

where G_i, $1 \leq i \leq k$ is the ith component of the weighted subgraph $(G|A, \omega)$. Now $\sum_{i:|V(G_i)>1|} |V(G_i)|$ is equal to $f(A)$ and $\sum_i (|V(G_i)|-1)$ is equal to $r(A)$. Hence

$$(-1)^{r(A)} = \prod_i (-1)^{(|V(G_i)|-1)}.$$

Thus when we set $x_1 = 1$ and $x_j = -(-p)^j$ for $j \geq 2$ we have

$$\begin{aligned} U_G(\mathbf{x}, 0) = W_G(\mathbf{x}, 0) &= \sum_{A \subseteq E} p^{f(A)} (-1)^{r(A)} (-1)^{|A| - r(A)} \\ &= A(G;p). \end{aligned}$$

□

The polymatroid polynomial which is the subject of Chapter 3 can also be obtained from U.

Theorem 5.6.3. *Let G be a loopless graph with no isolated vertices. Then $S(G;u,v)$ is given by*

$$S(G;u,v) = U_G(x_1 = u, x_2 = 1, x_j = v^{j-2} \text{ for } j > 2, y = v^2 + 1).$$

The restriction on isolated vertices is not really important because the polynomial S is unaffected by the addition of isolated vertices.

Proof. The proof is similar to that of the preceding theorem. In the case when all weights are one and $y = v^2 + 1$, Theorem 5.5.3 reduces to

$$W_G(\mathbf{x}, y) = \sum_{A \subseteq E} x_{|V(G_1)|} x_{|V(G_2)|} \cdots x_{|V(G_k)|} v^{2(|A|-r(A))}$$

where G_i, $1 \leq i \leq k$ is the ith component of the weighted subgraph $(G|A, \omega)$. Now the number of components of $G|A$ consisting of just a single vertex is $f(E) - f(A)$. Also

$$r(A) = \sum_{i:|V(G_i)|\geq 2} (|V(G_i)| - 1)$$

and hence

$$2r(A) - f(A) = \sum_{i:|V(G_i)|\geq 2} (|V(G_i)| - 2).$$

Thus, setting $x_1 = u$, $x_2 = 1$, $x_j = v^{j-2}$ for $j \geq 2$ and $y = v^2 + 1$ gives that

$$U_G(\mathbf{x}, y) = W_G(\mathbf{x}, y) = \sum_{A \subseteq E} u^{|f(E)-f(A)|} v^{2r(A)-f(A)} v^{2(|A|-r(A))} = S(G; u, v).$$

□

We now show that the number of cliques of given size appear as coefficients of monomials in U.

Proposition 5.6.4. *If G is simple and has n vertices then the coefficient of $x_k x_1^{n-k} y^{\binom{k}{2}-k+1}$ in $U_G(\mathbf{x}, y)$ gives the number of cliques of size k in G.*

The proposition follows from Theorem 5.5.7 and the following easy lemma.

Lemma 5.6.5. *Suppose G is a simple connected graph on n vertices, then G is a clique if and only if the coefficient of $y^{\binom{n}{2}-n+1}$ in $T(G;1,y)$ is positive.*

Proof. The evaluation of T at $(1,y)$ is given by

$$T(G;1,y) = \sum_A (y-1)^{|A|-n+1}$$

where the summation is over subsets of edges spanning G and so if G is a clique, taking $A = E(G)$ will give a term in $y^{\binom{n}{2}-n+1}$. Conversely if the coefficient of $y^{\binom{n}{2}-n+1}$ in $T(G;1,y)$ is non-zero then G must have $\binom{n}{2}$ edges and hence be a clique. □

Proof of Proposition: Suppose H is a set of vertices of G corresponding to a k-clique. Equation 5.5.8 and Lemma 5.6.5 show that the partition of V into $n-k+1$ blocks, one of which is H and the others are singletons, will lead to a term in $x_k x_1^{n-k} y^{\binom{k}{2}-k+1}$. Such terms can arise only if the partition in the sum in Equation 5.5.8 consists of one block with k vertices and singleton blocks otherwise. Then for some i, $T(G_i;1,y)$ contains a term of the form $y^{\binom{k}{2}-k+1}$ and hence by Lemma 5.6.5, G_i is a clique. □The *matching polynomial* of G is the polynomial $m(G;t)$ given by

$$m(G;t) = \sum_{k\geq 0} m_k t^k,$$

where m_k is the number of matchings of G with k edges. A consequence of Proposition 5.5.12 is that $m(G;t)$ is an evaluation of U.

Proposition 5.6.6. *The matching polynomial is given by*

$$m(G;t) = U_G(x_1 = 1, x_2 = t, x_j = 0 \text{ for } j > 2, y = 1).$$

Proof. Proposition 5.5.12 implies that if G has n vertices then the coefficient of $x_2^k x_1^{n-2k}$ in $U_G(\mathbf{x},1)$ is the number of matchings of size k in G. The result follows from this observation. □

5.7 A symmetric function application

A more interesting and less apparent specialisation of U_G is that it gives the symmetric function generalisation X_G of the chromatic polynomial developed in [Sta95].

For any graph G, X_G is a homogeneous symmetric function in $\mathbf{x} = (x_1, x_2, \ldots)$ of degree $n = |V|$ defined by

$$\begin{aligned} X_G &= X_G(\mathbf{x}) = X_G(x_1, x_2, \ldots) \\ &= \sum_{\kappa} x_{\kappa(v_1)} x_{\kappa(v_2)} \cdots x_{\kappa(v_n)} \end{aligned}$$

where the sum ranges over all proper colourings $\kappa : V \to \mathbb{Z}^+$. Note that the sum is infinite since we allow any proper colouring using positive integers. In [Sta95] Stanley develops X_G and its properties in terms of the standard 'natural' bases for the space of symmetric functions. The basis which is of most interest here is the *power sum basis* $\{p_r\}$ given by $p_0 = 1$ and $p_r(x_1, x_2, \ldots) = \sum_i x_i^r$, for $r \geq 1$, see Macdonald [Mac79].

Theorem 5.7.1. *For any graph G*

$$X_G = (-1)^{|V|} U_G(x_j = -p_j, y = 0).$$

Proof. For $S \subseteq E$, let $\lambda(S)$ be the partition of $n = |V|$, whose parts are equal to the vertex sizes of the connected components of $G|S$. For a partition π with blocks of size $s_1, \ldots s_k$ we let

$$p_\pi = p_{s_1} \cdots p_{s_k},$$

(see [Mac79]). Then from Theorem 2.5 of [Sta95] we know

$$X_G = \sum_{S \subseteq E} (-1)^{|S|} p_{\lambda(S)}.$$

Now use the representation we have given of W_G in Theorem 5.5.3 and noting that the number of parts (or length) of $\lambda(S)$ is exactly $k(G)$, it is easy to see that the substitution $x_j = -p_j$ gives X_G up to $(-1)^{|V|}$. □

Many of the results in [Sta95] now follow from our earlier interpretations of U_G.

In a later paper Stanley [Sta] suggests the following symmetric function generalisation of the Tutte polynomial of a graph. He does this by using the equivalent representation in terms of the *bad colouring* polynomial of [Wel93] which gives T as the generating function for the number of 'bad' or monochromatic edges over *all* colourings $\kappa : V(G) \to \{1, 2, \ldots\}$.

Definition. Let $\mathbf{x} = (x_1, x_2, \ldots)$ and t be indeterminates and define

$$X_G(\mathbf{x}, t) = \sum_{\kappa: V \to \mathbb{Z}^+} (1+t)^{b(\kappa)} x_{\kappa(v_1)} x_{\kappa(v_2)} \cdots x_{\kappa(v_n)},$$

where $b(\kappa)$ denotes the number of bad edges in the colouring κ and the sum is over *all* colourings.

Note that the sum is infinite since we allow all possible colourings using positive integers. If $f(1^n)$ denotes the substitution $x_1 = x_2 = \cdots = x_n = 1$, $x_{n+1} = x_{n+2} = \cdots = 0$ then

$$X_G(1^n, t) = n^{k(G)} t^{r(G)} T\left(G; \frac{t+n}{t}, t+1\right)$$

and

$$X_G(\mathbf{x}, -1) = X_G(\mathbf{x}).$$

Theorem 5.7.2. *For any G, U_G and $X_G(\mathbf{x}, t)$ determine each other. In particular X_G is easily obtained from U_G by the substitution*

$$X_G(\mathbf{x}, t) = t^{|V|} U_G\left(x_j = \frac{p_j}{t}, y = t+1\right).$$

Conversely, if we can expand X_G in terms of the power basis of symmetric functions then we can recover U_G.

Proof. A result attributed in [Sta] to T. Chow gives the representation

$$X_G(\mathbf{x}, t) = \sum_{S \subseteq E} t^{|S|} p_{\lambda(S)}$$

where we are using the same notation as in the previous theorem. Now put $x_j = p_j/t$ and $y = t + 1$ in the representation for U_G given in Theorem 5.5.3 and the result follows. □

One consequence of Theorem 5.7.2 and our original definition of W and which does not seem obvious from their definitions in [Sta] is:

Corollary 5.7.3. *The function $X_G(\mathbf{x}, t)$ and hence the symmetric chromatic polynomial X_G have recursive delete / contract derivations analogous to those of the normal chromatic polynomial.*

5.8 Complexity issues

Consider now the problem of computing W. This is clearly going to be a hard problem for general G and $\mathbf{x}$ because the Tutte polynomial is a specialisation of W and this is shown in [JVW90] to be $\#P$-hard to compute at most points. In a sense it appears that W should be no harder to compute than the Tutte polynomial as both have a similar recursive formula [OW79]. Here however, we show that unlike the Tutte polynomial, this polynomial is hard to compute even for the special cases where the input graph is a tree or a complete graph.

First we clarify the problems.

Problem 5.8.1 W-COEFFICIENT

Input Weighted graph G, finite ordered (multi)-set $P = \{p_1, \ldots p_k\}$ of positive integers and a positive integer q.

Output Coefficient of $x_{p_1}, \ldots, x_{p_k} y^q$ in $W_G(\mathbf{x}, y)$.

Problem 5.8.2 W-EVALUATION

Input Weighted graph G, finite set $S = \{(i, a_i) : i \in I\}$ where $I \subseteq \mathbb{Z}$ and each a_i is an integer together with an integer y_0.

Output The value which W takes on substituting

$$x_i = \begin{cases} a_i & \text{if } i \in I; \\ 0 & \text{if } i \notin I. \end{cases}$$

Note. We have made the slightly artificial restriction that the a_i be integral to avoid unnecessary complications. As in [JVW90] we could have allowed the a_i to take values in any algebraic field but since we are going to prove hardness results for most graphs and most integer points it does not seem worthwhile.

Theorem 5.8.3. *Computing* W-COEFFICIENT *is* $\#P$*-hard even if* G *is restricted to being a tree.*

To prove the theorem we need to consider the following four problems, the first of which is a counting version of one of Karp's original twelve NP-complete problems. Note that in the following we take $\mathbb{Z}^+$ to be the set of strictly positive integers so that we do not allow elements to have size 0.

Problem 5.8.4 #PARTITION

Input Finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

Output The number of subsets $A' \subseteq A$ with $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$.

Problem 5.8.5 #1/2-PARTITION

Input Finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

Output The number of subsets $A' \subseteq A$ with $|A'| = |A|/2$ and $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$.

Problem 5.8.6 #PAIRED-PARTITION

Input Finite set $A = \{a_1, \ldots a_{2n}\}$ and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$.

Output The number of subsets $A' \subseteq A$ such that A' contains precisely one of a_i and a_{2n-i} for each i with $1 \leq i \leq n$ and $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$.

Problem 5.8.7 #DISTINCT-PARTITION

Input Finite set A and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $s(a) \neq s(a')$ if $a \neq a'$.

Output The number of subsets $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$.

We need the following lemma.

Lemma 5.8.8. *Each of problems 5.8.4–5.8.7 is #P-complete.*

Proof. We show #PARTITION α_T #1/2PARTITION α_T #PAIRED-PARTITION α_T #DISTINCT-PARTITION. As far as we know there is nothing in the literature showing explicitly that #PARTITION is #P-hard. The reductions in [GJ79, Kar72] showing that PARTITION is NP-hard do not immediately give proofs of #P-hardness because they do not seem to preserve the number of solutions. However using the following two problems it is possible to obtain a reduction.

Problem 5.8.9 #3-COL

Input A graph G.

Output The number of proper 3-colourings of the vertices of G.

Problem 5.8.10 #EXACT COVER

Input Finite set A and a collection C of subsets of A.

Output The number of subcollections $C' \subseteq C$ such that every element of A is contained in precisely one member of C'.

The problem #3-COL is shown to be $\#P$-hard in [Lin86] and a reduction is given from the decision problem 3-COL to the decision problem EXACT COVER in [Kar72]. It is easy to check that this reduction preserves the number of solutions and so #EXACT COVER is $\#P$-hard. There is a reduction to PARTITION from a problem very similar to EXACT COVER, namely 3-DIMENSIONAL MATCHING in [GJ79] and this can be easily modified to give a reduction from EXACT COVER which doubles the number of solutions. Hence #PARTITION is $\#P$-hard.

Next we show #PARTITION α_T #1/2PARTITION by giving an algorithm which constitutes a Turing reduction from #PARTITION to #1/2PARTITION. We assume we are given an instance (A, s) of #PARTITION and an oracle for #1/2PARTITION. The command 1/2PARTITION(A; $s(a_1)$, ..., $s(a_{2n})$) calls the oracle with the instance (A, s) of #1/2PARTITION where $A = \{a_1, \ldots a_{2n}\}$. This algorithm is shown as Algorithm 18.

To show #1/2PARTITION reduces to #PAIRED-PARTITION we suppose we have an instance (A, s) of #1/2PARTITION. Since the number of solutions to #1/2PARTITION is zero if $|A|$ is odd we may assume that $|A|$ is even. Let $A = \{a_1, \ldots a_{2n}\}$ and let $M = \sum_{a \in A} s(a)$. We construct an instance (A', s') of #PAIRED-PARTITION by setting $A' = \{a_1, \ldots a_{4n}\}$ and defining s' by

$$s'(a_i) = \begin{cases} s(a_i) & \text{if } 1 \leq i \leq 2n, \\ M & \text{otherwise.} \end{cases}$$

It is clear that the number of solutions to the original instance of #1/2PARTITION equals the number of solutions to this instance of #PAIRED-PARTITION.

Finally we show #PAIRED-PARTITION α_T #DISTINCT-PARTITION. Suppose we have an instance (A, s) of #PAIRED-PARTITION where $A = \{a_1, \ldots, a_{2n}\}$. Any pair (a_i, a_{2n-i}) with $s(a_i) = s(a_{2n-i})$ is essentially irrelevant because deleting such a pair of elements just halves the number of solutions to #PAIRED-PARTITION, so assume that there are no pairs of this form. Let $M = \sum_{a \in A} s(a)$. We construct an instance (A, s') of #DISTINCT-PARTITION where s' is given by

$$s'(a_i) = \begin{cases} s(a_i) + M^i & \text{if } 1 \leq i \leq n, \\ s(a_i) + M^{2n-i} & \text{otherwise.} \end{cases}$$

This means that if $i \neq j$ then $s'(a_i) \neq s'(a_j)$ and again it is easy to see that this instance of #DISTINCT-PARTITION has the same number of solutions as the original instance of #PAIRED-PARTITION. □

We can now prove the theorem.

Proof of Theorem: We suppose we are given an instance (A, s) of #DISTINCT-PARTITION where $A = \{a_1, \ldots a_n\}$. Let $M = \sum_{a \in A} s(a)$ and consider the weighted tree T, shown in Figure 5.10 with vertices $v_1, \ldots, v_n$,

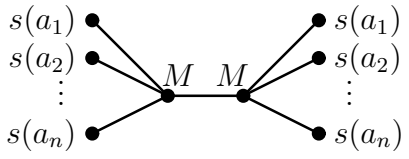


Figure 5.10: The weighted tree T.

$v'_1, \ldots, v'_n, v, v'$ where $\omega(v_i) = \omega(v'_i) = s(a_i)$ and $\omega(v) = \omega(v') = M$. For

Algorithm 18 Turing reduction from PARTITION to #1/2PARTITION

Input: $A = \{a_1, \ldots, a_n\}$, $s(a_i), 1 \leq i \leq n$

$M \leftarrow 2 \sum_{a \in A} s(a)$

$Count \leftarrow 0$

for $i = 1$ to n **do**

$s'(a_i) = s(a_i)$

end for

for $i = 1$ to $\lfloor n/2 \rfloor$ **do**

$A' \leftarrow \{a_1, \ldots, a_{2n-2i+2}\}$

for $j = n + 1$ to $2n - 2i + 1$ **do**

$s'(a_j) \leftarrow M$

end for

$s'(a_{2n-2i+2}) \leftarrow (n - 2i + 1)M$

$Count \leftarrow Count + 1/2\text{PARTITION}(s(a_1), \ldots s(a_{2n-2i+2}))$

end for

Output: $Count$

each i, T has an edge joining v_i with v, and joining v'_i with v'. Finally there is an edge between v and v'. Theorem 5.5.7 shows that the coefficient of $x_{s(a_1)} \cdots x_{s(a_n)} x^2_{\lfloor 3M/2 \rfloor}$ in $W(T)$ is equal to the number of partitions of $V(T)$ into $n+2$ blocks such that two blocks have total weight $\lfloor 3M/2 \rfloor$ and the other have weights $\{s(a_1), \ldots s(a_n)\}$. The number of such partitions is equal to the number of sets $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$. □

Theorem 5.8.11. *Computing* W-COEFFICIENT *is* $\#P$*-hard if* G *is restricted to being a complete graph and in the notation of Problem 5.8.1,*

$$P = \{\lfloor \sum_{v \in V} \omega(v)/2 \rfloor, \lfloor \sum_{v \in V} \lfloor \omega(v)/2 \rfloor \}$$

and $q = 0$.

Proof. We use a reduction from #1/2PARTITION. Suppose (A, s) is an instance of #1/2PARTITION where $A = \{a_1, \ldots, a_{2n}\}$. Let $M = \sum_{a \in A} s(a)$. Now let G be a complete weighted graph with vertices $\{v_1, \ldots, v_{2n}\}$ and $\omega(v_i) = s(a_i) + M$. Clearly $\sum_{v \in V} \omega(v) = (2n+1)M$. Adding M to $s(a_i)$ to give $\omega(v_i)$ ensures that if we partition $V(G)$ into two sets of equal weight then the two sets must each contain n vertices. Then Theorem 5.5.7 shows that the coefficient of $x_{\lfloor (2n+1)M/2 \rfloor} x_{\lfloor (2n+1)M/2 \rfloor}$ is equal to

$$(T(K_n; 1, 0))^2 \alpha(A, s)/2$$

where $\alpha(A, s)$ is the number of solutions to the instance of #1/2PARTITION. The factor of a half is because the #1/2PARTITION problem counts twice each partition of A into parts with the same total size. A recurrence relation to evaluate $T(K_n; 1, 0)$ is given in [Ann94] and it is easy to see that this leads to a polynomial time algorithm for evaluating $T(K_n; 1, 0)$. □

We now turn to the problem W-EVALUATION.

Theorem 5.8.12. *Computing* W-EVALUATION *is* $\#P$*-hard even if* G *is a star.*

Proof. The proof is by a reduction from PARTITION. Let (A, s) be an instance of PARTITION. Let $M = \sum_{a \in A} s(a)$. Now let G be the weighted star shown in Figure 5.11 with $|A|+1$ vertices where the central vertex has weight M and the other vertices have weights corresponding to sizes of distinct elements of A. The evaluation of W with $x_{\lfloor 3M/2 \rfloor} = 1$ and $x_{s(a)} = 1$ for all $a \in A$ but $x_j = 0$ otherwise, gives the number of partitions of A into two parts with equal size. □

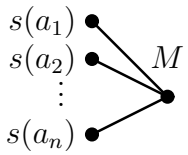


Figure 5.11: The graph G.

It seems worthwhile to note the following obvious corollary.

Corollary 5.8.13. *Computing* W-EVALUATION *is* $\#P$*-hard for trees.*

Theorem 5.8.14. *Computing* W-EVALUATION *is* $\#P$*-hard even if* G *is a complete graph and in the notation of Problem 5.8.2,* $S = \{(\lfloor \sum_{v \in V} \omega(v)/2 \rfloor, 1)\}$ *and* $y_0 = 0$.

Proof. Let $M = \sum_{v \in V} \omega(v)$. The theorem follows from the corresponding theorem concerning W-COEFFICIENT. Theorem 5.8.11 shows that it is $\#P$-hard to compute the coefficient of $x^2_{\lfloor M/2 \rfloor}$, where $M = \sum_{v \in V} \omega(v)$. If $M \geq 4$ then the evaluation of W with $x_{\lfloor M/2 \rfloor} = 1$ and $x_j = 0$ otherwise, is equal to the coefficient of $x^2_{\lfloor M/2 \rfloor}$. □

This highlights the fact that it is not really the graph, but the weights which make W-EVALUATION hard to compute. Of more interest is to consider U and its complexity. Clearly computing U is hard in general but one problem which seems non-trivial to solve is the following:

Problem 5.8.15. *Is computing U_G hard if G is a tree or is there a polynomial algorithm ?*

The complexity of the polynomial $U_G(\mathbf{x}, 0)$ when G is restricted to being an intersection graph of a chord diagram is of particular interest because these evaluations correspond to weight functions on chord diagrams. In [GJMP80], computing the chromatic number of such a graph, commonly known as a circle graph, is shown to be NP-hard. This implies the following theorem:

Theorem 5.8.16. *Computing $U_G(\mathbf{x}, 0)$ is NP-hard even if G is restricted to being the intersection graph of a chord diagram.*

Proof. Equation 5.4.1 shows that the chromatic polynomial can be computed from $U_G(\mathbf{x}, 0)$ and this easily gives the chromatic number. □

The reader will notice that, unlike our earlier results, we are only claiming in Theorem 5.8.16 that U_G is NP-hard. We strongly believe that it is possible to replace this by $\#P$-hard, however the only proof we can see is by following through the reductions of [GJMP80] and [Kar75] and showing that the reductions are at least weakly parsimonious. To do this rigorously would take a great deal of time and space and at this stage hardly seems justified.

5.9 Conclusion

Although the functions W_G, U_G are formally very similar there does seem to be a significant difference in their computational complexity. For example,

while W is hard even for the star or the complete graph, it is easy to see that

$$U(St_n; \mathbf{x}, y) = \sum_{k=0}^{n-1} \binom{n-1}{k} x_{k+1} x_1^{n-k-1}$$

if St_n denotes the star with n vertices. Similarly for the path P_n, if U_n denotes $U(P_n)$ then it is easy to prove

$$U_n = x_1 U_{n-1} + x_2 U_{n-2} + \cdots + x_{n-1} U_1 + x_n$$

so it too is easy to compute and in fact has a 'nice' generating function (see a corresponding formula for $X(P_n)$ in [Sta95, Proposition 5.3].

Finally, for the complete graph K_n, [Sta] gives a generating function completely analogous to that of the Tutte polynomial, namely

$$\sum_{d \geq 0} X_{K_d}(\mathbf{x}, t) \frac{u^d}{d!} = \exp\left(\sum_{m \geq 0} C_m(t) p_m(\mathbf{x}) \frac{u^m}{m!} \right)$$

where

$$C_m(t) = \sum_{i=m-1}^{\binom{m}{2}} c_{mi} t^i$$

and where c_{mi} is the number of connected simple graphs with m vertices and i edges. From our correspondence given by Theorem 5.7.2, we can read off the corresponding formula

$$\sum_{d \geq 0} U_{K_d}(\mathbf{x}, y) \frac{u^d}{d!} = \exp\left((y-1)^{-(m-1)} \sum_{m \geq 0} \frac{C_m(y-1) x_m u^m}{m!} \right).$$

It now follows in exactly the same way as in the proof of the corresponding result for ordinary Tutte polynomials by Annan [Ann94] that there exists a polynomial time algorithm for computing $U(K_n; \mathbf{x}, y)$. This contrasts with Theorem 5.8.14 and together these results suggest that computationally U is closer to T than to W. However although it is easy to find non-isomorphic

graphs with the same Tutte polynomial we know of no pair of non-isomorphic graphs which have the same polynomial U, although we are fairly sure they exist. A more intriguing question is whether there exist non-isomorphic trees with the same U.

Conclusion

We close by restating the main open problems from this thesis.

In [KO95], the authors pose problems concerning the complexity of the language of partitionable simplicial complexes. We have answered one of their questions by showing that the language of partitionable complexes is in NP but the following problem remains open.

Open Question 1. *Is the language of partitionable simplicial complexes NP-complete and if so is it NP-complete for fixed dimension d ?*

Even the case $d = 3$ does not seem to be easy to solve. Unlike in higher dimensions, it seems quite possible that there is a polynomial time algorithm for $d = 3$.

Chapter 4 poses many open problems, since the containment relations between the various classes are unclear.

Open Question 2. *What are the precise relationships between the various classes of graphs ?*

The most intriguing question on which we have spent the most time concerns the size of $\mathcal{S}$.

Open Question 3. *Does $\mathcal{S}$ contain all graphs ?*

We believe not and suspect that K_6 is probably not in $\mathcal{S}$. It is easy to see that if any member of the Petersen family is contained in $\mathcal{S}$ then they all are.

Open Question 4. *Is any member of the Petersen family in $\mathcal{S}$?*

The main problems from the last chapter are concerned with whether $U(G)$ determines G. We think that it is likely that there are two non-isomorphic graphs G_1, G_2 such that $U(G_1) = U(G_2)$ but have been unable to find such a pair.

Open Question 5. *Does there exist a pair of non-isomorphic graphs G_1, G_2 such that $U(G_1) = U(G_2)$?*

We have been unable to decide this question in the special case of trees.

Open Question 6. *Does there exist a pair of non-isomorphic trees T_1, T_2 such that $U(T_1) = U(T_2)$.*

This is obviously much less likely to be true than the previous question but we cannot disprove it.

Bibliography

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading Mass., 1974.

[Ann94] J. D. Annan. *Complexity of Counting Problems*. PhD thesis, Oxford University, 1994.

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989.

[BN95] D. Bar-Natan. On the Vassiliev knot invariants. *Topology*, 34:423–472, 1995.

[Bod93] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1–2):1–21, 1993.

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

[Bry82] T. H. Brylawski. The Tutte polynomial. In A. Barlotti, editor, *Matroid Theory and its Applications*, pages 125–275, Naples, September 1982. CIME, Liguori editore.

[BO92] T. H. Brylawski and J. G. Oxley. The Tutte polynomial and its applications. In N. White, editor, *Matroid Applications*, number 40 in Encyclopedia of Mathematics, chapter 6, pages 123–225. Cambridge University Press, Cambridge, 1992.

[CDL94a] S. V. Chmutov, S. V. Duzhin, and S. K. Lando. Vassiliev knot invariants: I. Introduction. *Advances in Soviet Mathematics*, 21:117–126, 1994.

[CDL94b] S. V. Chmutov, S. V. Duzhin, and S. K. Lando. Vassiliev knot invariants: II. Intersection graph for trees. *Advances in Soviet Mathematics*, 21:127–134, 1994.

[CDL94c] S. V. Chmutov, S. V. Duzhin, and S. K. Lando. Vassiliev knot invariants: III. Forest algebra and weighted graphs. *Advances in Soviet Mathematics*, 21:135–145, 1994.

[Far93] G. E. Farr. A correlation inequality involving stable sets and chromatic polynomials. *Journal of Combinatorial Theory Series B*, 58(1):14–21, 1993.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, New York, 1979.

[GJMP80] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of colouring circular arcs and chords.

SIAM Journal on Algebraic and Discrete Methods, 1(2):216–227, 1980.

[GS96] I. M. Gessel and B. E. Sagan. The Tutte polynomial of a graph, depth-first search, and simplicial complex partitions. *Electronic Journal of Combinatorics*, 3(2), 1996. The Foata Festschrift.

[GZ83] C. Greene and T. Zaslavsky. On the interpretations of Whitney numbers through arrangements of hyperplanes, zonotopes, non-radon partitions, and orientations of graphs. *Transactions of the American Mathematical Society*, 280(1):97–126, 1983.

[GLS80] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Number 2 in Algorithms and Combinatorics. Springer-Verlag, Berlin, 1980.

[JVW90] F. Jaeger, D. L. Vertigan, and D. J. A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 108(1):35–53, 1990.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In J. W. Thatcher and R. E. Miller, editors, *Complexity of computer computations*, pages 85–103, New York, March 1972. Plenum Press.

[Kar75] R. M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[KO95] P. Kleinschmidt and S. Onn. Oriented matroid polytopes and polyhedral fans are signable. In *Proceedings of Fourth Confer-*

ence on Integer Programming and Combinatorial Optimisation (IPCO IV), number 920 in Lecture Notes in Computer Science, pages 198–211. Springer, 1995.

[Kon93] M. Kontsevich. Vassiliev's knot invariants. *Advances in Soviet Mathematics*, 16(2):137–150, 1993.

[Lin86] N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic and Discrete Methods*, 7:331–335, 1986.

[Mac79] I. G. MacDonald. *Symmetric functions and Hall polynomials*. Oxford Mathematical Monographs. Oxford University Press, New York, 1979.

[MSS+95] D. B. Massey, R. Simon, R. P. Stanley, D. L. Vertigan, D. J. A. Welsh, and G. M. Ziegler. Lê numbers of arrangements and matroid identities. *Journal of Combinatorial Theory Series B*, 70:119–133, 1997.

[MM65] J. W. Milnor and J. C. Moore. On the structure of Hopf algebras. *Annals of Mathematics*, 81(1–3):211–264, 1965.

[Nob96] S. D. Noble. Recognising a partitionable simplicial complex is in NP. *Discrete Mathematics*, 152(1–3):303–305, May 1996.

[Nob97/98] S. D. Noble. Evaluating the Tutte polynomial for graphs of bounded tree-width. To appear in Combinatorics, Probability and Computing.

[Oxl92] J. G. Oxley. *Matroid Theory*. Number 3 in Oxford Graduate Texts in Mathematics. Oxford University Press, Oxford, 1992.

[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, London, 1979.

[OW92] J. G. Oxley and D. J. A. Welsh. Tutte polynomials computable in polynomial time. *Discrete Mathematics*, 109(1–3):185–192, 1992.

[OW93] J. G. Oxley and G. P. Whittle. Tutte invariants for 2-polymatroids. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, number 147 in Contemporary Mathematics, pages 9–19. AMS, 1993.

[Rei48] K. Reidemeister. *Knotentheorie*. Chelsea, New York, 1948.

[RST93] N. Robertson, P. D. Seymour, and R. Thomas. A survey of linkless embeddings. In N. Robertson and P. D. Seymour, editors, *Graph Structure Theory*, number 147 in Contemporary Mathematics, pages 125–136. AMS, 1993.

[SW93] W. Schwärzler and D. J. A. Welsh. Knots, matroids and the Ising model. *Mathematical Proceedings of the Cambridge Philosophical Society*, 113:107–139, 1993.

[Sta] R. P. Stanley. Graph colourings and related symmetric functions: ideas and applications. To appear in Discrete Mathematics.

[Sta73] R. P. Stanley. Acyclic orientations of graphs. *Discrete Mathematics*, 5:171–178, 1973.

[Sta95] R. P. Stanley. A symmetric function generalisation of the chromatic polynomial of a graph. *Advances in Mathematics*, 111(1):166–194, March 1995.

[Tru89] K. Truemper. On the delta-wye reduction for planar graphs. *Journal of Graph Theory*, 13(2):141–148, 1989.

[Tut54] W. T. Tutte. A contribution to the theory of chromatic polynomials. *Canadian Journal of Mathematics*, 6:80–91, 1954.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[VW92] D. L. Vertigan and D. J. A. Welsh. The computational complexity of the Tutte plane: the bipartite case. *Combinatorics Probability and Computing*, 1(2):181–187, 1992.

[Wel76] D. J. A. Welsh. *Matroid Theory*. Number 8 in London Mathematical Society Monographs. Academic Press, London, 1976.

[Wel93] D. J. A. Welsh. *Complexity : Knots, Colourings, and Counting*. Number 186 in London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, 1993.

[Wil96a] S. Willerton. Review of Chmutov, Duzhin and Lando, Vassiliev knot invariants: I–III. Mathematical Reviews 96i57002, 1996.

[Wil96b] S. Willerton. Vassiliev invariants and the Hopf algebra of chord diagrams. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(1):55–65, 1996.

[Wil72] R. J. Wilson. *Introduction to Graph Theory.* Academic Press, New York, 1972.