Exponential-Time Algorithms and Complexity of NP-Hard Graph Problems

Nina Taslaman

IT University of Copenhagen

Section of Theoretical Computer Science

PhD Dissertation

Abstract

NP-hard problems are deemed highly unlikely to be solvable in polynomial time. Still, one can often find algorithms that are substantially faster than brute force solutions. This thesis concerns such algorithms for problems from graph theory; techniques for constructing and improving this type of algorithms, as well as investigations into how far such improvements can get under reasonable assumptions.

The first part is concerned with detection of cycles in graphs, especially parameterized generalizations of Hamiltonian cycles. A remarkably simple Monte Carlo algorithm is presented for the problem of finding a cycle through a specified subset of vertices or edges. The running time is exponential only in the number k of specified elements, with a dependence of 2^k. Previously, the best upper bound for this problem was doubly exponential in k^{10}. The algorithm never reports a false positive, and with high probability any found solution is shortest possible. Moreover, the algorithm can be used to find a cycle of given parity through the specified elements.

The second part concerns the hardness of problems encoded as evaluations of the Tutte polynomial at some fixed point in the rational plane, referred to as the Tutte plane in this context. Under the Exponential Time Hypothesis, which claims a certain exponential-time requirement for solving the problem *3-Sat*, superpolynomial lower bounds are given for problems restricted to simple or planar graphs. The restriction to simple graphs has been studied previously and lower bounds exist for most of the Tutte plane; the contribution here is a first result for points on the line corresponding to computation of all-terminal network reliability. For these points, a novel reduction provides a lower bound that is asymptotically tight up to a polylogarithmic factor in the exponent. For planar graphs, lower bounds are found by examining and combining existing reductions. An asymptotically tight bound is found for points corresponding to evaluation of the 3-state Potts model partition function; for remaining points the obtained lower bounds are significantly further from the best known upper bound. These are the first results of this type for planar graphs, to the best of the knowledge of the author. Along one particular line in the Tutte plane this is also the first such result for general graphs.

Acknowledgements

This piece of work would not have been possible without the guidance of my advisor, Thore Husfeldt, my co-authors Andreas Björklund and Holger Dell, the support of my family and friends, and Olof. For helpful comments and discussions I am also very grateful to Danny Hermelin, Magnus Wahlström, Daniel Lokshtanov, and Konstantin Kutzkov. Finally I wish to thank my assessment committee, Markus Bläser, Fedor V. Fomin, and Rasmus Pagh, for their invested time and feedback.

Contents

Chapter 1

Introduction

Some problems are harder than others, so it seems. In the field of computer science, the most famous notion of a hard problem is that of an *NP-complete* problem, such as the classical *Hamiltonian Cycle* problem, which asks for the existence of a closed walk with no self-intersections that connects all vertices in a given input graph. NP-complete problems are considered hard not because we do not know how to solve them, but because no one has yet found a *polynomial-time algorithm* to solve any such problem, i.e. an algorithm that solves the problem within a number of steps that depends polynomially on the size of the input. It is a generally believed (but famously unproved) hypothesis that there can be no polynomial-time algorithm for any NP-complete problem. This hypothesis is also known as P $\neq$ NP.

It is a common misconception that one cannot solve NP-complete problems significantly faster than by exhaustive search, trying all valid possibilities for a solution. That this is not true has been known since the very introduction of the concept of NP-completeness in the 1970's; for example, the running time of exhaustive search for the *Hamiltonian Cycle* problem depends factorially on the number n of vertices, but already in the 1960's Bellman [5] discovered an algorithm which improved this to single exponential in n—a significant improvement indeed. Some decades ago, however, such improved exponential-time algorithms were often of little interest in practice, as the limit of computational capacity still put up a barrier for relevant input sizes. Clearly, the situation has changed since then, and during the last decade or so a renewed interest in improved exponential-time algorithms for NP-hard problems has arisen.

This thesis concerns possibilities and limits of improved algorithms for NP-complete graph problems. In other words, we investigate the complexity of such problems from an exponential-time perspective.

1.1 Basics

Our notation is standard, but for clarity and completeness this section gives a brief review of elementary concepts and terminology used in the thesis. Readers familiar with computational complexity and graph theory can skip directly to Section 1.2.

1.1.1 Problems, algorithms, and complexity

Consider the school-book exercise of finding an assignment to the boolean variables $\{x_1, x_2, x_3, x_4\}$ such that the 3-CNF formula[1]

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_3} \vee x_4)$$

is satisfied. This exercise is an *instance* of the following *computational problem*:

Input:	A 3-CNF formula φ.
Task:	Find a satisfying assignment to the variables of φ.

A *decision problem* is a computational problem where the solution is a yes-or-no answer to some question regarding the input. An example is the famous *3-Sat* problem of determining whether an input 3-CNF formula has any satisfying assignment. An instance of a decision problem for which the answer is 'yes' is a *yes-instance*, and an instance that is not a yes-instance is a *no-instance*. A *counting problem* is a computational problem where the solution gives the size of some set related to the input. An example is the problem of counting the number of all satisfying assignments of an input 3-CNF formula; this is called the *#3-Sat* problem, as it is the 'counting version' of *3-Sat*.

A list of computational problems can be found in Appendix A.

Algorithms. An *algorithm* for a computational problem is a method, given as a finite set of instructions, for solving arbitrary instances of the problem. For example, an algorithm for the *3-Sat* problem takes as input a specific 3-CNF formula φ, and within finite time outputs 'yes' or 'no' according to the satisfiability of φ.

The *running time* of an algorithm will in this thesis always refer to the *worst case* running time, i.e. the maximum number $T(n)$ of computational steps the algorithm will perform for *any* input of size n. If $T(n)$ is polynomially bounded, i.e. if $T(n) \in n^{O(1)}$, the algorithm is referred to as a *polynomial-time algorithm*. Otherwise the running time is *superpolynomial*, and the algorithm is either *exponential-time* with $T(n) \in \exp(n^{O(1)})$, or *subexponential-time* with $T(n) \in \exp(o(n))$.

A *Monte Carlo algorithm* uses a random choice at some step, and has a small probability $p < 1/2$ of producing an incorrect solution. The running time of a Monte Carlo algorithm is deterministic, i.e. independent of the random choice. We will only consider Monte Carlo algorithms with *one-sided error*, usually in the sense that a false positive is never reported. Algorithms that do not employ random choices are *deterministic*.

[1]A 3-CNF formula is a boolean formula in conjunctive normal form, i.e. a conjunction of disjunction-clauses, with 3 literals per clause.

Complexity. The *complexity* of a computational problem is the minimum running time achievable for any deterministic algorithm that solves the problem. The running time of any particular algorithm provides an upper bound on the complexity. A problem is *polynomial-time solvable* if there is a deterministic polynomial-time algorithm for it. The class of all such problems is called FP. The set of decision problems in FP is called P. A decision problem is in RP if it can be solved by a polynomial-time Monte Carlo algorithm of one-sided error.

NP and #P. The complexity class NP consists of all decision problems for which every yes-instance has a *witness* that can be *verified* by a deterministic polynomial-time algorithm. For example, the problem *3-Sat* is in NP: any satisfying assignment to the input 3-CNF formula would constitute such a witness, as it can be checked by a routine polynomial-time calculation that it satisfies the input formula, and hence that we are dealing with a *yes*-instance. The class of counting versions of problems in NP, such as *#3-Sat*, is called #P.

Conjectures. While it is clear that P $\subseteq$ NP, the question of the reverse inclusion is still open and generally believed false. One reason for this is that despite extensive efforts, no one has yet found a polynomial-time algorithm for *3-Sat*. Note that #P $\subseteq$ FP would imply P $=$ NP, so the conjecture that #P $\nsubseteq$ FP is at least as likely as P $\neq$ NP.

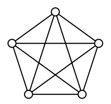
The *polynomial hierarchy* is an infinite hierarchy of complexity classes that generalizes the relationship between P and NP. The ith level of this hierarchy is denoted Σ_i^p. We have $\Sigma_0^p = \mathrm{P}$ and $\Sigma_1^p = \mathrm{NP}$, and $\Sigma_i^p \subseteq \Sigma_{i+1}^p$ for all $i > 0$. The hierarchy is said to *collapse to its ith level* if $\Sigma_i^p = \Sigma_{i+1}^p$. It is conjectured that the polynomial hierarchy does not collapse to any level.

Reductions. A *polynomial-time reduction* from a problem A to a problem B is a construction through which A could be solved in polynomial time given any polynomial-time algorithm for B. The problem A is said to be *polynomial-time reducible* to B if there exists such a reduction. A common type of polynomial-time reduction is that of a *mapping reduction* (a.k.a. *many-one reduction*), which defines a mapping from A-instances, I_A, to B-instances, I_B, such that the size of I_B is at most polynomial in the size of I_A, and such that the solution status for I_A is directly related to the solution status for I_B—for example, in the case of decision problems, such that I_A is a *yes*-instance to A if, and only if, I_B is a *yes*-instance to B.

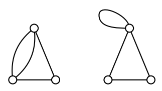
Hardness and completeness. For a given complexity class C, a problem is said to be *C-hard* if any problem in C is polynomial-time reducible to it. A C-hard member of C is said to be *C-complete*. The problem *3-Sat* is NP-complete, often taken as the canonical NP-complete problem. The canonical #P-complete problem is *#3-Sat*. Thus P $=$ NP if, and only if, *3-Sat* is polynomial-time solvable, and similarly #P $\subseteq$ FP if, and only if, *#3-Sat* is polynomial-time solvable.

1.1.2 Graph theory

A *graph* is a tuple (V, E), where V is a set of elements referred to as *vertices*, and E is a collection of pairs of vertices $(u, v) = uv$ referred to as *edges*. We let $n = |V|$ be the number of vertices, and $m = |E|$ the number of edges. One of these numbers is usually taken as a measure of the *size* of the graph. The graph is *directed* if each edge is considered as an ordered pair of vertices, so that $uv \neq vu$ in general, and *undirected* otherwise. An undirected graph can be made directed by supplying an *orientation* to determine the direction of the edges. All graphs in this thesis are assumed to be undirected, unless explicitly stated otherwise. A *subgraph* of a graph $G = (V, E)$ is a graph $H = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$ we say that H is a *spanning*. The *complement* of an undirected graph (V, E) is the graph $(V, V^2 \setminus E)$.


The nonplanar graph K_5.

Planar graphs. A *drawing* of a graph $G = (V, E)$ is a graphical representation with one dot for every vertex $v \in V$, and one curve connecting vertices u and v for every edge $uv \in E$. For directed graphs, an edge uv would be depicted as an arrow from u to v. A graph is *planar* if it can be drawn in the plane so that no two edges cross. Such a drawing is called a *planar drawing*. Given a planar drawing of a connected planar graph G, we can construct a *dual* G^* of G by introducing one node for each face of G in the drawing, including the unbounded outer face, and one edge for each pair of bordering faces; for example, is a planar dual (the only one) of .


Non-simple multigraphs.

Simple graphs. An edge of the type vv, i.e. between a vertex and itself, is called a *loop*. A graph is *simple* if it contains no loops and has at most one edge between any pair of vertices. The first part of this thesis concerns simple graphs only, but in the second part we will also consider non-simple graphs. To emphasize that a general graph is not necessarily simple we sometimes use the term *multigraph*.

Adjacency. Two vertices u, v in a graph $G = (V, E)$ are said to be *adjacent* or *neighbors* if $uv \in E$, and the edge uv is said to be *incident* to each of its end vertices u and v. A simple graph whose every vertex is adjacent to every other is *complete*, and a complete subgraph in G is a *clique* in G. The complement of a clique is an *independent set*. The *neighborhood* of a vertex $v \in V$ is the set of all neighbors of v,
$N(v)$ denoted $N(v)$, and the *degree* of v is the size of $N(v)$. The *maximum degree of* G,
$\Delta(G)$ denoted $\Delta(G)$, is the maximum degree over all vertices in G. A graph is *bipartite* it its vertex set can be partitioned into two subsets X and Y such that vertices in X only have neighbors in Y.

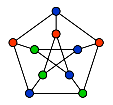
Walks. A *walk* of length l in a graph $G = (V, E)$ is an alternating sequence of vertices v_i and edges e_i of the type $W = (v_0, e_1, v_1, e_2 \ldots, e_l, v_l)$, such that $e_i = v_{i-1}v_i$ for $1 \leq i \leq l$. If G is simple then the walk is uniquely defined by its sequence of vertices, so we can write $W = (v_0, v_1, \ldots, v_l)$. The walk's *internal* *vertices* are $v_1, \ldots, v_{l-1}$. The walk is *closed* if $v_0 = v_l$, and *simple* if it has no repeated internal vertex.

Paths and cycles. The vertices and edges of a given walk in a graph form a subgraph that is called a *cycle* if the walk is closed, and a *path* otherwise. Note that two simple, closed walks form the same cycle if one is a reversal, or a cyclic permutation, of the other. An undirected graph containing no cycle is a *forest*. A directed graph containing no cycle is said to be *acyclic*.

Connectedness. A graph is *connected* if any two vertices are connected by a path. A *tree* is a connected forest. A maximal, connected subgraph of a graph is called a *component*. The number of components of a graph (V, E) is denoted by $\kappa(E)$; thus the graph is connected if, and only if, $\kappa(E) = 1$. A *cut* is a subset $\mathcal{C} \subseteq E$ whose removal increases the number of components of the graph. A singleton cut is called a *cut-edge*.

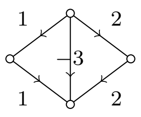
$\kappa(E)$

Other graph structures. A k*-coloring* of a graph $G = (V, E)$ is a partitioning of V into k classes (colors). A given k-coloring is said to be *proper* if no two adjacent vertices are in the same color class. The *chromatic number* of a graph G is the smallest number k such that G has a proper k-coloring.


Proper 3-coloring.

A *vertex cover* of a graph G is a vertex subset such that every edge of G is incident to at least one vertex in it. A *matching* of G is a subset of mutually nonadjacent edges, and a matching is *perfect* if it covers every vertex of G.

A k*-flow* in a directed graph (V, E) is an assignment $\phi : E \mapsto \pm\{1, \ldots, k-1\}$ to the edges, such that for any vertex $v \in V$ the sum of the assigned values for outgoing edges from v equals the sum over incoming edges to v. The number of k-flows is independent of the edge orientation, and for an undirected graph k-flows are considered with respect to an arbitrary fixed orientation.



4-flow.

1.2 Exponential-Time Complexity

With successful advances in exponential-time algorithms, an interest in the notion of *exponential-time complexity* followed naturally. For example, it would be interesting to know whether a given NP-complete problem is likely to require time exponential in the input size n, or whether there is hope for some algorithm with subexponential running time such as $2^{O(\sqrt{n})}$. For such questions, the notion of NP-hardness is not helpful, as it cannot distinguish between different superpolynomial time complexities. Instead, Impagliazzo, Paturi and Zane [57] introduced the *Exponential Time Hypothesis*, claiming a certain exponential-time lower bound for the problem *3-Sat*,

3-Sat

Input:	A 3-CNF formula φ with n variables and m clauses.
Task:	Decide whether φ is satisfiable.

The complexity claim is as follows.

Exponential Time Hypothesis (ETH):

There is a real number $c > 0$ such that no deterministic algorithm can decide *3-Sat* in time 2^{cn}.

Later, as a consequence of the so called *Sparsification lemma*, the same authors showed that ETH would also rule out the existence of subexponential-time algorithms for *3-Sat* in terms of the number of clauses.

Theorem 1.1 (Impagliazzo *et al.* [57]). *Assuming ETH, there is a real number $c > 0$ such that no deterministic algorithm can decide* 3-Sat *in time 2^{cm}.*

Under ETH we can get a more detailed picture of the complexity landscape within the class of NP-hard problems. Compared to classical complexity theory, we must here pay more attention to any blow-up of instance sizes in a reduction, so as to preserve information about an exponential-time lower bound.

1.2.1 Parameterized complexity

Assuming ETH to be true, many NP-complete graph problems can be shown to require time exponential in the size of the input graph, usually measured in the number of vertices n or edges m. However, for some problems there are other aspects, except the size of the graph, that affect the complexity. For example, finding a minimum vertex cover in a general graph requires time exponential in n under ETH [57], but if we fix a number k then deciding whether the graph has a vertex cover of size at most k is polynomial-time solvable: for every subset of k vertices—and there are $\binom{n}{k}n^{O(1)} \subset n^{O(k)}$ such subsets—check if it forms a vertex cover.

Detecting a vertex cover of given size is an example of a *parameterized problem*, where part of the input is a natural number k measuring some parameter of the input whose effect on the complexity we wish to highlight. Downey and Fellows give a systematic introduction to the theory of parameterized complexity in [32].

Fixed-parameter tractability. We can do better than the above suggested $n^{O(k)}$-solution to the parameterized vertex cover problem, by first performing a certain polynomial-time preprocessing step (see e.g. [37]) which either decides directly whether the input graph G has a vertex cover of size k, or returns a graph G' of size at most $2k$ such that G has a vertex cover of size k if, and only if, G' has one. Then the above brute-force approach can be applied to G'. In total, this gives a running time of $\binom{2k}{k}n^{O(1)} \subset 2^{O(k)}n^{O(1)}$. This is an example of *fixed-parameter tractability.*

Definition 1.1. *A computational problem is* fixed-parameter tractable *(FPT) with respect to parameter k if there is an algorithm for the problem with running time $f(k)n^{O(1)}$, where f is a function depending only on k, and n is the input size.*

This type of complexity may be acceptable for instances where k is small compared to n, provided the function $f(k)$ does not grow too rapidly.

There is an analogue concept of NP-hardness for parameterized problems called *W[1]-hardness*. We will not need the formal definition of the class W[1], but we note the following result.

Theorem 1.2 (Downey and Fellows [32]). *Assuming ETH, a problem that is W[1]-hard with respect to a parameter k cannot be fixed-parameter tractable with respect to k.*

Kernels. The preprocessing step mentioned above for the vertex cover problem is an example of a *kernel*: a mapping reduction from a parameterized problem to itself, that transforms any given instance into an equivalent instance of size $f(k)$, for some function f depending only on the parameter k and referred to as the *size* of the kernel. For example the above vertex cover problem, parameterized by solution size k, admits a kernel of size $2k$. Such *polynomial kernels* are especially sought after, as they usually translate into FPT-algorithms with a decent dependency on k, such as the $2^{O(k)}n^{O(1)}$-time algorithm mentioned above. It can be shown (see e.g [11, Theorem 1]) that a problem admits a kernel for a parameter k if, and only if, it is also fixed-parameter tractable with respect to k.

1.3 Overview of this thesis

Chapter 2 contains a brief review of algorithmic techniques of relevance for detecting long cycles in graphs: dynamic programming, inclusion-exclusion, color coding, and in particular a monomial sieving technique due to Koutis and Williams [69]. As a warm-up to Chapter 3, this monomial sieving technique is demonstrated on the problem of detecting induced cycles in degree-bounded graphs, such that the solution connects a small number of terminal vertices.

Chapter 3 concerns the problem of finding a cycle through a given set of specified vertices and/or edges. A Monte Carlo algorithm is presented with a running time that is exponential only in the number of specified elements. With high probability the algorithm finds a shortest solution. It uses several techniques discussed in Chapter 2, and the correctness follows from a subtle pairing argument. This is joint work with Andreas Björklund and Thore Husfeldt, and was published in [9]. Due to an observation by Magnus Wahlström, this algorithm could be adjusted to also give control of the parity of the length of a solution cycle. A problem concerning the parameterized complexity is discussed; this problem has now been solved by Magnus Wahlström.

Chapter 4 gives an introduction to the Tutte polynomial, and to the #P-hard problem of evaluating the Tutte polynomial of multigraphs at a given point in the (x, y)-plane. A brief survey of previous complexity results is given, especially the work by Dell *et al.* [30] on exponential-time complexity under #ETH, a counting analogue of ETH.

Chapter 5 concerns the exponential-time complexity of evaluating the Tutte polynomial of *simple* graphs, in particular for points on the line $x = 1$ corresponding to the *reliability polynomial*. In the framework proposed by Dell *et al.* [30] it is shown that, assuming #ETH, evaluation of the Tutte polynomial at a point with $x = 1$ requires time exponential in $\Omega(m/\log^2 m)$ for simple graphs of m edges, except for the point $(1, 1)$ which was already known to be polynomial-time solvable. As the problem is known to be solvable in vertex-exponential time for any point, this lower bound shows that the hardest instances are sparse graphs of roughly linear density. For $y > 1$ this is joint work with Thore Husfeldt, and was published in [56]. Joining efforts with Holger Dell, the result was finally extended to the full line, and incorporated into the journal paper [29] which now gives asymptotically tight (up to a polylogarithmic factor in the exponent) lower bounds under #ETH for the whole plane, except the line $y = 1$.

Chapter 6 concerns the exponential-time complexity of evaluating the Tutte polynomial of *planar* graphs. As an algorithm is known for this problem with a running time exponential only in $O(\sqrt{n})$, the question is whether this is asymptotically optimal under #ETH. By analysis of existing polynomial-time reductions, this is shown to indeed be the case for general points, as a matching lower bound is found for any point on the hyperbola $(x - 1)(y - 1) = 3$. For remaining nontrivial points in the plane, lower bounds exponential in $\Omega(n^{1/k})$ are found for various $k > 2$. In particular, a lower bound exponential in $\Omega(n^{1/8})$ is given for the line $y = 1$. To the best of the knowledge of the author, this is the first concrete superpolynomial lower bound on this line also for multigraphs in general.

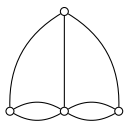
Part I

Algebraic Algorithms for Cycle Detection

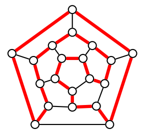
Chapter 2

Finding Cycles in Graphs

In 1736, Leonhard Euler [34] settled a much-debated question among the inhabitants of his hometown Königsberg: was it possible to walk through town, from some starting point and back again, such that one passed each of Königsberg's seven bridges exactly once? Euler proved the impossibility of such a walk by abstracting the problem, introducing mathematical objects we now know as *vertices*, *edges*, and *graphs*—graph theory was born.


Königsberg graph.

A seemingly similar puzzle is the *icosian game*, introduced by William Hamilton in 1857. Here the task is to find a walk along the edges of a dodecahedron, from any starting corner and back again, such that each of the 20 corners is passed exactly once. The existence of such walks was clear from the start, but *finding* one, at least the first time, still posed a pleasant leisure activity, and the game took form in a popular toy.


Icosian game.

Indeed, problems of finding various kinds of walks within some finite structure seems to have intrigued people for a long time. Except as a source of recreational games, problems of this type are often encountered in areas such as e.g. DNA-sequencing, network routing and traffic planning. In modern-day terms, the Bridges-of-Königsberg problem that Euler solved concerned the existence of an *Eulerian circuit*, i.e. a closed walk passing every edge in a graph exactly once, and the icosian game asks for a *Hamiltonian cycle*, i.e. a cycle through every vertex of a graph. From an algorithmic perspective these problems are quite different; while an Eulerian circuit can be found in polynomial time, deciding the existence of a Hamiltonian cycle is the famously NP-complete *Hamiltonian Cycle* problem. This difference in complexity is perhaps more easily grasped between the problem of finding a walk of length k in a graph, which is trivial as we can simply walk back and fourth along any edge, and the problem of finding a *path* of length k. The latter problem seems harder, and indeed it is NP-hard if k is part of the input, since $k = n$ gives the NP-complete *Hamiltonian Path* problem. This so called k-*Path* problem, and the corresponding k-*Cycle* problem, can thus be seen as parameterized generalizations of the *Hamiltonian Path* and *Hamiltonian Cycle* problems, respectively.

2.1 Algorithmic techniques for cycle finding

Over the years, a number of interesting algorithmic techniques have evolved from, or found nice application to, work on exact algorithms for the *Hamiltonian Cycle/Path* problem, or the k*-Path/Cycle* problem. This section describes four of the most influential such techniques. Three of these techniques will be useful in Chapter 3, where we study another generalization of the *Hamiltonian Cycle* problem.

We will only consider the cycle-version of these problems, but the same techniques apply to the path-versions. Indeed, given an input graph G to the *Hamiltonian Path* problem or the k*-Path* problem, we can construct a graph G' by adding a new vertex to G and making it connected to every other vertex. Then G' has a Hamiltonian cycle if, and only if, G has a Hamiltonian path, and G' has a cycle of length $k+1$ if, and only if, G has a path of length k.

2.1.1 Dynamic programming over subsets

A brute-force solution to the *Hamiltonian Cycle* problem enumerates all permutations of the vertices, and checks for each permutation whether it defines a closed walk in the graph. The resulting running time is thus $n!n^{O(1)}$. This upper bound was dramatically improved in 1962, when Bellman [5] and Held and Karp [53] independently discovered the following algorithm based on dynamic programming:

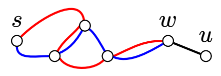
Algorithm H (*Compute the number of Hamiltonian cycles.*)

The input is a simple graph $G = (V, E)$, and an arbitrary starting vertex $s \in V$. For every $S \subseteq V$ and vertex u, let $T(S, u)$ be the number of paths in G from s to u using exactly the vertices in S.

H1. [Initialize table.] Set $T(\{s\}, s) = 1$, and all other entries to 0.

H2. [Dynamic programming.] Update the table as follows, for every subset $S \subseteq V$ containing s, and every vertex $u \in S$:

$$T(S,u) = \sum_{w \in N(u)} T(S \setminus u, w)\,.$$



Demonstration of two walks (red & blue) counted in some nonzero term of the sum.

H3. [Add relevant contributions.] Compute $\sigma = \sum_{u \in N(s)} T(V, u)$. This number counts every Hamiltonian cycle twice, once in each direction, so we return $H = \sigma/2$.

The running time is $2^n n^{O(1)}$ as we consider 2^n subsets $S \subseteq V$.

This use of dynamic programming greatly influenced subsequent algorithms for path- and cycle problems, and the $2^n n^{O(1)}$-bound for the *Hamiltonian Cycle* problem remained undisputed for almost 50 years.

2.1.2 The principle of inclusion and exclusion

An issue with the above dynamic programming solution is that except for exponential time, it also requires exponential space. A way around this is the following technique, based on the principle of inclusion and exclusion.

For $S \subseteq V$, let A_S be the set of closed walks of length n, starting and ending at some given vertex s, that *avoid* all vertices in S. Then the number H of Hamiltonian cycles satisfies

$$\begin{aligned} 2H = |A_\emptyset| &= |\overline{\bigcup_{u\in V} A_{\{u\}}}| \\ &= \sum_{S\subseteq V} (-1)^{|S|} |\bigcap_{u\in S} A_{\{u\}}| \\ &= \sum_{S\subseteq V} (-1)^{|S|} |A_S| \,, \end{aligned} \tag{2.1}$$

where the second equality follows from the principle of inclusion and exclusion; see e.g. (9) in [55]. Again, every Hamiltonian cycle is counted twice in $|A_\emptyset|$ (once in each direction). We compute each value $|A_S|$ by dynamic programming similar to Algorithm H: for every $u \in V$ and $k \leq n-1$, compute the number $T_S(k,u)$ of walks of length k from some starting vertex s to u that avoid the vertices in S, until we can return $|A_S| = \sum_{u\in N(s)} T_S(n-1,u)$. Note that for each subset $S \subseteq V$, the table T_S only requires quadratic space, and we only need to keep one such table at a time in memory, while still obtaining the time bound $2^n n^{O(1)}$.

This idea has been reinvented many times (see e.g. [67, 61, 4]) until the general technique gained popular attention in recent years; see [55] for an overview of applications. A variant of this technique is used in Björklund's $1.657^n n^{O(1)}$-time Monte Carlo algorithm [6] for the *Hamiltonian Cycle* problem—the first algorithm to beat the longstanding bound of $2^n n^{O(1)}$.

2.1.3 Color coding

For the k-*Cycle* problem, the brute-force solution has an upper bound of $O(n^k)$: for each choice of k vertices, check whether they constitute a simple cycle in the graph. Monien [80] gave an algorithm with running time $k!n^{O(1)}$, showing the problem to be fixed-parameter tractable with respect to the parameter k. Considering the known $2^n n^{O(1)}$-time algorithm for the *Hamiltonian Cycle* problem, a natural question was whether the factorial dependency on k of Monien's algorithm could be improved to exponential. Alon, Yuster and Zwick [1] provided a positive answer to this question, giving a $5.44^k n^{O(1)}$-time randomized algorithm for the problem, and a deterministic version in time $c^k n^{O(1)}$ for some large constant c, by introducing the technique of *color coding*.

The idea of color coding is to pick a random k-coloring of the vertices, $c : V \to [k]$, and then to look for a *colorful* k-cycle, i.e. a cycle containing exactly one vertex of each color. For any given such coloring c we can again use dynamic programming similar to Algorithm H to count the number of colorful k-cycles: for every subset $S \subseteq [k]$ and vertex pair $u, v \in V$, compute the number $T(S, u, v)$ of colorful paths from u to v that uses exactly the colors from S, each color exactly once. If the graph contains a k-cycle, then it has probability $k!/k^k > 1/e^k$ of becoming colorful by a random k-coloring, so an expected number of $O(e^k)$ colorings are needed to detect it. Hence the expected time to find an existing k-cycle is $e^k 2^k n^{O(1)} \in 5.44^k n^{O(1)}$.

This technique proved successful for finding other types of subgraphs as well, and moreover had the benefit of easily extending to weighted graphs. It was successfully applied for identifying certain interaction chains in protein interaction networks [86, 88], which renewed the interest in efficient algorithms for the k-*Cycle* problem.

2.1.4 Monomial sieving

The next question was whether the base of the exponential term in the new upper bound for the k-*Cycle* problem could be replaced by 2, as for the classical algorithm for the *Hamiltonian Cycle* problem. A Monte Carlo algorithm satisfying this was eventually found by Williams [97], by refining an algebraic 'sieving' technique introduced by Koutis [68]. The idea is to associate a certain polynomial with the graph, such that every term corresponds to a walk of length k, and such that a term is multilinear, i.e. contains no squared variable, if, and only if, the corresponding walk is simple.[1] While this polynomial is easily defined implicitly, for example as a recursion, it would not be practical to compute the expanded form; indeed, there mere number of terms to check would exceed the wanted time bound. Instead the following theorem is used to probabilistically sieve the given polynomial for monomials corresponding to k-cycles.

Theorem 2.1 (Koutis, Williams [69]). *Let $p(x_1, \ldots, x_t)$ be a polynomial of degree at most k, and suppose p can be represented by an arithmetic circuit of size polynomial in t, with no scalar multiplications. Then the existence of a multilinear term in p can be decided by a Monte Carlo algorithm in time $2^k t^{O(1)}$, with small probability of a false negative.*

In short, the algorithm uses the arithmetic circuit, typically a dynamic programming formulation, to evaluate the polynomial at a random point of coordinates from a certain group algebra over a finite field $\mathbf{F}_q$, such that any square evaluates to zero in this algebra. The remaining sum, corresponding to multilinear terms, will then evaluate to something nonzero with a probability given by the following lemma:

Lemma 2.1 (DeMillo-Lipton-Schwartz-Zippel). *Let $p \in \mathbf{F}_q[x_1, \ldots, x_t]$ be a nonzero polynomial of total degree at most d. Then, for $r_1, r_2, \ldots, r_t \in \mathbf{F}_q$ selected independently and uniformly at random,*

$$\Pr[\, p(r_1, r_2, \ldots, r_t) \neq 0\,] \geq 1 - \frac{d}{q}\,.$$

[1]The idea of associating a polynomial with a certain structure we want to find goes back to Tutte [91], who noted that a graph has a perfect matching if, and only if, the determinant of a certain matrix with variable entries is a nonzero polynomial. Lovász [79] was the first to realize the algorithmic potential of this result, when coupled with the DeMillo-Lipton-Schwartz-Zippel lemma.

The probability of a false negative in Theorem 2.1 decreases exponentially with the number of such random evaluations of the given polynomial, with a rate that depends on the size of the chosen field $\mathbf{F}_q$ but not on k. The exponential term of the running time comes from the cost of arithmetic operations in the group algebra.

This approach to the *k-Cycle* problem inspired Björklund to the breakthrough algorithm for the *Hamiltonian Cycle* problem [6], and by further development also led to algorithms for a number of packing problems [8]. The result of Chapter 3 is also inspired by this work.

2.2 Monomial sieving in action

As a warm-up to Chapter 3, this section demonstrates a somewhat elaborate application of the above monomial sieving technique.

A subgraph of a given graph G is said to be *induced* if it can be obtained from G by a sequence of vertex deletions. In particular, an *induced cycle* in a graph G is a cycle C containing no *chord* in G, i.e. an edge connecting two nonadjacent vertices on C. And induced cycle of length at least four is called a *hole*. A hole is *odd* or *even* according to the parity of its length.

It is a curious fact that, whereas we know how to decide in polynomial time whether a graph contains an even hole [24, 23, 21], it is still open whether odd holes can be detected in polynomial time. This problem received much interest over the years due to the *Strong Perfect Graph Conjecture*, which claimed that a graph is perfect[2] if, and only if, either itself or its complement contains an odd hole. This was finally proved by Chudnovsky *et al.* [22], yielding a complete characterization of perfect graphs, and also a polynomial-time algorithm for the problem, as it turned out that the problem of detecting either an odd hole *or the complement* of an odd hole is polynomial-time solvable [20]. This makes the unclear complexity of detecting odd holes quite intriguing.

Now suppose we want to check for odd holes in a graph of bounded degree. We will demonstrate how Theorem 2.1 can be used to do this, while also requiring the hole to pass through a small set of specified vertices. Thus we define the following problem.

Δd-l-K-Hole

Input:	A simple graph $G = (V, E)$ with n vertices, maximum degree d, a specified subset $K \subset V$ of size $k = (\log n)^{O(1)}$, and a number $l \geq 3$.
Task:	Decide whether G contains an induced cycle of length l, containing all vertices in K.

[2]A graph G is said to be *perfect* if the chromatic number of each induced subgraph H in G is the size of the biggest clique in H.

The Monte Carlo algorithm resulting from Theorem 2.1 will have a running time of $2^{dl}n^{O(1)}$. This is not very interesting in itself, as the problem can be solved in time $2^{l \log d}n^{O(1)}$ by simple branching, but the construction here serves to introduce several technical aspects we will encounter in Chapter 3.

2.2.1 Monomial sieving for induced cycle detection

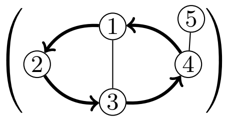
To apply Theorem 2.1, we seek a polynomial p associated with any input graph G, such that p has a multilinear term if, and only if, the graph G contains an induced cycle of length l through the specified vertices in K, and such that p can be represented by a commutative arithmetic circuit of size polynomial in the number of variables. The running time of the algorithm will be exponential in the degree of the polynomial p.

Defining the polynomial. For a given graph $G = (V, E)$ of n vertices and m edges, and a specified subset $K \subset V$ of size at most $(\log n)^{O(1)}$, we construct a polynomial p as follows:

Introduce a variable x_v for every vertex $v \in V$, and a variable y_e for every edge $e \in E$. For any $v \in V$, let $I(v)$ denote the set of incident edges to v. To each closed walk $W = (v_0, v_1, \ldots, v_{l-1}, v_0)$ in G, we associate the monomial $m(W)$ of degree at most dl given by

$$m(W) = \left(x_{v_0} \prod_{\substack{e \in I(v_0) \\ e \neq v_{l-1}v_0}} y_e \right) \cdot \left(x_{v_1} \prod_{\substack{e \in I(v_1) \\ e \neq v_0 v_1}} y_e \right) \cdots \left(x_{v_{l-1}} \prod_{\substack{e \in I(v_{l-1}) \\ e \neq v_{l-2}v_{l-1}}} y_e \right) . \tag{2.2}$$

For example, consider the monomial for the following walk of length 4 in a graph of maximum degree 3. (The walk is indicated by arrows in the graph.)

m  $= (x_1 y_{12} y_{13})(x_2 y_{23})(x_3 y_{13} y_{34})(x_4 y_{45} y_{41})$.

This monomial has degree 11, which is less than $4 \cdot 3$. Note that the variable y_{13} appears twice in the monomial, and the walk is a cycle, but not an induced cycle.

Let $\mathcal{W}_l^K$ be the set of all closed walks of length l in G that contain every vertex in K. Set

$$p(\mathcal{W}_l^K) = \sum_{W \in \mathcal{W}_l^K} m(W) .$$

Then $p(\mathcal{W}_l^K)$ is a polynomial of degree at most dl.[3]

[3]Note that for any l-cycle in G through the vertices in K, there are $2l$ corresponding walks in $\mathcal{W}_l^K$: one for every choice of starting vertex and orientation. Thus each monomial in $p(\mathcal{W}_l^K)$ will have an even coefficient of size at least $2l$. This poses no problem for the current application, but in the next chapter it will be necessary to avoid such a situation.

Correctness. We now argue that it is sufficient to look for multilinear terms in the polynomial $p(\mathcal{W}_l^K)$.

Lemma 2.2. *The polynomial $p(\mathcal{W}_l^K)$ contains a multilinear term if, and only if, G contains an l-hole containing every vertex in K.*

Proof. Let $W \in \mathcal{W}_l^K$. If W is not a cycle, then it contains some repeated internal vertex v, yielding a square variable x_v^2 in $m(W)$. If W is a cycle, but not induced (as in the above example), then there will be some chord of W, i.e. an edge $e \in E$ incident to two vertices v_i and v_j on W with $|i-j| > 1$ (indices considered modulo l). The corresponding edge variable y_e will be counted twice, in the ith and jth factor of $m(W)$, yielding a square factor y_e^2. Thus, if W is not an induced cycle, then $m(W)$ is not multilinear.

Now suppose W is an induced cycle in G. Being a cycle, W will pass no vertex more than once, so $m(W)$ is linear in every node variable x_v. Being induced, there are only two ways in which an edge $e \in E$ can have an endpoint on W: either e is incident to only one vertex v_i of W, or $e = v_i v_{i+1}$ for two consecutive vertices v_i, v_{i+1} of W (indices considered modulo l). In either case the variable y_e appears only once in $m(w)$, in the ith factor. Thus $m(W)$ is linear also in every edge variable y_e—it is a multilinear term in $p(\mathcal{W}_l^K)$. □

Arithmetic circuit. Then we show how to evaluate the polynomial efficiently.

Lemma 2.3. *The polynomial $p(\mathcal{W}_l^K)$ can be represented by an arithmetic circuit of size polynomial in the number of variables.*

Proof. We give a dynamic programming formulation of the polynomial $p(\mathcal{W}_l^K)$.

For every closed walk $W \in \mathcal{W}_l^K$, we would like to build the monomial $m(W)$ from smaller monomials corresponding to subwalks of W. To this end we define, for any walk $W = (v_0, v_1, \ldots, v_{r-1}, v_r)$, the monomial

$$\hat{m}(W) = \left(x_{v_0} \prod_{e \in I(v_0)} y_e \right) \cdot \left(x_{v_1} \prod_{\substack{e \in I(v_1) \\ e \neq v_0 v_1}} y_e \right) \cdots \left(x_{v_r} \prod_{\substack{e \in I(v_r) \\ e \neq v_{r-1} v_r}} y_e \right) . \tag{2.3}$$

Note that if $v_r \in N(v_0)$, so that we can append the edge $e = v_r v_0$ to the end of W to form a closed walk W', then $\hat{m}(W) = y_{v_{r-1}v_0} \cdot m(W')$.

For all lengths $r \leq l$, vertices $u, v \in V$, and subsets $S \subseteq K$, let $\mathcal{W}_r^S[u,v]$ be the set of walks of length r from u to v passing every vertex of S, and set

$$T(r, u, v, S) = \sum_{W \in \mathcal{W}_r^S[u,v]} \hat{m}(W) .$$

We have

$$\begin{aligned} p(\mathcal{W}_l^K) &= \sum_{W \in \mathcal{W}_l^K} m(W) \\ &= \sum_{v \in V} \sum_{u \in N(v)} \sum_{W \in \mathcal{W}_{l-1}^K[u,v]} \frac{\hat{m}(W)}{y_{uv}} \\ &= \sum_{v \in V} \sum_{u \in N(v)} T(l-1, v, u, K)/y_{uv} \,. \end{aligned} \tag{2.4}$$

This can be computed by dynamic programming as follows:

Algorithm C (*Circuit for* $p(\mathcal{W}_l^K)$).

The input is a simple graph $G = (V, E)$, an integer l $(0 \le l \le n-1)$ and a vertex subset $K \subset V$ of size $k = (\log n)^{O(1)}$.

C1. [Initialize table.] Set all table entries to 0. For all $u \in V$, set

$$T(0, u, u, S) = \begin{cases} x_u \prod_{e \in I(u)} y_e & \text{if } u \in K \text{ and } S = \{u\}\,, \\ x_u \prod_{e \in I(u)} y_e & \text{if } u \notin K \text{ and } S = \emptyset\,. \end{cases}$$

C2. [Dynamic programming.] Update the table as follows, for every $u, v \in V$ and every $S \subseteq K$:

If $v \in K$, then for each $S \ni v$ set

$$T(r+1, u, v, S) = x_v \sum_{w \in N(v)} \Big(T(r, u, w, S) + T(r, u, w, S \setminus v) \Big) \prod_{\substack{e \in I(v) \\ e \neq vw}} y_e \,.$$

(Here the value of $T(r, u, w, S)$ covers those u, w-walks that already visited v, and $T(r, u, w, S \setminus v)$ those that did not.)

If $v \notin K$, set

$$T(r+1, u, v, S) = x_v \sum_{w \in N(v)} T(r, u, w, S) \prod_{\substack{e \in I(v) \\ e \neq vw}} y_e \,.$$

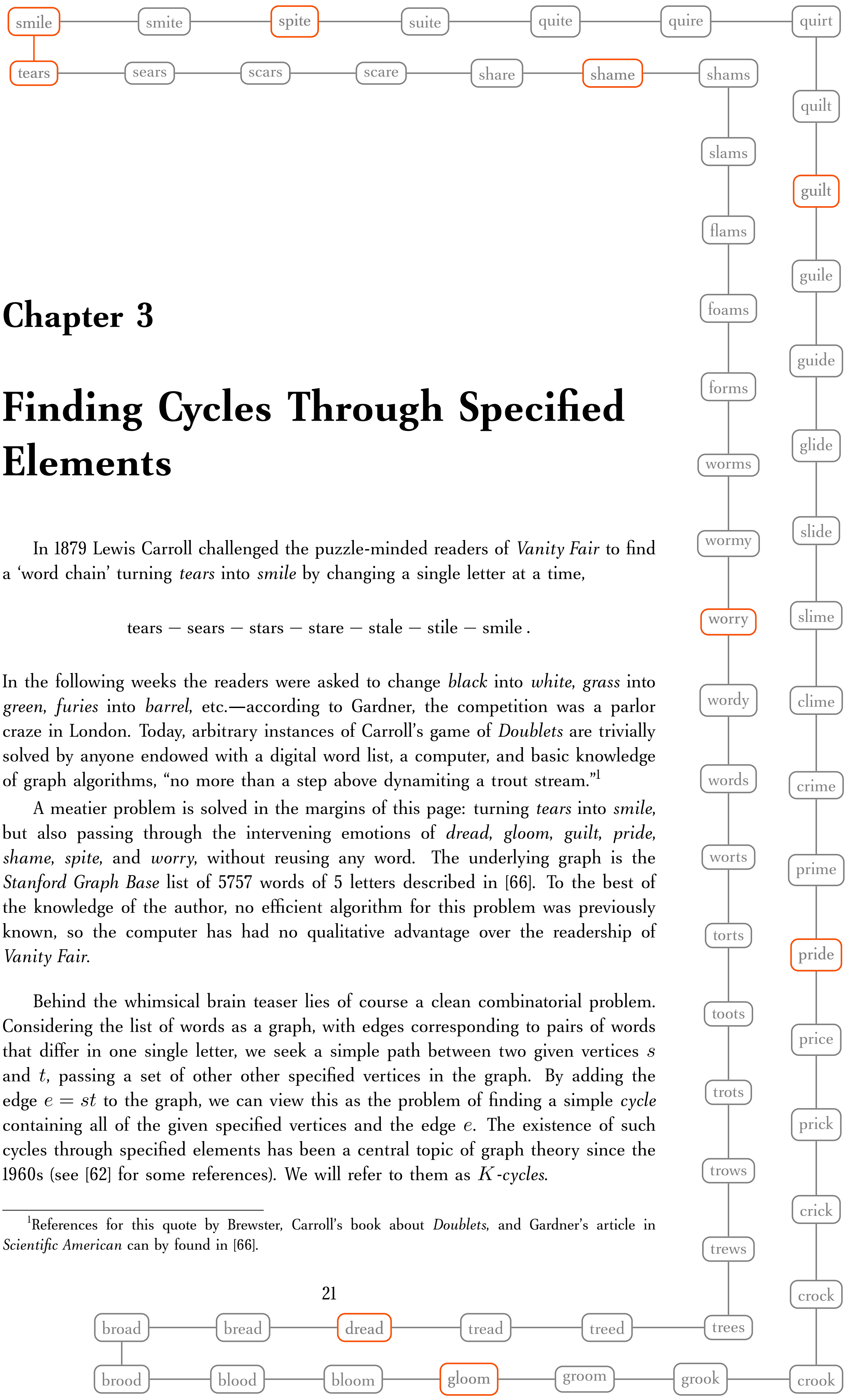
All other $T(S, r, b, y, z)$ remain at 0. Increment r and repeat C2 until $r = l$.

C3. [Add relevant contributions.] According to (2.4), return

$$p(\mathcal{W}_l^K) = \sum_{v \in V} \sum_{u \in N(v)} T(l-1, v, u, K)/y_{uv} \,.$$

This gives an arithmetic circuit for $p(\mathcal{W}_l^K)$ of size $O(2^k n^2 d^2)$, which is polynomial in $n + m$ since $k = (\log n)^{O(1)}$ by assumption. The number of variables of $p(\mathcal{W}_l^K)$ is $n + m$. □

Thus Lemma 2.2 and Lemma 2.3 shows that the polynomial $p(\mathcal{W}_l^K)$ satisfies the assumptions in Theorem 2.1. As the degree of the polynomial is dl, this yields a $2^{dl} n^{O(1)}$-time Monte Carlo algorithm for the problem Δd-l-K-*Hole*.



Chapter 3

Finding Cycles Through Specified Elements

In 1879 Lewis Carroll challenged the puzzle-minded readers of *Vanity Fair* to find a 'word chain' turning *tears* into *smile* by changing a single letter at a time,

tears — sears — stars — stare — stale — stile — smile .

In the following weeks the readers were asked to change *black* into *white*, *grass* into *green*, *furies* into *barrel*, etc.—according to Gardner, the competition was a parlor craze in London. Today, arbitrary instances of Carroll's game of *Doublets* are trivially solved by anyone endowed with a digital word list, a computer, and basic knowledge of graph algorithms, "no more than a step above dynamiting a trout stream."[1]

A meatier problem is solved in the margins of this page: turning *tears* into *smile*, but also passing through the intervening emotions of *dread*, *gloom*, *guilt*, *pride*, *shame*, *spite*, and *worry*, without reusing any word. The underlying graph is the *Stanford Graph Base* list of 5757 words of 5 letters described in [66]. To the best of the knowledge of the author, no efficient algorithm for this problem was previously known, so the computer has had no qualitative advantage over the readership of *Vanity Fair*.

Behind the whimsical brain teaser lies of course a clean combinatorial problem. Considering the list of words as a graph, with edges corresponding to pairs of words that differ in one single letter, we seek a simple path between two given vertices s and t, passing a set of other other specified vertices in the graph. By adding the edge $e = st$ to the graph, we can view this as the problem of finding a simple *cycle* containing all of the given specified vertices and the edge e. The existence of such cycles through specified elements has been a central topic of graph theory since the 1960s (see [62] for some references). We will refer to them as K*-cycles*.

[1] References for this quote by Brewster, Carroll's book about *Doublets*, and Gardner's article in *Scientific American* can by found in [66].

3.1 The *K-Cycle* problem

For a graph $G = (V, E)$ and a set $K \subseteq V \cup E$ of specified vertices or edges, a K*-cycle* in G is a simple cycle that includes all elements of K. By the *parity* of a K-cycle we refer to the parity of its length. We define the *K-Cycle* problem as follows,

K-Cycle

Input:	A simple graph $G = (V, E)$ with n vertices, and a subset $K \subseteq V \cup E$ of size k.
Task:	Decide whether G contains a K-cycle.

We can restrict our attention to the case where K contains only vertices; indeed, any specified edge uv can be replaced by adding a fresh vertex w to K and V and adding the edges uw and wv to E. This increases n by at most k.

The *K-Cycle* problem can be seen as a generalization of the *Hamiltonian Cycle* problem, corresponding to the case $K = V$. Consequently, the problem is NP-hard in general, and assuming ETH it cannot be solved in time $2^{o(k)}n^{O(1)}$ [57].

3.1.1 Relation to the *Disjoint Paths* problem

The *K-Cycle* problem has an interesting relation to the following well-known problem.

Disjoint Paths

Input:	A simple graph $G = (V, E)$ with n vertices, and a set of k vertex pairs $(s_1, t_1), \ldots, (s_k, t_k)$.
Task:	Decide whether G contains k paths $P_1, \ldots, P_k$, pairwise disjoint except possible overlapping endpoints, such that P_i connects s_i to t_i.

This problem is central in areas such as high-speed network routing and transportation networks; see [42] for a survey of applications. It was a seminal result of Robertson and Seymour's *Graph Minors Project* that the *Disjoint Paths* problem is fixed-parameter tractable with respect to the number k of terminal pairs [85]. We have the following connection to the *K-Cycle* problem:

Proposition 3.1. *The* Disjoint Paths *problem is computationally equivalent to a* K-oriented *version of the* K-Cycle *problem, in which the specified elements are required to be visited in a given order.*

Proof. Given a K-oriented *K-Cycle* instance with k specified vertices x_i to be visited in the order $x_0, x_1 \ldots, x_{k-1}, x_0$, we get an equivalent instance of *Disjoint Paths* by defining the ith pair of terminals as $(s_i, t_i) = (x_i, x_{i+1})$, with indices taken modulo k. Conversely, a given *Disjoint Paths* instance with terminal pairs (s_i, t_i) can be made into an equivalent K-oriented *K-Cycle* instance by adding specified edges $e_i = t_i s_{i+1}$, requiring these to be visited in the order $e_0, e_1 \ldots, e_{k-1}, e_0$, as illustrated in Figure 3.1. □

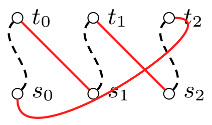


Figure 3.1: *Disjoint Paths* to K-oriented *K-Cycle*.

From this connection it is immediate from Robertson and Seymour's result that the *K-Cycle* problem is fixed-parameter tractable with respect to the number of specified elements: simply consider each possible order to visit the k specified elements. Thus we know that the problem can be solved in time $f(k)n^{O(1)}$ for some function $f(k)$. This should, however, be considered a theoretical result, as the time dependency on k that follows implicitly from Robertson and Seymour's constructions is quite extreme, involving repeated exponentiation and tower functions, which makes it completely impractical for any value of k. The first improvement over this was given recently by Kawarabayashi [62], with an algorithm whose dependency on k is doubly exponential in k^{10}.

3.1.2 Main theorem

Section 3.2 presents a Monte Carlo algorithm for an optimization version of the *K-Cycle* problem, with a running time that matches the exponential-time lower bound under ETH. The author is obliged to Magnus Wahlström for observing that the algorithm also solves a parity version of the problem.

Theorem 3.1. *Let G be an n-vertex graph, and K a set of k specified vertices of edges in G. A K-cycle of given parity in G can be found by a Monte Carlo algorithm in time $2^k n^{O(1)}$, with a small probability of a false negative, or of a solution that is not shortest possible.*

The problem of determining the length of a shortest K-cycle was not known to be fixed-parameter tractable, to the best of the knowledge of the author.

Note that this result is an improvement not only in the theoretical sense. Whereas previous algorithms would outperform the brute force solution only for inputs of galactic size, a straightforward implementation of the algorithm in Section 3.2 is able to find cycles through several specified elements in a graph with thousands of vertices. Also, the algorithm is short and conceptually simple, using nothing more complicated than dynamic programming. The correctness argument is a bit more subtle, but except for the DeMillo-Lipton-Schwartz-Zippel lemma, the presentation is self-contained.

If the graph has no K-cycle of the given parity, the algorithm will never report a false positive. If the graph has a K-cycle of the given parity, then such a cycle will be found with high probability, and it will be shortest possible with high probability.

3.1.3 Related work

The *K-Cycle* problem. For $k = 1$, the problem can be solved by breadth first search, and for $k = 2$ it corresponds to detecting two vertex-disjoint paths between the specified vertices, which is solvable by an adaption of the *Ford-Fulkerson* method; see e.g. [16, Chapter 9.1]. For $k = 3$, it can be solved in linear time by a dedicated algorithm [36, 73]. The mentioned algorithm by Kawarabayashi [62], involving some of the ideas of Robertson and Seymour [85], finds a cycle through k specified edges in polynomial time provided $k \in O((\log\log n)^{1/10})$. This remains the best known deterministic algorithm for general k.

The arguments needed to establish the correctness of previous algorithms for $k \geq 3$ are complicated. The $k = 3$ algorithm by [73] requires extensive case analysis partially omitted from the journal version and appearing only in [72]. The other algorithms rely on combinatorial results, in the extreme case of Robertson and Seymour's algorithm [85] the correctness proof requires hundreds of pages.

For directed graphs, the case $k = 1$ is solved as for undirected graphs, but already the detection problem for $k = 2$ is NP-hard [41].

Shortest K-cycle. For the optimization problem of finding a *shortest* K-cycle, little was known. Dean's list of open questions from a 1991 conference on graph minors [27] asks if the problem can be solved in polynomial time for fixed k; i.e. if the problem is fixed-parameter tractable with respect to k. For $k = 2$, the problem is a special case of minimum-cost network flow and solvable by textbook algorithms; see [90] for some early results. According to [27], the case $k = 3$ is solved by Fleischner and Woeginger in [36], though this is not made explicit. An algorithm for fixed $k > 3$ does not follow from Robertson-Seymour techniques, and the question seems to have remained open.

K-cycle of given parity. Kawarabayashi, Li, and Reed [64] give an algorithm for detecting a K-cycle of given parity. For fixed k the running time is polynomial in n, so the problem was known to be fixed-parameter tractable with respect to k, but the dependency on k is not given.

K-cycle of given length. The brute-force way to find a K-cycle of length l is of course to consider all $\binom{n}{l}$ candidate vertex subsets in G and see if they describe a K-cycle. Algorithms for long paths whose running time is exponential in the path length are known, and it is easy to change these algorithm to consider only such paths that visit K, as for the algorithm described in Section 2.2. For example, the algorithm in [8] can be modified to detect a K-cycle of length l in time $1.66^l n^{O(1)}$. While this may be competitive with previous algorithms for the he *K-Cycle* problem for $k > 3$, it would for example be time-consuming to find the solution on the title page, which has $l = 58$. Moreover, it seems difficult to modify these algorithms to be able to detect the *absence* of a K-cycle in time subexponential in n.

Induced K-cycle. Kobayashi and Kawarabayashi [63] give an algorithm for detecting an induced K-cycle in a planar graph in time $w^{O(w)}n^2$, where $w = k^{2/3}$. In particular, their algorithm runs in polynomial time for $k = o((\log n/\log\log n)^{2/3})$. For general graphs, this problem is NP-complete even for $k = 2$, and the ideas of this chapter do not seem to easily lend themselves to this induced version of the problem.

3.1.4 Technique

The algorithm presented in this chapter uses the monomial sieving idea discussed in the previous chapter. Compared to other recent papers using this technique, such as [6, 8, 69, 97], the construction here is quite simple, but the analysis is more delicate.

One could view the determinant summation technique of [6], for the *Hamiltonian Cycle* problem, as detecting a cycle of length n through $n/2$ specified elements. The running time there is exponential in the number of vertices *between* the specified ones. For the algorithm below the situation is reversed: the specified vertices are exponentially expensive, and the vertices in-between are cheap.

3.2 Algorithm

We first introduce some relevant concepts.

3.2.1 Terminology and definitions

Given a walk $W = (v_0, v_1, \ldots, v_l)$, we let $V(W)$ and $E(W)$ denote the set of vertices and edges of W, respectively. A *subwalk* of a walk W is a walk of the form $(v_i, v_{i+1}, \ldots, v_j)$ for some i, j with $0 \le i \le j \le l$. A subwalk is a *prefix* of W if $i = 0$, and a *suffix* of W if $j = l$. Given a vertex subset S, an S*-walk* is a walk that includes every vertex from S exactly once. A *digon* is a walk of the form u, v, u. An S*-digon* is a digon u, v, u where $v \in S$.

A set of closed walks. We define a certain set of closed walks, constituting candidate K-cycles. Fix an arbitrary starting vertex $a \in K$ and an arbitrary total order $\prec$ of the vertices in the neighborhood $N(a)$.

Definition 3.1. *For given length l ($2 \le l \le n$), define the set $\mathcal{C}_l$ of closed walks $W = (v_0, v_1, \ldots, v_{l-1}, v_0)$ with the following properties:*

- P1 *(start) $v_0 = a$,*
- P2 *(K-walk) every vertex in K appears exactly once on W,*
- P3 *(oriented) $v_1 \prec v_{l-1}$*
- P4 *(no K-digons) every internal vertex on W that belongs to K has different predecessors and successors on W; i.e. if $v_i \in K$ with $1 \le i < l$ then $v_{i-1} \ne v_{i+1}$; see Figure 3.2 (iii).*

The set $\mathcal{C}_l$ includes the K-cycles of length l, but can contain other, self-intersecting closed walks as well. Property P3 is used to ensure that a cycle and its reversal are considered only once; we arbitrarily decide to consider the cycle's direction that goes from a to the lower-ordered neighbour. Property P4 is more technical.

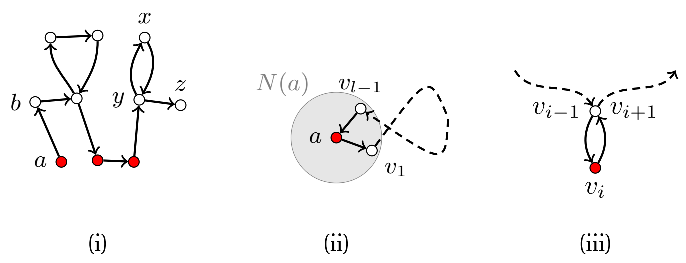


Figure 3.2: Walks in a graph with K-vertices in red. (i) A K-walk. (ii) A closed walk $v_0, v_1, ..., v_l$ with $v_0 = v_l = a$. (iii) A K-digon. All K-digons are forbidden in $\mathcal{C}_l$, while the digon y, x, y in (i) is allowed.

A function on sets of walks. With foresight, we will work in $\mathbf{F}_q$ with $q = 2^{1+\lceil \log n \rceil}$, a finite field of characteristic 2 and size $q \geq 2n \geq 2l$.

Definition 3.2. *For every edge $e \in E$ in a graph $G = (V, E)$, associate a value $f(e) \in \mathbf{F}_q$. Extend the definition of f to walks $W = v_0, v_1, \ldots, v_l$ in G by*

$$f(W) = f(v_0v_1)f(v_1v_2)\cdots f(v_{l-1}v_l)\,,$$

and to sets $\mathcal{W}$ of walks by

$$f(\mathcal{W}) = \sum_{W \in \mathcal{W}} f(W)\,.$$

3.2.2 Algorithm

We are ready to present the main algorithm. Essentially, we define f by choosing $f(e_1), \ldots, f(e_m)$ at random from $\mathbf{F}_q$ and then check whether $f(\mathcal{C}_l)$ is nonzero for increasing lengths l.

Algorithm M (*Find the length of a shortest K-cycle of given parity*).

The input is a simple graph $G = (V, E)$, a specified vertex subset $K \subseteq V$.

M1. [Initialize.] Choose $f(e) \in \mathbf{F}_q$ for each $e \in E$ uniformly at random. Choose a starting vertex $a \in K$ and an ordering of $N(a)$ arbitrarily. Set $l = |K|$ or $l = |K| + 1$, according to the given parity.

M2. [Iterate over all lengths.] Compute $f(\mathcal{C}_l)$ using dynamic programming (Algorithm F below). If $f(\mathcal{C}_l) \neq 0$, answer that G contains a K-cycle of the given parity of length at most l, and that, with high probability, l is the shortest length of such a cycle. Otherwise increase l by 2 and repeat step M2 until $l \geq |V|$.

M3. [Admit defeat.] Answer that no K-cycle of the given parity was found.

This algorithm establishes Theorem 3.1. The proof of correctness is in Section 3.3.

Note that if the algorithm reports a K-cycle at length l, then this is probably the length of a shortest K-cycle, but as we will see there is a possibility that a shorter K-cycle was missed on a previous iteration, i.e., that we got a false negative at that iteration. The probability of this situation is bounded by a fixed number $p < 1/2$, just as the probability of a false negative in general, and we can simply repeat the algorithm until we are sufficiently confident of the answer.

Dynamic programming. The values $f(\mathcal{C}_l)$ in step M2 can be computed using dynamic programming over the subsets of K and the length of the walk's prefix, in a similar vein as Algorithm H in the previous chapter. We only need to maintain some extra information about the last two vertices on a walk's prefix (in order to avoid building an K-digon) and the second vertex (in order to determine the orientation of the final closed walk), as follows.

For every vertex subset $S \subseteq K$, vertices $b, y, z \in V$, and length $r \leq n-1$, define the values

$$T(S, r, b, y, z) = \sum_{W \in \mathcal{W}_r} f(W)\,,$$

where the sum is taken over the set $\mathcal{W}_r$ of all walks $W = v_0, \ldots, v_r$ with the properties

S1 (start and end) $v_0 = a$, $v_1 = b$, $v_{r-1} = y$, and $v_r = z$,

S2 (S-walk) every vertex in S appears exactly once on W, and no other vertex from K appears on W,

S3 (no S-digons) if $v_i \in S$ with $1 \leq i < r$ then $v_{i-1} \neq v_{i+1}$.

The values $T(S, r, b, y, z)$ can be computed for all arguments by dynamic programming in time $O(2^k n^5)$; the details are given below.

Algorithm F (*Compute* $f(\mathcal{C}_l)$).

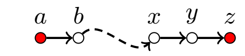
The input is a simple graph $G = (V, E)$, a vertex subset $K \subseteq V$ with $a \in K$ a fixed starting vertex, an integer l ($0 \leq l \leq n-1$), and values $f(e)$ for each $e \in E$.

F1. [Initialize table.] Set all table entries to 0. Set $T(\{a\}, 0, b, a, b) = f(ab)$ for each $b \in N(a) \setminus K$, and $T(\{a, b\}, 0, b, a, b) = f(ab)$ for each $b \in N(a) \cap K$. Set $r = 2$.

This covers all walks of length zero that satisfy properties S1, S2, and S3.

F2. [Dynamic programming.] Update the table as follows, for every $b, y, z \in V$ with $ab \in E$ and $yz \in E$ and every $S \subseteq K$. *(These are the S-walks of length r of the form $a, b, \ldots, y, z$.)*

The situation for a nonzero term of the sum looks like this:



If $z \in K$ then for each $S \ni z$ set

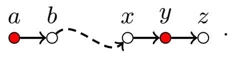
$$T(S, r+1, b, y, z) = f(yz) \sum_{x \in V} T(S \setminus z, r, b, x, y)\,.$$

By induction, the prefix $(a, b, \ldots, x, y)$ is an $(S \setminus z)$-walk, so S2 remains satisfied. In particular, $z \neq x$, so S3 is satisfied even if $y \in S$.

If $z \notin K$ then set

$$T(S, r+1, b, y, z) = \begin{cases} f(yz) \sum_{x \in V} T(S, r, b, x, y) & \text{if } y \notin K\,, \\ f(yz) \sum_{\substack{x \in V \\ x \neq z}} T(S, r, b, x, y) & \text{if } y \in S\,. \end{cases}$$

The condition $x \neq z$ for the case $y \in S$ ensures that S3 is satisfied in a situation like this:



All other $T(S, r, b, y, z)$ remain at 0. Increment r and repeat F2 until $r = l$.

F3. [Add relevant contributions.] Return

$$f(\mathcal{C}_l) = \sum_{b \in N(a)} \sum_{\substack{y \in N(a) \\ b \prec y}} f(ya) \sum_{x \in V} T(K, l-1, b, x, y)\,. \tag{3.1}$$

(Here the innermost sum gives the contribution of all walks only missing the edge $e = ya$ to be in $\mathcal{C}_l$.)

3.2.3 Implementation details

For *finding* a cycle, rather than merely reporting its existence, we search through all $v \in N(a)$ with binary search to detect a K-path from v to a of length l; then, for each successful v, through $v' \in N(v)$ for a K-path from v' to a of length $l-1$, and so on. Note that a K-path between vertices a and v can be detected by the algorithm by temporarily adding the edge $e = va$ to K, and asking for a K-cycle. This increases the running time by a factor $l \log n$.

Time improvements. The dynamic programming solution above is presented without attention to efficiency, and the polynomial factor is thus unnecessarily large. The program can be sped up considerably, for example by iterating over $x \in N(y)$ instead of $x \in V$, and by treating outgoing and incoming edges around $N(a)$ differently to break symmetry instead of considering orientations. The implementation used to find the word-chain in the introduction runs in time $O(2^k n^2 l)$.

Space improvements. The space requirement of Algorithm F is exponential in k. We can get this down to polynomial in n and k by using the principle of inclusion and exclusion as mentioned in the previous chapter:

For a subset $S \subseteq K$ of specified vertices, let $\mathcal{C}_l[S]$ be the set of closed walks of length l containing all of S, none of $K \setminus S$, and satisfying properties P1, P3 and P4. Then $\mathcal{C}_l = \mathcal{C}_l[K]$. Let $\mathcal{C}_l[i, S]$ be the set of closed walks of length l containing exactly i vertices from S, none of $K \setminus S$, and satisfying properties P1, P3 and P4. We have

$$\sum_{S \subseteq K} f(\mathcal{C}_l[S]) = \sum_{i=0}^{k} f(\mathcal{C}_l[i, K])\,.$$

By the principle of inclusion and exclusion—we here use a different formulation than that used for (2.1)—we get

$$f(\mathcal{C}_l[K]) = \sum_{S\subseteq K} (-1)^{|K|-|S|} \sum_{i=0}^{k} f(\mathcal{C}_l[i,S])\,.$$

Thus

$$f(\mathcal{C}_l) = \sum_{S\subseteq K} \sum_{i=0}^{k} f(\mathcal{C}_l[i,S]) \mod 2\,, \tag{3.2}$$

and for all $S \subseteq K$ the inner sum in (3.2) can be computed by dynamic programming in $\mathbf{F}_q$ in a similar fashion as Algorithm F above, over the walk's prefix/suffix, length and *number* of visited K-vertices, rather than over the subsets themselves.

3.3 Correctness

To see that Algorithm M is correct, we use a polynomial-sieving formulation akin to the one in Section 2.2. We consider the polynomial $p_l \in \mathbf{F}_q[x_1,\ldots,x_m]$ defined for a given graph $G = (V,E)$ and specified subset $K \subseteq V$ by

$$p_l(x_1,\ldots,x_m) = \sum_{W\in\mathcal{C}_l} \prod_{e_i\in E(W)} x_i\,. \tag{3.3}$$

Then $f(\mathcal{C}_l) = p_l(f(e_1),\ldots,f(e_m))$. From the definition, it is clear that p_l is a polynomial in m variables of total degree l.

The following result implies correctness of Algorithm M. We let $\pi(l)$ denote the parity of l.

Lemma 3.1. *Let $G = (V,E)$ be a simple graph with $K \subseteq V$, and let $p_l \in \mathbf{F}_q[x_1,\ldots,x_m]$ be defined as in* (3.3). *If G has no K-cycle of parity $\pi(l)$ and length at most $l-2$, then it has a K-cycle of length l if and only if p_l is nonzero.*

We break this lemma into Lemma 3.2 and Lemma 3.3 below.

Since Algorithm M chooses the values $f(e_1),\ldots,f(e_m)$ at random, we can view its behaviour as evaluating $p_l(x_1,\ldots,x_m)$ at a random point in $\mathbf{F}_q^m$. If p_l is identically zero, then Algorithm M never reports a nonzero value. Conversely, the probability of a false negative, that is, reporting 0 when p_l is not identically zero, is bounded by l/q by the DeMillo-Lipton-Schwartz-Zippel lemma (Lemma 2.1).

We note that if a false negative has been reported at length l, Algorithm M will continue to search for longer K-cycles. Then Lemma 3.1 no longer applies for these subsequent iterations, and the algorithm may report something nonzero at a higher iteration, regardless of whether there are longer K-cycles or not in the graph. Thus, if the algorithm terminates with $p_l \neq 0$ for some l, we can be completely certain that there is a shortest K-cycle of length *at most* l, but with a probability less than l/q the shortest K-cycle is actually shorter than l. If there is no K-cycle of the reported length l in the graph, then this is necessarily discovered during the search step, and we simply repeat the algorithm until we find a shorter length. Otherwise we find a K-cycle of length l that has a small probability of not being a shortest one of the given parity.

It remains to establish Lemma 3.1.

Lemma 3.2. *If G has a K-cycle of length l, then p_l is nonzero in $F_q[x_1, \dots, x_m]$.*

Proof. A K-cycle $C \in \mathcal{C}_l$ contributes the monomial

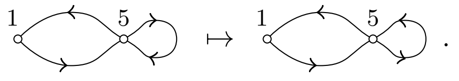
$$m_C = \prod_{e_i \in E(C)} x_i$$

to p_l. This monomial depends only on the set of edges on C. With properties P3 and P1, the simple cycle C can be recovered from $E(C)$, so m_C will be a unique contribution to p_l and thus constitutes a nonzero term over F_q. $\square$

We now argue that all walks in $\mathcal{C}_l$ must pair up and cancel whenever G has no K-cycle of length at most l and of the same parity as l. To show this, we define a *fixed-point-free involution* on $\mathcal{C}_l$, that is, a mapping $\phi : \mathcal{C}_l \to \mathcal{C}_l$ such that $\phi(\phi(W)) = W$ and $\phi(W) \neq W$ for all $W \in \mathcal{C}_l$. Such a mapping partitions $\mathcal{C}_l$ into pairs of walks $\{W, \phi(W)\}$.

Lemma 3.3. *If G has no K-cycle of parity $\pi(l)$ and length at most l, then there is a fixed-point-free involution $\phi : \mathcal{C}_l \to \mathcal{C}_l$ such that $f(W) = f(\phi(W))$ for each $W \in \mathcal{C}_l$.*

The basic idea of the proof is to define ϕ like so: Every walk in $\mathcal{C}_l$ that is not a K-cycle must contain a repeated internal vertex. For example, in a graph of nine vertices labeled 1–9, the closed walk 123567541 contains the repeated internal vertex 5. We want to map this walk to the walk $123\overleftarrow{5675}41 = 123576541$ obtained from reversing the cycle between the first and last occurrence of 5, like this:



The resulting closed walk is different, yet corresponds to the same monomial since it contains the same edges. For this idea to work for general closed walks, we need to be careful about internal palindromes ($123\overleftarrow{567654}1$) and how to choose the repeated internal vertex.

Proof of Lemma 3.3. We first need some extra terminology. If the end vertex of a walk W and the starting vertex of another walk W' are neighbors, we write WW' for the concatenated walk. We let $\overleftarrow{W}$ denote the reversal of the walk W, and say that W is a *palindrome* if $W = \overleftarrow{W}$. A palindrome is *nontrivial* if it contains more than one vertex. A repeated internal vertex on a walk is *critical.* If u is a critical vertex in W, define

$[uWu]$: The subwalk in W of maximum length starting and ending at u.

Given a critical vertex u of W, we can decompose W as

$$W = X[uWu]Z\,, \tag{3.4}$$

for some prefix X and suffix Z of W, such that neither X nor Z contains u. By *contraction* of $[uWu]$ in W we refer to the operation $X[uWu]Z \mapsto XuZ$.

Let G be a graph without K-cycles of parity $\pi(l)$ and length at most l. We define the mapping $\phi : \mathcal{C}_l \to \mathcal{C}_l$ as follows. Given a walk $W \in \mathcal{C}_l$, let v be the output from the following procedure:

Algorithm R (*Find* v).

```
The input is $W \in \mathcal{C}_l$.
R1. Let $i = 0$ and $W_0 = W$.
R2. Let $v$ be the first critical vertex in $W_i$.
R3. Decompose $W_i$ as $X[vW_iv]Z$. If $[vW_iv]$ is a palindrome, set $W_{i+1} = XvZ$,
      increment $i$, and go to R2.
R4. Return $v$. ($[vW_iv]$ is not a palindrome.)
```

Decompose W as $X[vWv]Z$ for the vertex v returned by Algorithm R, and let

$$\phi : X[vWv]Z \mapsto X\overleftarrow{[vWv]}Z\,.$$

(*For example, on input* 12345432345467861, *Algorithm R returns* $v = 6$*:*

$$\begin{aligned} W_0 &= 1[2345432]345467861\,,\\ W_1 &= 123[454]67861\,,\\ W_2 &= 1234[6786]1\,, \end{aligned} \tag{3.5}$$

so

$$\begin{aligned} \phi(12345432345467861) &= 1234543234546\overleftarrow{786}1\\ &= 12345432345468761\,.) \end{aligned}$$

To see that ϕ is well-defined, we need to show that Algorithm R does output a critical vertex v on input $W \in \mathcal{C}_l$. We first show that Algorithm R satisfies the following invariant:

I1. $W_i \in \mathcal{C}_{l-2m}$ for some $m \geq 0$.

The proof is by induction on i, with the case $i = 0$ established by the input requirement. Suppose that $W_{i-1} \in \mathcal{C}_{l-2m'}$ for some $m' \geq 0$. Note that W_i is obtained from W_{i-1} by contracting some nontrivial palindromic subwalk. Since any palindromic walk has even length, the length of W_i must be $l - 2m$ for some $m > m'$. It remains to check that W_i satisfies properties P1–P4.

Firstly, no nontrivial palindrome $[uW_{i-1}u]$ can contain any specified verticex $x \in K$, for as W_{i-1} satisfies property P4, x cannot be the middle vertex in $[uW_{i-1}u]$, and as W_{i-1} satisfies property P2, x cannot be any of the other vertices of $[uW_{i-1}u]$, because these are necessarily critical. Thus, W_i must contain every vertex in K that is present in W_{i-1}, so properties P1 (since $a \in K$) and P2 will remain satisfied in W_i. Also P3 remains satisfied, since the given total order of vertices in $N(a)$ is unaffected by contractions. As for property P4, note that if the jth node v_j on W_{i-1} is removed in W_i, then its neighbors v_{j-1} and v_{j+1} on W_{i-1} must both appear in the palindrome $[uW_{i-1}u]$ that is contracted in W_i, so by the above argument $v_{j-1}, v_{j+1} \notin K$. This means that neighbors on W_{i-1} of any vertex in K must be preserved in W_i, so no K-digon can appear in W_i as a result of the contraction, and property P4 remains satisfied. We conclude that $W_i \in \mathcal{C}_{l-2m}$ for some $m > m' \geq 0$.

It follows that Algorithm R must terminate; otherwise, W_i would eventually have no critical vertex, and by invariant I1 be a K-cycle in G of length $l - 2m$, contrary to assumption. Also, the output vertex v must be critical in the input walk W, because $V(W_i) \subseteq V(W)$.

To see that ϕ is an involution on $\mathcal{C}_l$, write

$$W = X[vWv]Z \quad \text{and} \quad \phi(W) = X\overleftarrow{[vWv]}Z\,.$$

We first note that Algorithm R outputs the same vertex v also on input $\phi(W)$. This follows as W and $\phi(W)$ share the same prefix X, so Algorithm R will perform the same contractions until it reaches v. It then terminates, returning v, since $\overleftarrow{[vWv]}$ is not a palindrome if $[vWv]$ is not a palindrome. Thus

$$\phi(\phi(W)) = X\overleftarrow{\overleftarrow{[vWv]}}Z = W\,.$$

To see that ϕ is fixed-point-free, it suffices to show that $[vWv]$ is not a palindrome for the output vertex v. It is clear that at some step of Algorithm R, the subwalk $[vW_iv]$ is not a palindrome. The following invariant then provides proof by contrapositive.

> **I2.** If u is a critical vertex in W_i such that $[uW_iu]$ is a palindrome, then $[uW_{i+1}u]$ will also be a palindrome.

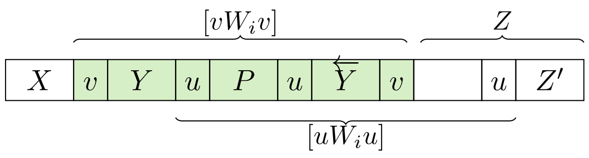
We verify I2 by considering cases. Let v be the first critical vertex in W_i. If $v = u$, then $[uW_{i+1}u] = u$, a palindrome. If $[vW_iv]$ contains no copy of u, or is not a palindrome, then step R3 of the algorithm leaves $[uW_iu]$ unaffected, so $[uW_{i+1}u] = [uW_iu]$, a palindrome. Otherwise we can write W_i in two ways using (3.4),

$$W_i = X[vW_iv]Z = X'[uW_iu]Z' \,, \qquad (u \neq v)\,,$$

where $[vW_iv]$ is a palindrome containing u. Thus, for some (possibly empty) subwalk Y not containing u, and some palindrome P, we have

$$[vW_iv] = \begin{cases} vYu\overleftarrow{Y}v & \text{if } u \text{ appears only once on } [vW_iv]\,; \\ vYuPu\overleftarrow{Y}v & \text{otherwise}\,. \end{cases}$$

We can handle both cases at once with the notational convention that $u = uPu$ when P is the empty palindrome. As v is the first critical vertex in W_i, the critical vertex u does not appear in the prefix X. If u does not appear in the suffix Z either, then $W_{i+1} = XvZ$ contains no copy of u, so $[uW_{i+1}u]$ is the empty walk, pathologically a palindrome. The final, and interesting, case is when u appears on the suffix Z. (An example is $u = 4$ in W_0 of equation (3.5).) Pictorially,



Since uW_iu is a palindrome by assumption, and the suffix Z does not contain v, it must be that $Z = YuPuZ'$. Thus, after the contraction in step R3, we have

$$W_{i+1} = XvZ = XvYuPuZ' \,.$$

As neither X, Y nor Z' contain u, this gives $[uW_{i+1}u] = uPu$, a palindrome.

We have established that ϕ is a fixed-point-free involution on $\mathcal{C}_l$. Finally, we have $f(W) = f(\phi(W))$, since $E(W) = E(\phi(W))$. $\square$

3.4 Kernelization issues

As the *K-Cycle* problem is fixed-parameter tractable with respect to $k = |K|$, we know that it must admit a kernel for k as well. This leads us to the following question:

Does the K-Cycle *problem, parameterized by the number k of the specified elements, admit a* polynomial *kernel?*

The related *Disjoint Paths* problem, parameterized by the number of specified terminal pairs, was recently shown *not* to admit a polynomial kernel, unless the polynomial hierarchy collapses to its third level. Thus, under this same assumption, we know that the 'K-oriented' version of *K-Cycle*, described in Proposition 3.1, admits no polynomial kernel in k. A number of attempts were made to prove a similar result for the standard *K-Cycle* problem, but all were inconclusive. This section gives a brief report of approaches that were considered.

Latest developments. We now have an explanation to the failure of the below described attempts; it has, quite recently, come to the attention of the author that Magnus Wahlström has found a polynomial kernel for the *K-Cycle* problem. This marks a fundamental difference to the *Disjoint Paths* problem. The exponential-time upper bound for the *K-Cycle* problem remains unchanged, however, as the newly found kernel does not provide a better exponential dependency on k than the algorithm presented in this chapter.

3.4.1 Kernelization lower bounds via composition

In [12], Bodlaender *et al.* initiated the study of superpolynomial lower bounds of kernel sizes for fixed-parameter tractable problems. To this end they introduced the concepts of *or-composition* and *and-composition* for a parameterized problem.

Definition 3.3. *An* or-composition *for a parameterized problem $\mathcal{P}$ is a polynomial-time algorithm with the following behavior:*

Input: *A number of $\mathcal{P}$-instances $(P_1, k), \ldots, (P_r, k)$.*
Task: *Find a $\mathcal{P}$-instance (P', k') such that*
1. *k' is polynomial in k, and*
2. *(P', k') has a solution $\Leftrightarrow$ (P_i, k_i) has a solution for some i.*

An *and-composition* is defined analogously, with requirement *2.* sharpened to require that (P_i, k_i) has a solution for *all* $i \in \{1, \ldots r\}$.

Together with the following theorem, these types of compositionality can be used to rule out polynomial kernels. The result for or-composition is due to Bodlaender *et al.* [12] and Fortnow and Santhanam [38]; the result for and-composition is due to Bodlaender *et al.* [12] and Drucker [33].

Theorem 3.2 (Bodlaender *et al.* [12], Fortnow and Santhanam [38], Drucker [33])**.** *Let $\mathcal{P}$ be an NP-complete problem that is fixed-parameter tractable with respect to parameter k, and suppose $\mathcal{P}$ is or-compositional, or and-compositional, with respect to k. Then $\mathcal{P}$ admits no polynomial kernel in k, unless the polynomial hierarchy collapses to its third level.*

Several NP-complete, fixed-parameter tractable problems are known to be or-compositional. This is easily seen for problems such as k*-Cycle* or k*-Path* parameterized by k, where we can simply consider the disjoint union of the input instances. No such obvious or-composition presents itself for the *K-Cycle* problem, as e.g. the total set of specified elements will be spread out over the input instances.

For other problems the notion of and-composition is more natural. For example, the NP-complete problem of finding a Hamiltonian path with given start and end, parameterized by the width of a given tree-decomposition, is easily seen to be and-compositional by forming a chain of the input graphs, connecting them by their specified end nodes [14]. Again, the situation is not immediate for the *K-Cycle* problem. For example, using some chain-construction in the style of the mentioned example would multiply our parameter $k = |K|$ by the number of input graphs.

Cross-composition. Recently, Bodlaender *et al.* [13] generalized the concept of or-composition to allow for a broader range of input instances, instances that are not parameterized and not necessarily of the same problem.

Definition 3.4. *A parameterized problem $\mathcal{P}$ is* cross-composed *by a non-parameterized problem $\mathcal{Q}$ if there is a polynomial-time algorithm with the following behavior:*

Input: *A number of $\mathcal{Q}$-instances $Q_1, \ldots, Q_r$, each of size m.*
Task: *Find a $\mathcal{P}$-instance (P, k) such that*
1. *k is polynomial in $m + \log(r)$, and*
2. *(P, k) has a solution $\Leftrightarrow$ Q_i has a solution for some i.*

With the following theorem, this gives a powerful tool for kernel lower bounds.

Theorem 3.3 (Bodlaender *et al.* [13])**.** *Let $\mathcal{Q}$ be a parameterized problem with parameter k. If $\mathcal{Q}$ is cross-composed by an NP-hard problem $\mathcal{P}$, then $\mathcal{Q}$ has no polynomial kernel with respect to k, unless the polynomial hierarchy collapses to its third level.*

This opens up for many possibilities, but for the *K-Cycle* problem no natural candidate problem $\mathcal{Q}$ to cross-compose could be identified.

3.4.2 An or-compositional generalization

While the *K-Cycle* problem seems to have no immediate or-composition for the parameter k, the situation is different for a slight generalization of the problem, which we can solve within a similar time bound.

Monochrome K-Cycle

Input: An simple, edge-colored graph $G = (V, E)$, with n vertices and m edges, and $K \subseteq V$.
Task: Decide whether G contains a *monochrome* K-cycle, i.e. a K-cycle of edges from a single color class.

Note that *K-Cycle* is just the restriction to edge-monochrome graphs, and that we can solve *Monochrome K-Cycle* by applying the described *K-Cycle*-algorithm to each subgraph induced by K and the edges of a single color class. As the number of color classes is bounded by $m \in O(n^2)$, as is the size of each class, we are still within the time bound $2^k n^{O(1)}$. As a consequence, we have:

Proposition 3.2. *The problem* Monochrome K-Cycle*, parameterized by the number of specified vertices k, has no polynomial kernel unless the polynomial hierarchy collapses to its third level.*

Proof. As a generalization of the *Hamiltonian Cycle* problem, the problem is NP-hard, and thus NP-complete as solutions can be checked in polynomial time. The above time bound $2^k n^{O(1)}$ shows that it is fixed-parameter tractable with respect to k. To see that it is also or-compositional, consider r given instances (G_1, K_1), $\ldots$, (G_r, K_r) with $|K_i| = k$ for all i. Let C_i be the set of colors used by the

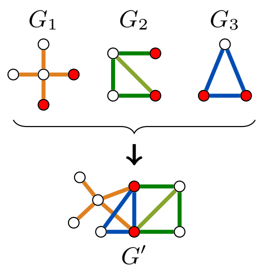


Figure 3.3: Composition for *Monochrome K-Cycle*.

edges in G_i. We can assume w.l.o.g. that $C_i \cap C_j = \emptyset$ for all $i \neq j$. Now form a superinstance (G', K') using colors $C_1 \cup C_2 \cup \cdots \cup C_r$ by simply setting $K' = K_1 = K_2 = \cdots = K_r$, i.e. taking the disjoint union of all input graphs and making them connected by arbitrarily identifying the set of specified vertices; see Figure 3.3. Then (G', K') has a monochrome K'-cycle if, and only if, any instance (G_i, K_i) has a monochrome K_i-cycle. □

Note, however, that this result has no bearing on the situation for the *K-Cycle* problem; given a polynomial kernel for *K-Cycle*, the best way to use this for an instance (G, K) of *Monochrome K-Cycle* would be to apply the kernel to each subgraph induced by K and a single color class. This would yield a 'polynomial kernel' for *Monochrome K-Cycle* consisting of $O(m)$ instances, each of size polynomial in k, that could be considered separately—but the above result does not exclude the possibility of such a so-called *Turing kernel*. For example, the problem *Clique* parameterized by maximum degree Δ admits no polynomial kernel unless the polynomial hierarchy collapses to its third level, but by requiring any single vertex to be in the clique, the problem has a trivial kernel of size at most Δ, so *Clique* parameterized by Δ has a Turing kernel of size $n\Delta$. There is, as of yet, no known method for excluding Turing kernels.

3.4.3 Kernelization lower bounds via reduction

When no direct composition can be found for a problem, an option is to look for a *polynomial parameter transformation* (PPT) from some other problem for which a polynomial kernel has been excluded. This type of reduction was introduced by Bodlaender *et al.* in [15]; it is a polynomial-time mapping reduction from one parameterized problem to another, such that the parameter increases at most polynomially.

From *Disjoint Factors*. In [15], a polynomial kernel for the *Disjoint Paths* problem, parameterized by number of terminals, is excluded by PPT-reduction from the following problem,

Disjoint Factors

Input:	A finite string s of digits from $\{1, 2, \ldots, k\}$.
Task:	Decide whether there are k disjoint substrings $s_1, \ldots, s_k$ of s such that s_i starts and ends with the digit i.

A polynomial kernel in terms of k for *Disjoint Factors* was excluded via direct or-composition in [15], under the assumption that the polynomial hierarchy does no collapse to its third level. Given the connection in Proposition 3.1 between the *Disjoint Paths* problem and the *K-Cycle* problem, and the fact that both problems are fixed-parameter tractable with respect to the size of a set of specified elements, it seemed a reasonable idea to try to modify the given PPT-reduction from the *Disjoint Factors* problem. Again, these attempts were not successful. It seems the constraint that some vertices are visited in a certain order, as required for a *Disjoint Paths* solution, would be necessary for a reduction from *Disjoint Factors*.

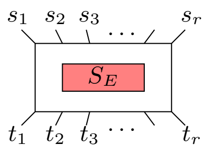
From *Hitting Set*. The next idea for a starting point was some version of the following problem, parameterized by m,

(Exact) Hitting Set

Input:	A hypergraph $\mathcal{H}$ of m subsets of a finite set V.
Task:	Decide whether there is a subset $S \subset V$ such that, for all $E \in \mathcal{H}$, $\lvert S \cap E \rvert \neq 0$. ($\lvert S \cap E \rvert = 1$ for exact version.)

A polynomial kernel for m was excluded in [31] (general version), [70] (exact version), under assumption that the polynomial hierarchy collapses to its third level. This problem seemed like an interesting candidate, as for example it provided a successful starting point for the *Steiner Tree* problem with the number of terminals as part of the parameter [31]. Although this problem is quite different from the *K-Cycle* problem, it shares this type of 'specified subset'-parameter.

A connection to the *K-Cycle* problem is provided by the standard mapping reduction from *Vertex Cover* to *Hamiltonian Cycle* in [44, Section 3.1.4]. As a vertex cover is just hitting set for a graph, and the *Hamiltonian Cycle* problem is a special case of *K-Cycle*, one idea was to find a PPT-reduction from *(Exact) Hitting Set* parameterized by m, to *K-Cycle* parameterized by k, inspired by this classical reduction, so that each set $E \in \mathcal{H}$ would correspond to some subset $S_E \subset K$ of constant size. However, any natural such attempt seemed to call for gadgets of the following type, for $r \leq n$,


s_1 s_2 s_3 $\cdots$ s_r
S_E
t_1 t_2 t_3 $\cdots$ t_r

If a K-cycle enters at the vertex s_i, it must pass through some/all of S_E, and then leave at the vertex t_i.

Given this type of gadget, for general r, one could easily construct an or-composition for *K-Cycle*. Thus this approach proved no easier than looking for an or-composition directly.

3.4.4 A reduction idea

The only idea the author encountered in the direction of a polynomial kernel, is the following reduction rule due to Daniel Lokshtanov [78]. It applies to 'very K-disconnected' instances, in the sense that the input graph G has a vertex subset $K' \supseteq K$ whose removal splits G into $r > \lvert K' \rvert^2$ connected components C_i, demonstrated in Figure 3.4. We can then construct a bipartite graph $H = (A \cup B, E)$ as follows:

1. A has one vertex a_{uv} for every ordered pair (u, v) of (not necessarily distinct) vertices in K',
2. B has one vertex b_i for every connected component C_i of $G \setminus K'$, and
3. $a_{uv}b_i \in E$ if, and only if, G has a path from u to v with all interior vertices in C_i.

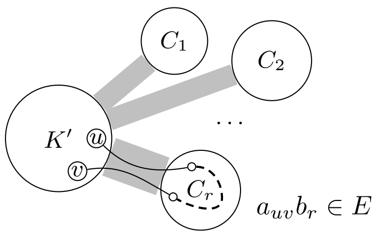


Figure 3.4: Reduction idea.

As $|B| > |A|$ by assumption, the graph H must contain a so called *crown*, consisting of subsets $B' \subseteq B$ and $A' \subseteq A$ with $|A'| < |B'|$, such that A' is the combined neighborhood of every vertex in B', and such that there is a matching $M \subset E$ from A' into B'. If $b_i \in B'$ is not matched by M, it can be shown that $G \setminus C_i$ has a K-cycle if, and only if, G has a K-cycle. Thus any such unmatched component can be removed in G.

There is no guarantee here that the remaining graph will be of size polynomial in k in general, but the observation is still interesting. In view of recent developments, it might be of use for improving the newly found polynomial kernel.

Part II

Exponential-Time Complexity in the Tutte Plane

Chapter 4

The Tutte Polynomial

As a young student at Cambridge, William T. Tutte was fascinated by a certain recursion formula that many graph-related counting problems seemed to satisfy. If $f(G)$ is the solution to some counting problem regarding a multigraph $G = (V, E)$, the recursion is

$$f(G) = f(G \setminus e) + f(G/e)\,, \tag{4.1}$$

for any edge $e \in E$ that is not a loop or a cut-edge, where $G \setminus e$ arises from G by *deleting* the edge e, for example like $\mapsto$, and G/e arises from G by *contracting* the edge e, for example like $\mapsto$, that is, deleting the edge e and identifying its endpoints (so any remaining edges between these two endpoints become loops). For this reason such a recursion is called a *deletion-contraction identity*. For example, the number $C(G)$ of spanning trees of G satisfies (4.1), and if we multiply the number $\chi(G;k)$ of proper k-colorings of G by $(-1)^{|V|}$, and the number $\phi(G;k)$ of k-flows in G by $(-1)^{|V|+|E|}$, then both of the quantities obtained also satisfy (4.1). Tutte used the term 'W-function' to describe any such graph quantity satisfying a deletion-contraction identity.

The number $\chi(G;k)$ of k-colorings of G can be seen as the evaluation at k of the *chromatic polynomial*, $x \mapsto \chi(G;x)$, and similarly the number $\phi(G;k)$ of k-flows in G is the evaluation at k of the *flow polynomial*, $y \mapsto \phi(G;y)$. These two famous graph polynomials had already been studied for some time, when in 1953 Tutte made a discovery [92]. In [93] he recalls:

> *"Playing with my W-functions I obtained a two-variable polynomial from which either the chromatic polynomial or the flow-polynomial could be obtained by setting one of the variables equal to zero, and adjusting signs."*

This two-variable polynomial he so playfully arrived at today goes under the name of the *Tutte polynomial* and is the subject of quite serious discussions. For a multigraph $G = (V, E)$ it looks like this:

$$T(G;x,y) = \sum_{A \subseteq E} (x-1)^{\kappa(A)-\kappa(E)} (y-1)^{\kappa(A)+|A|-|V|}\,, \tag{4.2}$$

where $\kappa(A)$ denotes the number of connected components of the subgraph (V, A). It is easily checked that the exponents of the sum must all be nonnegative,[1] so that this is indeed a well-defined polynomial.

As Tutte wrote, we get the chromatic polynomial from the restriction $y = 0$, as

$$\chi(G; x) = (-1)^{|V|-\kappa(E)} x^{\kappa(E)} T(G; 1 - x, 0)\,, \tag{4.3}$$

and the flow polynomial is given by $\phi(G, y) = (-1)^{|E|-|V|+1} T(G, 0, 1 - y)$. It turns out that the value of $T(G; x, y)$ gives interesting combinatorial interpretations for several other points (x, y) in the so called *Tutte plane* as well. For example, the number of spanning trees in G is given by $T(G; 1, 1)$, and the number of forests is given by $T(G; 2, 1)$. Also, except the lines $y = 0$ and $x = 0$ already mentioned, there are other lines and curves where the Tutte polynomial specializes to well-known graph polynomials. We will have a closer look at several examples.

All of the above mentioned graph invariants satisfy (4.1) with proper adjustment of signs, just as for $\chi(G; k)$ and $\phi(G; k)$. In fact, the Tutte polynomial encodes *any* graph invariant that is multiplicative over components and satisfies (4.1) up to signs [82]. As many intensively studied graph parameters are of this kind, that explains the interest and importance of any result concerning this monster of a polynomial.

4.1 The Potts model and the multivariate Tutte polynomial

While Tutte was playing with his graph polynomials, Renfrey Potts studied models in statistical mechanics. In 1952 he introduced what is now known as the partition function of the *q-state Potts model*, which was to become one of the most well-studied concepts of the field. This is a model for systems of interacting particles, each of which can exist in one of q possible states. We can view such a system as a graph $G = (V, E)$, where V is the set of particles and E is the set of interacting particle pairs. Typically, G has the form of some lattice. A certain interaction-energy J is associated with any two interacting particles in the same state, and for a given configuration $\sigma : V \to [q]$ of states over G the corresponding *interaction Hamiltonian* is

$$H_\sigma = J \sum_{uv \in E} \delta_{\sigma(u),\sigma(v)}\,, \tag{4.4}$$

where $\delta_{i,j}$ is the Kronecker delta. The *partition function* $Z_q(G)$ of the q-state Potts model is defined as

$$Z_q(G) = \sum_{\sigma: V \to [q]} e^{-H_\sigma/(kT)} \tag{4.5}$$

where k is Boltzmann's constant and T is the temperature. This quantity is interesting because it allows us to compute the probability of finding the system in a certain configuration of states σ, for given temperature and interaction-energy, like so:

$$\Pr(\sigma) = e^{-H_\sigma/(kT)}/Z_q(G)\,.$$

[1]The number $\kappa(A) + |A| - |V|$ is the *nullity* of the graph (V, A).

We can view the partition function as the evaluation at $w = e^{-J/(kT)}$ of the *q-state Potts polynomial*

$$Z_q(G; w) = \sum_{\sigma: V \to [q]} (1 + w)^{\mathrm{mono}(\sigma, G)} , \tag{4.6}$$

where $\mathrm{mono}(\sigma, G)$ is defined as the number of edges $e = uv$ of G that are 'monochromatic' under σ, i.e. whose corresponding two interacting particles u and v both receive the same state $\sigma_u = \sigma_v$.

4.1.1 Relation to the Tutte polynomial

Consider (4.6), and let A_σ be the set of all monochromatic edges under σ. Note that for a given edge set A, the number of assignments σ such that $A \subseteq A_\sigma$ is determined by the number $\kappa(A)$ of connected components in the graph (V, A), as all vertices of such a component should obtain the same state by σ. Using this fact, and some standard manipulations, we can rewrite (4.6) as

$$\begin{aligned} Z_q(G; w) &= \sum_{\sigma: V \to [q]} (1 + w)^{|A_\sigma|} \\ &= \sum_{\sigma: V \to [q]} \sum_{A \subseteq A_\sigma} w^{|A|} = \sum_{A \subseteq E} \sum_{\substack{\sigma: V \to [q] \\ A \subseteq A_\sigma}} w^{|A|} \\ &= \sum_{A \subseteq E} q^{\kappa(A)} w^{|A|} . \end{aligned} \tag{4.7}$$

The subgraph expansion we arrive at is the *Fortuin-Kasteleyn representation* of the Potts polynomial. The nice thing about this formulation is that it allows a natural continuation of the parameter q beyond natural numbers, which is not possible for the standard representation of (4.6). This continuation gives a bivariate polynomial

$$Z(G; q, w) = \sum_{A \subseteq E} q^{\kappa(A)} w^{|A|} , \tag{4.8}$$

such that $Z(G; q, w) = Z_q(G; w)$ for $q \in \mathbf{N}^+$. This is none other than the Tutte polynomial in disguise; with $w = y - 1$ and $q = (x - 1)(y - 1)$ we have

$$Z(G; q, w) = (x - 1)^{k(E)} (y - 1)^{|V|} T(G; x, y) . \tag{4.9}$$

This important link between the Potts model and the Tutte polynomial was discovered by Fortuin and Kasteleyn in the late 1960's [39, 40].

4.1.2 The multivariate Tutte polynomial

The above relation to the Potts model provides an interpretation of the Tutte polynomial's second variable y as an edge weight carried by all edges of the graph G. We can extend this to the case of general edge weights, given by a function $\mathbf{w}\colon E \to \mathbf{Q}$, as

$$Z(G;q,\mathbf{w}) = \sum_{A\subseteq E} q^{k(A)} \prod_{e\in A} \mathbf{w}(e)\,. \tag{4.10}$$

This is what Sokal [89] calls the *multivariate Tutte polynomial*, and it often provides great technical simplification for analysis of the Tutte polynomial, even though we are ultimately interested in the single-valued case. If $\mathbf{w}(e) = w$ for all $e \in E$, we recover $Z(G;q,w)$.

4.2 Algorithms

Suppose we are given a connected[2] graph G with n vertices and m edges, and that we want to compute $T(G;x,y)$ for some given point (x,y). Direct evaluation of the polynomial as given in (4.2) takes $2^m m^{O(1)}$ arithmetic operations. The standard approach usually does better, making use of the fact that the Tutte polynomial satisfies the following deletion-contraction identity,

$$T(G;x,y) = \begin{cases} 1 & \text{if } G \text{ has no edges;}\\ yT(G\setminus e;x,y) & \text{if } e \text{ is a loop;}\\ xT(G/e;x,y) & \text{if } e \text{ is a cut-edge;}\\ T(G\setminus e;x,y) + T(G/e;x,y) & \text{if } e \text{ is ordinary.}\end{cases}$$

where an *ordinary* edge is an edge that is neither a loop nor a cut-edge. The recursive algorithm based on this formula has a running time of $1.6180^{n+m} m^{O(1)}$ [96], which beats direct evaluation for any nonsparse graph. Until recently, this was the best known upper bound for computing the Tutte polynomial.

In practice it is often possible to speed up this algorithm, by coupling it with some isomorphism-test of the intermediate graphs at each level in the recursion tree. As the Tutte polynomials of isomorphic graphs are identical, detecting such isomorphic graphs—the earlier the better—may avoid much redundant computation [52].

4.2.1 A vertex-exponential algorithm

It was a breakthrough when Björklund *et al.* [7] showed how $T(G;x,y)$ could be computed in time $2^n m^{O(1)}$, i.e. no longer exponential in the number of edges. This algorithm employs the multivariate formulation of the Tutte polynomial defined in (4.10). In essence, it evaluates, for $q = 1, 2, \ldots n+1$, the q-state Potts polynomial $Z_q(G;w)$ at the fixed value $w = y-1$ in time $2^n m^{O(1)}$ using the principle of inclusion and exclusion. By (4.7) this provides $n+1$ distinct evaluations of the nth degree polynomial $q \mapsto Z(G;q,y-1)$, whose coefficients can then be found in polynomial time by Lagrange interpolation. By (4.9), an evaluation of this polynomial at $q = (x-1)(y-1)$ reveals the value of $T(G;x,y)$.

[2]Unconnected graphs are handled by Theorem 4.1 in Section 4.4.1.

We especially observe two techniques used in this algorithm that are also of great for establishing lower bounds for the complexity of evaluating the Tutte polynomial.

- The formulation in terms of the multivariate Tutte polynomial $Z(G; q, w)$.
- Lagrange interpolation.

We will make extensive use of both in the following chapters.

4.2.2 Obtaining coefficients

We have discussed exponential-time algorithms for evaluating the Tutte polynomial at a fixed point (x, y). In fact, any such algorithm can be used to compute the full polynomial, i.e. the coefficients $t_{i,j}$ such that

$$T(G; x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m} t_{i,j} x^i y^j , \tag{4.11}$$

in time that is a polynomial factor of a single evaluation, as follows:

For a fixed number x, use the given algorithm to compute $T(G; x, y)$ for $m + 1$ distinct values of y. This gives $m + 1$ distinct evaluations of the mth degree polynomial $y \mapsto T(G; x, y)$, and we get the coefficients of this polynomial by Lagrange interpolation. The jth such coefficient can be written as $c_j(x) = \sum_{i=0}^{m} t_{i,j} x^i$. Repeat the procedure for $n+1$ distinct values of x. This gives $n+1$ distinct evaluations of the nth degree polynomial $x \mapsto c_j(x)$, for every j, and a second round of Lagrange interpolation gives the coefficients $t_{i,j}$—all within the exponential-time bound of a single evaluation.

4.3 Complexity

Despite their unified definition (4.2), the problems encoded as evaluations of the Tutte polynomial seem to differ widely in computational complexity across the so called *Tutte plane*. For example, $T(G; 1, 1)$ gives the number of spanning trees in G, which can be computed in polynomial time using Kirchhoff's matrix-tree theorem [65], while the best known algorithm for computing $T(G; 2, 1)$, the number of spanning forests, runs in vertex-exponential time [7]. Thus the following (x, y)-parameterized problem has received considerable interest over the years.

Tutte(x, y)

Input:	A multigraph G with n vertices and m edges.
Task:	Compute $T(G; x, y)$.

For complexity analysis of this problem we need to specify a domain for the parameters x and y, such that standard polynomial-time reductions are not destroyed by heavy algebraic manipulations in the computation of $T(G; x, y)$. For ease of presentation we will usually assume that (x, y) is a point with rational coordinates, and these are also the points of interest for most classical combinatorial interpretations. In fact, any algebraic point would do, as for any fixed such point we can compute symbolically in a finite degree extension of $\mathbf{Q}$.[3] This is sometimes useful even if we are ultimately interested in a rational point. Thus we define the Tutte plane as the set of all algebraic points $(x, y) \in \mathbf{R}^2$.

4.3.1 Polynomial-time complexity

In 1990, Jaeger *et al.* [59] presented a complete map of the complexity across the Tutte plane. It was already known that *Tutte*(x, y) is polynomial-time solvable at any point on the hyperbola $(x-1)(y-1) = 1$ and the four points $(-1, -1)$, $(0, -1)$, $(-1, 0)$, and $(1, 1)$. Jaeger *et al.* showed that except for these *trivial points*, the problem *Tutte*(x, y) is #P-hard everywhere. Thus, except for trivial points (x, y), there can be no polynomial-time algorithm for *Tutte*(x, y) unless #P $\subseteq$ FP. Until recently, this was the only notion of hardness in the Tutte plane.

4.3.2 Exponential-time complexity

Given the vertex-exponential upper bound by Björklund *et al.* [7], one may ask whether this exponential factor could be further improved to something subexponential, such as $\exp(O(\sqrt{n}))$, which, as we will see in Chapter 6, is actually the case for the planar restriction. This question was investigated by Dell *et al.* [30] under the following counting analogue of the Exponential Time Hypothesis:

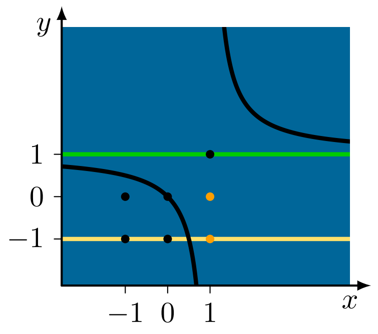
> *Counting Exponential Time Hypothesis* (#ETH)
>
> There is a real number $c > 0$ such that no deterministic algorithm can solve *#3-Sat* in time 2^{cn}.

The currently best bound for *#3-Sat* is $O(1.6423^n)$ [71].

Under #ETH, it was shown in [30] that for most of the #P-hard points (x, y) in the Tutte plane, there is a lower bound of $\exp(\Omega(n))$ for the problem *Tutte*(x, y) to match the upper bound of Björklund *et al.*; see Figure 4.1 for an overview. Thus there is little hope for a subexponential algorithm for general points of the Tutte plane. For the line $y = -1$, the lower bound from [30] follows from the analysis of the restriction to *simple graphs*, a problem we will discuss further in Chapter 5, and as a consequence the bound is slightly subexponential, falling a polylogarithmic factor short of n in the exponent.

The line $y = 1$ and the points $(1, 0)$ and $(1, -1)$ were left open in [30]. Subexponential (but superpolynomial) lower bounds for these points will follow as consequences of results in the following two chapters, where we analyze the restriction to simple- and planar graphs, respectively. Hence, we now have concrete superpolynomial lower bounds for every point of the Tutte plane.

[3]See discussions in [59, p.41] and [95, p.693] for treatment of individual irrationals. A more general approach is presented in [10].



🟦 $\exp(\Omega(n))$ [30], [29]
🟧 $\exp(\Omega(n/\log^2 n))$ [29] (simple graphs, Chapter 5)
🟨 $\exp(\Omega(n/\log^3 n))$ [30, 29] (simple graphs)
🟩 $\exp(\Omega(\sqrt[8]{n})$ (planar graphs, Chapter 6)
⬛ $n^{O(1)}$ [59]

Figure 4.1: Exponential time complexity under #ETH of the Tutte plane for multigraphs in terms of n, the number of vertices. All bounds hold in particular for graphs with $m \in n^{O(1)}$ edges. The black hyperbola $(x-1)(y-1) = 1$ and the four points close to the origin are in P.

Remark 1. The number m of edges in a multigraph may be exponential in the number n of vertices, which means that the lower bound of $\exp(\Omega(n))$ as it is stated in [30, 29] does not immediately rule out even an edge-polynomial algorithm. However, it is easily checked from the proof of this result that the hardest instances have $m \in O(n^4)$ edges.

Remark 2. As a curiosity, we note that whenever a superpolynomial lower bound is given for some point (x_0, y_0) in the Tutte plane, then there can only be a finite number of 'exponentially easier' points on any curve through (x_0, y_0) with a polynomial parameterization. More precisely, if a lower bound exponential in $\Omega(f(n))$ is given for (x_0, y_0), for some $f(n) \leq n$, then on each polynomial-parameterized curve that contains (x_0, y_0) there can only be a finite number of points solvable in time exponential in $o(f(n))$. To see this, consider a given such curve $\mathcal{C} : s \mapsto (x(s), y(s))$ where $x(s)$ and $y(s)$ are polynomials of degree d. Then, for graphs G of n vertices and m edges, the function $s \mapsto T(G; x(s), y(s))$ would be a polynomial of degree d^2nm. Now suppose there were an infinite number of exponentially easier points on $\mathcal{C}$. Then we could pick $d^2nm + 1$ such points $(x(s_i), y(s_i))$, for each of them compute $T(G; x(s_i), y(s_i))$, and then obtain the coefficients of $s \mapsto T(G; x(s), y(s))$ by interpolation. This, in turn, would give us the value of $T(G; x_0, y_0)$ faster than the given lower bound—a contradiction.

4.3.3 Complexity of approximation and sign

Goldberg and Jerrum [48] studied the problem of computing an approximate value of $T(G; x, y)$. The notion of approximation algorithm they consider is that of a *fully polynomial randomized approximation scheme* (FPRAS): a polynomial-time algorithm that with probability at least $3/4$ outputs an approximation arbitrarily close to the true value. They show that there can be no FPRAS for the problem *Tutte*(x, y) a.e. on the half-planes $x < -1$ and $y < -1$, unless RP = NP. The status for remaining points with $x, y > -1$ remains open, except at the known trivial points and the upper branch of the hyperbola $(x-1)(x-2) = 2$, for which a FPRAS follows from [60].

Very recently, these results were sharpened as a result of Goldberg and Jerrum's work on the complexity of computing the *sign* of $T(G; x, y)$. This seems 'almost like a decision problem', as there are only three possible outcomes: positive, negative, or zero, but in [50] they show that, for large regions of the plane, this problem is in fact #P-hard. As a consequence, approximation at these points is also #P-hard. Thus, in terms of polynomial-time complexity, approximation is as hard as exact computation for large parts of the Tutte plane. For most remaining points, including all those with $x, y > -1$ and $x + y > 0$, it is shown that the sign can be computed in polynomial time, and that an approximation can be computed in polynomial time using an NP oracle. This marks a clear distinction between these problems and exact computation of *Tutte(x, y)*, where we have #P-hardness a.e.

4.3.4 The line $y = 1$

The line $y = 1$ in the Tutte plane deserves some attention, because, as Figure 4.1 shows, the current lower bound under #ETH for these points is 'very sub-exponential'. Here the Tutte polynomial specializes to the generating function of forests in G, weighted by the number of edges it contains,

$$F(G; w) = \sum_{\substack{F \subseteq E \\ (V,F) \text{ forest}}} w^{|F|} . \tag{4.12}$$

The relation to the Tutte polynomial is $F(G; w) = w^{|V| - \kappa(E)} T(G; 1 + 1/w, 1)$.

This line resist the analysis of [48, 50, 30, 29], and, unlike all other nontrivial points of the Tutte plane, the only proof of #P-hardness at these points relies on the #P-hardness for the restricted class of *planar* graphs, for which there is a direct proof by Vertigan [95]. As the problem *Tutte(x, y)* restricted to planar graphs has a known upper bound exponential in $O(\sqrt{n})$ [87], this approach could at best give a lower bound exponential in $\Omega(\sqrt{n})$ under #ETH. However, not even such a result has been found for this line.

In Chapter 6 we note that the planar reduction of [95] yields a lower bound exponential in $\Omega(\sqrt[8]{n})$ under #ETH for planar multigraphs. This is by reduction from the problem of computing the coefficients of $w \mapsto F(G; w)$, which has the same lower bound under #ETH. The same reduction can be used directly for general multigraphs, but there is no immediate gain in this approach, as no better bound for finding coefficients of $w \mapsto F(G; w)$ for general multigraphs has yet been found.

4.4 Some useful properties

We here list some elementary properties of the Tutte polynomial that will be useful in the following chapters.

4.4.1 Deletion-contraction and connectedness

The deletion-contraction identity in (4.13) is sometimes taken as the definition of the Tutte polynomial. We restate it here for easy reference.

$$T(G;x,y) = \begin{cases} 1 & \text{if } G \text{ has no edges;} \\ yT(G \setminus e;x,y) & \text{if } e \text{ is a loop;} \\ xT(G/e;x,y) & \text{if } e \text{ is a cut-edge;} \\ T(G \setminus e;x,y) + T(G/e;x,y) & \text{if } e \text{ is ordinary.} \end{cases} \tag{4.13}$$

This formula can be obtained by routine calculations from the subgraph expansion (4.2), considering the effects of a deletion or contraction on the number of components, vertices, and edges, depending on the type of edge that is manipulated.

The subgraph expansion for the multivariate Tutte polynomial in (4.10) has a less complicated dependency on the number of components than that of the standard Tutte polynomial, and the resulting deletion-contraction identity takes the same form regardless of the type of edge under consideration. It follows easily from (4.10) that

$$Z(G;q,\mathbf{w}) = Z(G \setminus e;q,\mathbf{w}) + \mathbf{w}(e)Z(G/e;q,\mathbf{w})\,. \tag{4.14}$$

Using (4.13), it follows by induction on the number of ordinary edges that the Tutte polynomial is multiplicative over 2-connected components:

Theorem 4.1. *Let G be the union of two graphs G_1 and G_2 with at most one vertex in common. Then*

$$T(G;x,y) = T(G_1;x,y) \cdot T(G_2;x,y)\,. \tag{4.15}$$

This means that for exponential-time analysis of *Tutte(x,y)*, we can always assume that the input graph is 2-connected without loops.

4.4.2 Point shifts, thickenings and stretches

Given two points (x,y) and (x',y') in the Tutte plane, there are always some graphs G' and G such that $T(G';x,y) = T(G;x',y')$. For example,

$T($ $;x,y) = T($ $;0,x^3) = x^3\,.$

Sometimes it is possible to define a mapping φ, such that for any graph G

$$T(\varphi(G);x,y) \sim T(G;x',y')\,,$$

where $\sim$ denotes equality up to some simple closed-form expression. We then say that the point (x,y) is *shifted* to the point (x',y') via the transformation φ. Such a shift provides a reduction from *Tutte(x',y')* to *Tutte(x,y)*. This was one of the main techniques used by Jaeger *et al.* in [59] for studying #P-hardness across the Tutte plane, and it is also a crucial tool in the analysis of exponential-time complexity. Note that the strength of such a reduction depends on the size of $\varphi(G)$. If, for example, the number of nodes increases quadratically, then a lower bound of $\exp(\Omega(n))$ for *Tutte(x,y)* only implies a lower bound of $\exp(\Omega(\sqrt{n}))$ for *Tutte(x',y')*.

Thickenings and stretches. The most fundamental point-shift transformations are so called *thickenings* and *stretches*.

Definition 4.1. *The* k-stretch *of a graph* G*, denoted* G^k*, is the graph obtained by subdividing every edge of* G *into a* k*-path, and the* k-thickening *of a graph* G*, denoted* G_k*, is the graph obtained by replacing every edge in* G *by a bundle of* k *edges.*

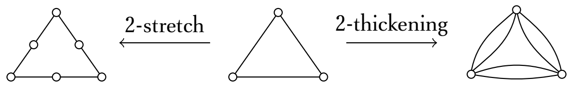


Figure 4.2: Definition of thickenings and stretches.

Note that already a 2-stretch of a graph with n^2 edges increases the number of vertices quadratically, so we must use such transformations with care in our analysis. We have the following point shift identities (see e.g. [48, Section 3]).

Theorem 4.2. *Let* $G = (V, E)$ *be a graph, and let* $q = (x-1)(y-1)$*. Then*

$$T(G^k; x, y) = \left(\tfrac{x^k-1}{x-1}\right)^{\eta(G)} T(G; x^k, \tfrac{q}{x^k-1} + 1) \tag{4.16}$$

where $\eta(G) = |E| + \kappa(E) - |V|$*, and*

$$T(G_k; x, y) = \left(\tfrac{y^k-1}{y-1}\right)^{r(G)} T(G; \tfrac{q}{y^k-1} + 1, y^k) \tag{4.17}$$

where $r(G) = |V| - \kappa(E)$*.*

We will also need corresponding identities in the formulation of the multivariate Tutte polynomial. These can be derived from the identities in Theorem 4.2, using the relationship (4.9) with $w = y - 1$.

Theorem 4.3. *Let* $G = (V, E)$ *be a graph. Then*

$$Z(G^k; q, w) = \tfrac{(q/w)^k-1}{q/w-1} \cdot Z(G; q, w') \,, \tag{4.18}$$

where w' *satisfies* $(1 + q/w') = (1 + q/w)^k$*, and*

$$Z(G_k; q, w) = Z(G; q, (1+w)^k - 1) \,. \tag{4.19}$$

Hyperbolas. Note that, for both stretches and thickenings, the value of $q = (x-1)(y-1)$ is preserved, so these transformations give point shifts along fixed *hyperbolas* in the Tutte plane. This gives such hyperbolas a special status, as we will see in the following two chapters. We let H_q denote the hyperbola $(x-1)(y-1) = q$ in the Tutte plane.

Chapter 5

Lower Bounds for Simple Graphs

In this chapter we consider the restriction of *Tutte*(x, y) to *simple* input graphs.

Simple Tutte(x, y)

Input:	A simple graph G with n vertices and m edges.
Task:	Compute $T(G; x, y)$.

This restriction is of particular interest because most of the graphs studied in statistical mechanics arise from bond structures that are simple. From the perspective of polynomial-time complexity, the picture of the Tutte plane for simple graphs coincides with the general case, i.e. *Simple Tutte*(x, y) is #P-hard for all points (x, y) except for the hyperbola H_1 and the points $(-1, -1)$, $(0, -1)$, $(-1, 0)$, and $(1, 1)$, where it is polynomial-time solvable. The study of exponential-time complexity under #ETH was initiated by Dell *et al.* in [30], where lower bounds exponential in n were given for the line $y = 0$, and exponential in $m/\log^3(m)$ for most other points of the Tutte plane. These exponential bounds in terms of edge size, together with the known $2^n m^{O(1)}$-time upper bound, show that the hardest instances for such points are *sparse* graphs.

A few lines were left open in [30], in particular the line $x = 1$ which requires a somewhat different treatment and will be the focus of this chapter. This so called *reliability line* was the subject of [56], which is joint work with Thore Husfeldt and contains a lower bound exponential in $m/\log^2(m)$ for all points of the line with $y > 1$. Together with Holger Dell, the result was extended to the full line $x = 1$. This work is incorporated into the journal paper [29], which now gives exponential (at least within a polylogarithmic factor) lower bounds under #ETH for the whole Tutte plane, except the line $y = 1$; see Figure 5.1 for an overview.

5.1 The reliability polynomial

On the line $x = 1$, the Tutte polynomial specializes to the *reliability polynomial*, an object studied in algebraic graph theory [47, Section 15.8]. For a connected

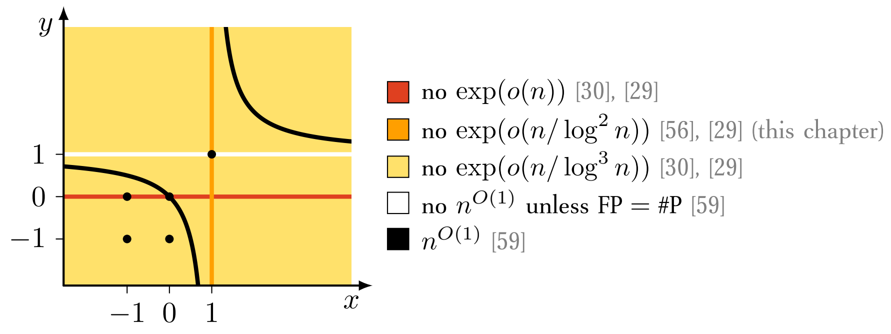


Figure 5.1: Exponential time complexity under #ETH of the Tutte plane for simple graphs in terms of n, the number of vertices. For white points we only have the #P-hardness of [59].

graph $G = (V, E)$, the reliability polynomial is given by

$$R(G;p) = \sum_{\substack{A\subseteq E\\ \kappa(A)=1}} p^{|E\setminus A|}(1-p)^{|A|}\,, \tag{5.1}$$

where $\kappa(A)$ denotes the number of connected components in the subgraph (V, A). We have the following connection to the Tutte polynomial, provided $p \neq 0$,

$$R(G;p) = p^{|E|-|V|+1}(1-p)^{|V|-1}T(G;1,1/p)\,. \tag{5.2}$$

For rational $p \in [0,1]$, the value of $R(G;p)$ gives the probability that G stays connected if every edge is independently removed with probability p. For example $R(\;;\frac{1}{3}) = Pr(\;)+5Pr(\;) = (\frac{2}{3})^5+5\cdot\frac{1}{3}\cdot(\frac{2}{3})^4 = \frac{112}{243}$. This gives a simple model of connectedness in networks that are subject to random failure of communication channels, a situation naturally occurring in many areas; see [2] for an extensive survey of applications.

Upper bounds. Just as for general evaluation of the Tutte polynomial, computing graph reliability directly from its definition (5.1) takes $2^m m^{O(1)}$ operations, where m is the number of edges in G. A vertex-exponential upper bound for the problem was found already in 1980, when Buzacott [18] gave an algorithm with a running time of $3^n m^{O(1)}$. The current best upper bound is the $2^n m^{O(1)}$-time algorithm of Björklund *et al.* [7], which computes $T(G;x,y)$ for any rational (x,y).

Lower bounds. For the general case of multigraphs, a lower bound exponential in $\Omega(n)$ under #ETH follows from [30]. It would be even more interesting to know how hard the problem is for the classical restriction to *simple* graphs. To investigate this, we study the exponential-time complexity of *Simple Tutte*$(1,y)$. It would not be unreasonable to hope for a subexponential algorithm for this problem, as for example the corresponding restriction to planar graph allows an algorithm that is exponential only in $O(\sqrt{n})$ for any rational point (x,y) [87].

We have $R(G;0) = 1$ and $R(G;1) = 0$ for all connected graphs, so for $p = 0$ and $p = 1$ the reliability polynomial is always easy to evaluate. One may ask what the situation is for values close to these two extremes. Note that $p = 1$ corresponds to the limit point $(1, 1)$ in the Tutte plane, which we know is also a polynomial-time point for the problem *Tutte*(x, y), though for less trivial reasons (Kirchhoff's matrix-tree theorem). Moreover, for $p = \frac{1}{2}$ it is easily seen that (5.1) equals the number of connected, spanning subgraphs of G, divided by 2^m. This is an interesting enumeration problem in itself, and one could be tempted to hope for an algorithm faster than vertex-exponential, considering that the related problem of finding the number of spanning *trees* of G can be solved in polynomial time (again, the point $(1, 1)$ in the Tutte plane).

5.1.1 Main theorem

We give a lower bound of the problem of computing all-terminal graph reliability for the class of simple graphs for all nontrivial probabilities p. Beyond this interpretation, the lower bound extends to all nontrivial points on the full line $x = 1$ in the Tutte plane. We use the framework introduced by Dell *et al.* in [30] for studying exponential-time complexity in the Tutte plane under the hypothesis #ETH.

Theorem 5.1. *Let $y \neq 1$ be a rational number. Then, under #ETH, there is no $\exp(o(m/\log^2 m))$-time algorithm for* Simple Tutte($1, y$).

In particular, the bound holds for $y = 2$, i.e. counting the number of connected spanning subgraphs of a given graph.

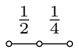
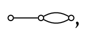
We have expressed the lower bound in terms of the parameter m, the number of edges of the input graph. Since $n \leq m + 1$ for connected graphs, the result implies a lower bound of $\exp(\Omega(n/\log^2 n))$ also in terms of the parameter n, the number of vertices of the input graph. The formulation in terms of m is interesting, because together with the vertex-exponential algorithm from [7] it shows that the hardest instances have roughly linear density, ruling out a better algorithm than $\exp(O(n/\log^2 n))$ even for the restricted case of *sparse* graphs.

Our bound does not quite match the upper bounds of [18, 7]. This situation is similar to the bounds reported in [30] for other regions of the Tutte plane, which also fall a few logarithmic factors (in the exponent) short of the best known algorithms. The bound does, however, suffice to separate the complexity of reliability computation for simple graphs from the subexponential-time bound for the planar case.

5.1.2 Related work

The structural complexity (in terms of #P-hardness) of *Simple Tutte*($1, y$) follows from the mentioned result of Jaeger *et al.* [59], which covered every point in the Tutte plane, but in fact the case of all-terminal graph reliability was settled by Provan and Ball already in the 1980's [84]. The reductions in these papers do not preserve the parameters n and m, so that the running time bounds under #ETH implicitly provided by their techniques are typically exponential in $\Omega(n^{1/k})$ for some k.

The problem *Simple Tutte*$(1,y)$ has a number of natural extensions.

1. We can consider the computational problem of finding the reliability polynomial itself, instead of its value at a fixed point $p = 1/y$. The input to this problem is a graph, and the output is a list of coefficients. For example, on input , the output should be $R($ $;p) = 4p^5 - 15p^4 + 20p^3 - 10p^2 + 1$.
2. We can associate individual probabilities to every edge. For example, the graph  becomes disconnected with probability $\frac{5}{8}$.
3. We can consider multigraphs like , but with the same edge-failure probability p. As indicated by the example (set $p = \frac{1}{2}$ and compare to the above), the multigraph case is a special case of the individually edge-weighted case, a fact that we will use later.

All of these problems are at least as hard as the problem under consideration in this chapter. Lower bounds of size $\exp(\Omega(m))$ are given in [30] or follow relatively easily; see Section 5.3.

A recent paper of Hoffman [54] studies the complexity of another graph polynomial, the independent set polynomial, in the same framework.

5.2 Tools adjusted for the reliability line

The line $x = 1$ in the Tutte plane is the vertical component of the hyperbola

$$H_0 = \{(x,y) \;:\; (x-1)(y-1) = 0\}\,.$$

This hyperbola resists the analysis of e.g. [30], because the the multivariate formulation typically assumes that $q = (x-1)(y-1) \neq 0$. Thus we will need some modified versions of the tools described in Chapter 4.

By the multiplicative property (4.15), we can restrict our attention to connected graphs $G = (V,E)$, i.e. we may assume that $\kappa(E) = 1$ as in the definition of the reliability polynomial.

5.2.1 Multivariate formulation

The multivariate Tutte polynomial $Z(G;q,\mathbf{w})$ defined in (4.10) evaluates to zero whenever $q = 0$, which is not very helpful. Instead we consider the following slightly modified polynomial introduced in Sokal [89, Section 2.3],

$$\begin{aligned} Z_0(G;q,\mathbf{w}) &= q^{-\kappa(E)} Z(G;q,\mathbf{w}) \\ &= \sum_{A\subseteq E} \prod_{e\in A} \mathbf{w}(e) q^{\kappa(A)-\kappa(E)}\,. \end{aligned} \tag{5.3}$$

For $\kappa(E) = 1$ and $q = 0$ this gives a weighted version of the reliability polynomial,

$$\hat{R}(G;\mathbf{w}) = \sum_{\substack{A\subseteq E\\ \kappa(A)=1}} \prod_{e\in A} \mathbf{w}(e)\,. \tag{5.4}$$

We get the following connection to the reliability polynomial, for $p \neq 0$,

$$R(G;p) = p^{|E|}\hat{R}(G; \tfrac{1}{p} - 1)\,. \tag{5.5}$$

5.2.2 Deletion-contraction identity

A deletion-contraction identity for Z_0 follows implicitly from (4.14). Since $Z_0 = q^{-k(E)}Z$, equation (4.14) gives

$$Z_0(G;q,\mathbf{w}) = \begin{cases} qZ_0(G-e;q,\mathbf{w}) + \mathbf{w}(e)Z_0(G/e;q,\mathbf{w}) & \text{if } e \text{ is a cut-edge,} \\ Z_0(G-e;q,\mathbf{w}) + \mathbf{w}(e)Z_0(G/e;q,\mathbf{w}) & \text{otherwise.} \end{cases} \tag{5.6}$$

5.2.3 Stretches and thickenings

Along the reliability line, the weight shift identities of Theorem 4.2 for stretches and thickenings are not valid. Goldberg and Jerrum derived corresponding versions for the polynomial $\hat{R}(G;\mathbf{w})$ in [48] for the special case when every edge of the inserted k-path or k-bundle has the same weight. We need multiweighted versions of these. By $\mathbf{w}[e \mapsto w']$ we denote the function $\mathbf{w}' : E(G) \to \mathbf{Q}$ that is identical to $\mathbf{w}$ except at the edge e where $\mathbf{w}'(e) = w'$.

Lemma 5.1. *Let G be a connected graph with edge weights given by $\mathbf{w} : E(G) \to \mathbf{Q}$.*

(i) If $\varphi(G)$ is obtained from G by replacing a single edge $e \in E$ with a simple path of k edges $P = \{e_1, ..., e_k\}$ with $\mathbf{w}(e_i) = w_i$, then

$$\hat{R}(\varphi(G);\mathbf{w}) = C_P \cdot \hat{R}(G;\mathbf{w}[e \mapsto w'])\,,$$

where

$$\frac{1}{w'} = \frac{1}{w_1} + \cdots + \frac{1}{w_k} \qquad \text{and} \qquad C_P = \frac{1}{w'}\prod_{i=1}^{k} w_i\,. \tag{5.7}$$

(ii) If $\varphi(G)$ is obtained from G by replacing a single edge $e \in E$ with a bundle of k parallel edges $B = \{e_1, \ldots, e_k\}$ with $\mathbf{w}(e_i) = w_i$, then

$$\hat{R}(\varphi(G);\mathbf{w}) = \hat{R}(G;\mathbf{w}[e \mapsto w'])\,,$$

where

$$w' = -1 + \prod_{i=1}^{k}(1 + w_i)\,. \tag{5.8}$$

Proof. We repeat the arguments from the proof in [48, Section 4.3]. For a given set $A \subset E$ of edges we let $\mathbf{w}(P) = \prod_{e \in A}\mathbf{w}(e)$.

Let S be the set of subsets $A \subseteq E \setminus \{e\}$ that already span the whole graph $G = (V,E)$, i.e.

$$S = \{A \subseteq E \setminus \{e\} \,:\, \kappa(A) = 1\}\,,$$

and let T be the set of subsets that need the edge e to span the graph, i.e.

$$T = \{A \subseteq E \setminus \{e\} \,:\, \kappa(A) = 2 \text{ and } \kappa(A \cup \{e\}) = 1\}\,.$$

With w' denoting the weight of the edge e in the graph G, (5.3) gives

$$\hat{R}(G; \mathbf{w}[e \mapsto w']) = \sum_{A \in S} \mathbf{w}(A)(1 + w') + \sum_{A \in T} \mathbf{w}(A) w'\,. \tag{5.9}$$

We will compare the S-sum and the T-sum here to the corresponding sums obtained when we replace the edge e by a path or a bundle.

(i) When φ is the operation described in (ii), we get

$$\begin{aligned}\hat{R}(\varphi(G); \mathbf{w}) &= \sum_{A \in S} \mathbf{w}(A) \left(\mathbf{w}(P) + \sum_{i=1}^{k} \mathbf{w}(P \setminus e_i) \right) + \sum_{A \in T} \mathbf{w}(A)\mathbf{w}(P) \\ &= \sum_{A \in S} \mathbf{w}(A) \left(\prod_{i=1}^{k} w_i + \sum_{j=1}^{k} \prod_{i \neq j} w_i \right) + \sum_{A \in T} \mathbf{w}(A) \prod_{i=1}^{k} w_i\,.\end{aligned}$$

Comparing the S-sum and T-sum here to those in (5.9), it is easy to check that w' and C_p as defined in (5.7) indeed satisfy $\hat{R}(\varphi(G); \mathbf{w}) = C_P \cdot \hat{R}(G; \mathbf{w}[e \mapsto w'])$.

(ii) When φ is the operation described in (ii), we have

$$\hat{R}(\varphi(G); \mathbf{w}) = \sum_{A \in S} \mathbf{w}(A) \left(1 + \sum_{\emptyset \subset A' \subseteq B} \mathbf{w}(A') \right) + \sum_{A \in T} \mathbf{w}(A) \left(\sum_{\emptyset \subset A' \subseteq B} \mathbf{w}(A') \right),$$

and (5.8) follows since

$$\sum_{\emptyset \subset A' \subseteq B} \mathbf{w}(A') = \prod_{i=1}^{k} (w_i + 1) - 1\,.$$

□

The case of uniform edge weights will still be useful to us, so we state it here for easy reference.

Corollary 5.1. *If $\varphi(G)$ is obtained from G by replacing a single edge $e \in E$ with a simple path of k edges of weight w, then*

$$\hat{R}(\varphi(G); \mathbf{w}) = k w^{k-1} \cdot \hat{R}(G; \mathbf{w}[e \mapsto w/k])\,, \tag{5.10}$$

and if it is obtained from G by replacing $e \in E$ with a bundle of k parallel edges of weight w, then

$$\hat{R}(\varphi(G); \mathbf{w}) = \hat{R}(G; \mathbf{w}[e \mapsto (1 + w)^k - 1])\,. \tag{5.11}$$

These rules are transitive [48, Lemma 1], and so can be freely combined for more intricate weight shifts. As the problem *Simple Tutte(x, y)* that we are ultimately analyzing concerns unweighted (or rather 'constantly weighted') graphs G, the weight shifts we use on these graphs must be the same for all edges. This calls for the graph theoretic version of Brylawski's tensor product for matroids [17]. We will use the following terminology.

Definition 5.1 (Graph inflation)**.** *Let H be a graph with two distinguished 'terminal' vertices. For any graph $G = (V, E)$, an H-*inflation *of G, denoted $G \otimes H$, is obtained by replacing every edge $xy \in E$ by (a fresh copy of) H, identifying x with one of the terminals of H and y with the other.*

Stretches and thickenings are special cases of graph inflations. For example, $G^2 = G \otimes ($ $)$ and $G_2 = G \otimes ($ $)$.

Note that if H is not symmetric with respect to its two terminals, then a given graph G may have several non-isomorphic inflations $G \otimes H$, since there are in general two non-isomorphic ways two replace an edge by H. For us this difference does not matter, as the resulting Tutte polynomials turn out to be the same; this follows from the fact that any graph so obtained defines the same cycle matroid, which uniquely determines the Tutte polynomial [25]. Thus we choose $G \otimes H$ arbitrarily among the graphs that satisfy the condition in the definition above. Graph inflation is not commutative and Sokal [89] uses the notation $\vec{G}^H$.

5.3 Hardness of computing coefficients

As a first step towards Theorem 5.1 we consider the problem of computing the coefficients of the Tutte polynomial along fixed hyperbolas H_q. Our starting point is the following hardness result, related to the problem of counting 3-terminal minimum cuts—an approach first considered by Goldberg and Jerrum [48] in the context of approximation complexity.

Lemma 5.2 (Lemma 1 in [30])**.** *Let q be a rational number with $q \notin \{1, 2\}$, and consider a simple graph $G = (V, E)$, with a subset $T \subseteq E$ of three specified edges, and edge weights given by*

$$\mathbf{w}(e) = \begin{cases} -1, & \textit{if } e \in T, \\ w, & \textit{if } e \in E \setminus T. \end{cases} \tag{5.12}$$

Assuming #ETH, there is no $\exp(o(m))$-time algorithm computing the coefficients of the polynomial $w \mapsto Z_0(G; q, \mathbf{w})$ for such weighted graphs G of m edges.

The following lemma gives an extension to the case $q = 0$ of [30, Lemma 2].

Lemma 5.3. *Assuming #ETH, there is no $\exp(o(m))$-time algorithm computing the coefficients of the polynomial $v \mapsto \hat{R}(G; v)$ for simple graphs G of m edges.*

Proof. Let G be a simple graph of m edges, weighted as in (5.12) for three specified edges $T = \{e_1, e_2, e_3\}$. From Lemma 5.2, with $q = 0$, it only remains to get rid of the negative weights and reduce to a single-valued weight function. As only the three edges in T have negative weights, we can use deletion-contraction for this. The proof of Lemma 5.2 given in [30] uses the restriction that the subgraph $(V, E \setminus T)$ is connected, so we can assume that no edge in T is a cut-edge. Three applications of (5.6) with $q = 0$, to delete and contract these T-edges, gives

$$\hat{R}(G; \mathbf{w}) = \sum_{C \subseteq \{1,2,3\}} (-1)^{|C|} \hat{R}(G_C; \mathbf{w}), \tag{5.13}$$

where for each $C \subseteq \{1, 2, 3\}$, the graph G_C is constructed from G according to the rule: if $i \in C$ then contract e_i, otherwise delete e_i. The remaining edges in each graph G_C are in one-to-one correspondence with the edges in $E \setminus T$; especially, they all have the same weight w, so $\hat{R}(G_C; \mathbf{w}) = \hat{R}(G_C; w)$.

The graphs G_C are not necessarily simple, as the contractions may produce multiple edges and loops. To address this we consider the 3-stretch G_C^3. From m applications of the weight-shift identity (5.10), one for every edge of G_C, we get

$$\hat{R}(G_C^3; w) = (3w^2)^m \cdot \hat{R}(G_C; w/3)\,. \tag{5.14}$$

In summary, an algorithm to find the coefficients of $v \mapsto \hat{R}(G; v)$ for simple graphs G could be used to compute the coefficients of $v \mapsto \hat{R}(G_C^3; v)$, and thus via (5.14) the coefficients of $v \mapsto \hat{R}(G_C; v)$ for each subset $C \subseteq \{1, 2, 3\}$. By (5.13) this would give the coefficients of $w \mapsto \hat{R}(G; \mathbf{w})$, which by Lemma 5.2 cannot be done in time $\exp(\Omega(m))$ under #ETH. We note that each 3-stretch G_C^3 has $O(m)$ vertices and edges, so the lower bound is preserved. □

5.4 Hardness of point evaluations

In this section we address the hardness of *Simple Tutte*$(1, y)$ for fixed rational $y \neq 1$, formulated as the problem of computing $\hat{R}(G; w)$ for a given simple graph G and nonzero rational number w. We use a technique based on interpolation. In broad strokes, we seek a suitable class of graphs $\{H_i\}$ such that we can compute the coefficients of the univariate polynomial $v \mapsto \hat{R}(G; v)$ for given G by interpolation from sufficiently many evaluations $\hat{R}(G; w_i) \sim \hat{R}(G \otimes H_i; w)$. For this to work, we need that the number of distinct weight shifts $\{w \mapsto w_i\}$ provided by the graph inflations is at least $m + 1$, one more than the degree of the polynomial.

5.4.1 Wump graphs

Bump! Bump! Bump! Did you ever ride a Wump? We have a Wump, with just one hump. But we know a man called Mr. Gump. Mr. Gump has a seven hump Wump. So... if you like to go Bump! Bump! just jump on the hump of the Wump of Mr. Gump. – Dr. Seuss

We here define a class of *Wump graphs*[1], and show that inflation by such graphs, *Wump inflations*, give rise to distinct weight shifts along the reliability line of the Tutte polynomial. Wump graphs are mildly inspired by l-byte numbers, in the sense that each has associated to it a sequence of length l, such that the lexicographic order of these sequences determines the size of the corresponding shifted weights.

Definition 5.2. *For positive integers i and s, an (i,s)-*hump *is the graph obtained by identifying all the left and all the right endpoints of i simple paths of length s. Given a sequence $S = \langle s_1, s_2, \ldots, s_l\rangle$ of positive integers, the* Wump graph W_S *is the graph obtained by concatenating l humps by their endpoints, where the i-th hump is an (i, s_i)-hump. The number l is the* length *of the Wump graph W_S.*

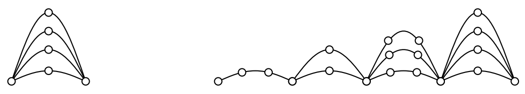

$(4,2)$-hump $\quad\quad$ $S = \langle 3,2,3,2\rangle$

Figure 5.2: Construction of a Wump graph.

Note that for sequences $S = \langle s_1, s_2, \ldots, s_l\rangle$ with bounded elements s_i the size of W_S is $O(l^2)$, and that W_S is simple whenever all elements $s_i \geq 2$.

A Wump-inflation corresponds to a weight-shift for the weighted reliability polynomial as follows.

Lemma 5.4. *For any graph G with m edges, sequence $S = \langle s_1, s_2, \ldots, s_l\rangle$ of positive integers, and non-zero rational number w, we have*

$$\hat{R}(G \otimes W_S; w) = C_S^m \cdot \hat{R}(G; w_S)\,,$$

where

$$\frac{1}{w_S} = \sum_{i=1}^{l} \frac{1}{(1 + w/s_i)^i - 1} \tag{5.15}$$

and

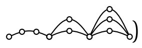
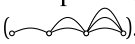
$$C_S = \frac{1}{w_S} \cdot \prod_{i=1}^{l} w^{(s_i-1)i}\left((w+s_i)^i - s_i^i\right)\,.$$

Proof. Starting out with the graph $G \otimes W_S$, we will look at the effect of replacing one of the m canonical copies of W_S with a single edge e. We show that, with φ denoting this operation,

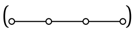
$$\hat{R}(G \otimes W_S; w) = C_S \cdot \hat{R}(\varphi(G \otimes W_S); \mathbf{w}[e \mapsto w_S])\,, \tag{5.16}$$

[1]Originally, such as in [56], Wump graphs were called *Bounce graphs* for obvious reasons (see Figure 5.2), but then the quoted passage from Dr. Seuss's book *One fish two fish red fish blue fish* was brought to attention.

where w_S satisfies (5.15), and $\mathbf{w}$ takes the old value w on all unaffected edges. The lemma then follows by successively applying φ to each canonical copy of W_S in $G \otimes W_S$.

The first step towards transforming a Wump graph (say, ) into a single edge, is to contract each s_i-path of the i-th hump into a single edge, for $i = 1, \dots, l$. In other words, we perform the inverse of an s_i-stretch to the i-th hump. By (5.10) of Corollary 5.1, this 'unstretching' of the i-th hump contributes a factor $(s_i w^{s_i-1})^i$ to the polynomial, and each edge in the resulting $(i, 1)$-hump receives a weight of w/s_i. Repeating this process for each of the l humps in W_S gives a simplified Wump graph W_S^1 generated by a sequence of 1s (). We let $\phi(G \otimes W_S)$ denote the graph obtained from $G \otimes W_S$ by simplifying one canonical copy of W_S like this. By transitivity, we get the weight shift

$$\hat{R}(G \otimes W_S; w) = \Big(\prod_{i=1}^{l} (s_i w^{s_i-1})^i\Big) \cdot \hat{R}(\phi(G \otimes W_S); \mathbf{w}'),$$

where $\mathbf{w}'$ takes the value w/s_i on every edge of the ith hump of the simplified subgraph W_S^1, and the old value w outside W_S^1. Next, we replace each hump of W_S^1 by a single edge, to get a simple path () of length l. This corresponds to the inverse of an i-thickening for ith hump, for $i = 1, \dots, l$. By (5.11) of Corollary 5.1, this 'unthickening' does not produce any new factors for the polynomial, but the weight of the ith edge in the resulting path becomes

$$w_i = (1 + w/s_i)^i - 1 .$$

Finally, we compress the l-path into a single edge e, and the claim in (5.16) follows from Lemma 5.1 (i). □

We now show that Wump inflations provide a rich enough class of weight shifts $\{w \mapsto w_S\}$ to provide points enough for interpolation. The ranges of w for which we prove this are general enough to allow for interpolation on the whole reliability line, and we make no attempt at extending the ranges.

Lemma 5.5. *Let w be a rational number with $w \in (-1, 0)$ or $w \in (9, \infty)$. For all integers $m \geq 1$, there exist integer sequences $S_0, \dots, S_m$ such that*

(i) W_{S_i} is simple with $O(\log^2 m)$ edges for all i, and

(ii) $w_{S_i} \neq w_{S_j}$ for all $i \neq j$, with w_S as defined in (5.15).

Furthermore, such sequences S_i can be computed in time polynomial in m.

Proof. Let r be a positive integer to be chosen later, only depending on w. We consider the set of sequences $S = \langle s_1, \dots, s_l \rangle$ of length $l = r \log(m + 1)$, such that $s_i \in \{2, 3\}$ whenever i is a multiple of r, and $s_i = 2$ otherwise. Since r is a constant and all $s_i \geq 2$, this construction satisfies (i).

Now consider any two distinct such sequences $S = \langle s_i \rangle$ and $T = \langle t_i \rangle$. To address (ii), we consider the difference

$$\Delta = \frac{1}{w_S} - \frac{1}{w_T}\,,$$

and show that $\Delta \neq 0$.

Using Lemma 5.4 we get a sum expression for Δ,

$$\begin{aligned}\Delta &= \sum_{i=1}^{l} \frac{1}{(1+w/s_i)^i - 1} - \sum_{i=1}^{l} \frac{1}{(1+w/t_i)^i - 1} \\ &= \sum_{i=1}^{l} g\left((1+w/s_i)^i\right) - \sum_{i=1}^{l} g\left((1+w/t_i)^i\right)\,,\end{aligned} \tag{5.17}$$

where $g(x) = \frac{1}{x-1}$. This function is negative and strictly decreasing on $(0,1)$, and positive and strictly decreasing on $(1,\infty)$. For convenience we let

$$a = \min\left\{(1+\tfrac{w}{3}),(1+\tfrac{w}{2})\right\} \;\text{ and }\; b = \max\left\{(1+\tfrac{w}{3}),(1+\tfrac{w}{2})\right\}.$$

By the monotonicity of g we have $g(a^i) > g(b^i)$ for all positive numbers i.

Case 1: $w > 9$. Here we have $a = (1+w/3)$ and $b = (1+w/2)$. We set $r = 1$ and let k be the smallest index for which $s_k \neq t_k$. We can safely assume that $s_k = 3$ and $t_k = 2$, otherwise just exchange the roles of S and T. Thus, in (5.17), terms of the sums for $i < k$ cancel. The terms corresponding to $i = k$ give $g(a^k) - g(b^k) > 0$. We apply the monotonicty of g to the terms for $i > k$, which allows us to lower bound Δ as follows,

$$\Delta \geq \left(g(a^k) + \sum_{i=k+1}^{l} g(b^i)\right) - \left(g(b^k) + \sum_{i=k+1}^{l} g(a^i)\right) = f(a) - f(b)\,,$$

where

$$f(x) = g(x^k) - \sum_{i=k+1}^{l} g(x^i) = \frac{1}{x^k - 1} - \sum_{i=k+1}^{l} \frac{1}{x^i - 1}\,. \tag{5.18}$$

We now claim that f is strictly decreasing on $(4,\infty)$. This implies $\Delta > 0$, because for $w > 9$ we have $4 < a < b$, and then $\Delta \geq f(a) - f(b) > 0$. To prove this claim, consider

$$f'(x) = -\frac{kx^{k-1}}{(x^k-1)^2} + \sum_{i=k+1}^{l} \frac{ix^{i-1}}{(x^i-1)^2}\,. \tag{5.19}$$

Let $T_i(x)$ denote the ith term in this sum. For $x > 4$ we have

$$2T_{i+1}(x) < T_i(x)\,,$$

because this inequality is equivalent to

$$2\left(1+\frac{1}{i}\right)x < \left(x+\frac{x-1}{x^i-1}\right)^2 ,$$

and in this expression we have LHS $\leq 4x < x^2 \leq$ RHS for all $x > 4$ and positive integers i. Thus, for $x > 4$ we can upper bounds the derivative in (5.19) as

$$f'(x) < \frac{kx^{k-1}}{(x^k-1)^2}\left(-1+\sum_{i=k+1}^{l}\frac{1}{2^{i-k}}\right) < 0 ,$$

which proves the claim.

Case 2: $w \in (-1, 0)$. Here we have $a = (1 + w/2)$ and $b = (1 + w/3)$. We choose r to be a positive integer that satisfies $b^r < \frac{1}{4}$. Let rk be the smallest index for which $s_{rk} \neq t_{rk}$. We assume w.l.o.g. that $s_{rk} = 3$ and $t_{rk} = 2$, otherwise exchange the roles of S and T. In (5.17), terms of the sum for $i < rk$ cancel, and so do terms for those i's that are not multiples of r. The terms corresponding to $i = rk$ contribute $g(b^{rk}) - g(a^{rk}) < 0$. We apply the monotonicty of g to the remaining terms for $i > rk$, which allows us to upper bound Δ as follows,

$$\Delta \leq \left(g(b^{rk}) + \sum_{i=k+1}^{l/r} g(a^{ri})\right) - \left(g(a^{rk}) + \sum_{i=k+1}^{l/r} g(b^{ri})\right) ,$$

where $g(x) = \frac{1}{x-1}$. For $x \in (0, 1)$, we can expand $g(x)$ into the geometric series

$$g(x) = -\sum_{j=0}^{\infty} x^j .$$

Applying this representation to our estimate for Δ and rearranging terms, we arrive at

$$\begin{aligned}\Delta &\leq \sum_{j=0}^{\infty}\left((a^{rj})^k - (b^{rj})^k + \sum_{i=k+1}^{l/r}\left((b^{rj})^i - (a^{rj})^i\right)\right)\\ &= \sum_{j=0}^{\infty}\left(F(a^{rj}) - F(b^{rj})\right) ,\end{aligned}$$

where F is the function

$$F(y) = y^k - \sum_{i=k+1}^{l/r} y^i .$$

We claim that F is strictly increasing on $(0, \frac{1}{4})$. This implies $\Delta < 0$, because with the given choice of r we have $0 < a^{rj} < b^{rj} < \frac{1}{4}$ for all positive integers j, and then $F(a^{rj}) - F(b^{rj}) < 0$ for $j \geq 1$, and for $j = 0$ the term is 0. To prove this claim, consider

$$F'(y) = ky^{k-1} - \sum_{i=k+1}^{l/r} iy^{i-1} .$$

We obtain $F'(y) > 0$ for $y \in (0, \frac{1}{4})$ from the following calculation.

$$\begin{aligned}\frac{1}{ky^{k-1}} \cdot \sum_{i=k+1}^{l/r} iy^{i-1} &= \sum_{i=k+1}^{l/r} \frac{i}{k} y^{i-k} = \sum_{i=1}^{l/r-k} \left(1 + \frac{i}{k}\right) y^i \\ &\leq \sum_{i=1}^{l/r-k} (1+i)\, y^i \leq \sum_{i=1}^{\infty} y^i + \sum_{i=1}^{\infty} iy^i \\ &= \frac{1}{1-y} - 1 + \frac{y}{(1-y)^2} \leq \frac{4}{3} - 1 + \frac{4}{9} < 1 .\end{aligned}$$

□

5.4.2 Hardness of points on the reliability line

We are ready to prove Theorem 5.1. We consider the following equivalent claim.

Proposition 5.1. *Let $w \neq 0$ be a rational number. Then, assuming #ETH, there is no $\exp(o(m/\log^2 m))$-time algorithm computing the value of $\hat{R}(G; w)$ for simple input graphs G of m edges.*

Proof. If $w < -1$, we can pick a positive integer k big enough such that

$$w' := w/k > -1 .$$

This weight shift corresponds to the k-stretch of G by (5.10) of Corollary 5.1. On the other hand, if $0 < w < 9$, we can pick a positive integer k such that

$$w' := (w/2 + 1)^k - 1 > 9 .$$

This weight shift corresponds to the 2-stretch of the k-thickening of G, by combining the weight shift identities of Corollary 5.1. In any case, the transformed graph G' remains simple, the number of edges is increased by a constant factor of at most $2k$, and the value of $\hat{R}(G; w')$ can be obtained in constant time from the value of $\hat{R}(G'; w)$. Thus we can safely assume that $w \in (-1, 0)$ or $w > 9$.

Now, by contradiction, suppose there was an $\exp(o(m/\log^2 m))$-time algorithm for evaluating $\hat{R}(G; w)$ for simple graphs G of m edges. Given such a graph G, we can, according to Lemma 5.5, construct $m+1$ Wump graphs W_S in polynomial time, such that each Wump inflation $G \otimes W_S$ is simple with $O(m \log^2 m)$ edges, and such that the corresponding weight shifts $w \mapsto w_S$ are all distinct. By Lemma 5.4, we could then use the assumed algorithm to compute $\hat{R}(G \otimes W_S; w) \sim \hat{R}(G; w_S)$ for each W_S, giving $m + 1$ distinct evaluations of the mth-degree polynomial $v \mapsto \hat{R}(G; v)$, and thus its coefficients by interpolation—all in time $\exp(o(m))$. By Lemma 5.3 there can be no such algorithm, unless #ETH fails. □

Chapter 6

Lower Bounds for Planar Graphs

In this chapter we consider the restriction of *Tutte(x, y)* to *planar* input graphs.

Planar Tutte(x, y)

Input:	A planar multigraph G with n vertices and m edges.
Task:	Compute $T(G; x, y)$.

For this problem the complexity picture of the Tutte plane is slightly different. One of the most important differences is the hyperbola H_2, i.e. the set of points (x, y) satisfying $(x - 1)(y - 1) = 2$, along which *Planar Tutte(x, y)* is polynomial-time solvable, in addition to the trivial points for *Tutte(x, y)*. This is a big thing in theoretical physics, because on this hyperbola the Tutte polynomial specializes to the partition function of the so called *Ising model* (a.k.a. the 2-state Potts model). Vertigan [95] showed that that except for the hyperbolas H_2, H_1 and the points $(-1, -1)$, $(0, -1)$, $(-1, 0)$, and $(1, 1)$, every other point of the Tutte plane is #P-hard for *Planar Tutte(x, y)*.

From the perspective of exponential-time complexity, the planar restriction gives an even more striking difference, because even at the #P-hard points, the problem *Planar Tutte(x, y)* has an upper bound that is exponential only in $O(\sqrt{n})$, due to an algorithm by Sekine *et al.* [87]. A natural question is then whether this is optimal under #ETH. To investigate this we will dig into the literature and look at existing standard reductions with exponential-time glasses. We will see that the algorithm by Sekine *et al.* is indeed optimal for the general problem, as a matching lower bound under #ETH is found for points on the hyperbola H_3. For remaining regions of the plane, lower bounds exponential in $\Omega(n^{1/k})$ are found under #ETH for various $k > 2$. In particular, for points on the line $y = 1$ we get a lower bound exponential in $\Omega(n^{1/8})$, which seems to be the first explicit superpolynomial lower bound also for the general problem *Tutte($x, 1$)*. All bounds are shown to hold for graphs of $O(n^2)$ edges; see Figure 6.1 for an overview.

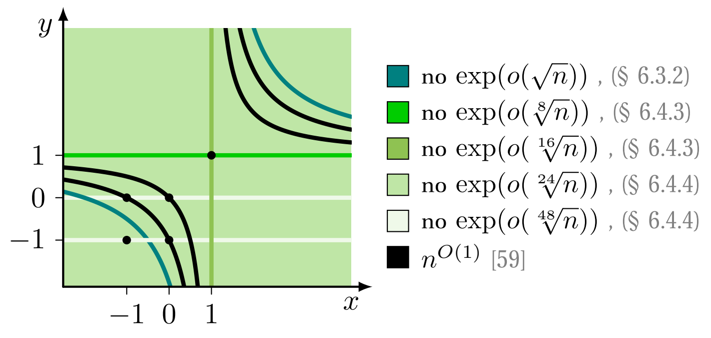


Figure 6.1: Exponential time complexity under #ETH of the Tutte plane for planar graphs in terms of n, the number of vertices. All bounds hold in particular for graphs with $m \in O(n^2)$ edges.

6.1 Planar Tutte computations

It is generally considered one of the landmarks of statistical physics when Onsager [81] could give a closed form expression of the partition function of the Ising model (or 2-state Potts model) for planar lattice graphs in 1944. A number of serious attempts were made over the years to extend his result to other nontrivial graph classes, and to q-state Potts models for $q > 2$, but no progress was made. A vivid description of the frustrations of these years can be found in the introduction of [58]. With our current knowledge of computational tractability over the Tutte plane, and the relation (4.7) between the q-state Potts model and the Tutte polynomial along H_q, we now have an explanation to the failure of these attempts. The #P-hardness map of the Tutte plane by Jaeger *et al.* [59] implies that Onsager's result is unlikely to be extended to general graphs, and the corresponding #P-hardness map for planar graphs by Vertigan [95] implies that Onsager's result is unlikely to be extended to $q > 2$ for planar graphs.

A subexponential upper bound From the previous chapters we know that, assuming #ETH, computation of $T(G; x, y)$ for general fixed (x, y) requires time exponential in the number of vertices or edges for multigraphs, and that for simple graphs we are within a polylogarithmic factor, in the exponent, of this bound. The situation for the planar case is quite different, as there is a $2^{O(\sqrt{n})}m^{O(1)}$-time algorithm by Sekine *et al.* [87] from 1995. This situation is common to many other NP-hard problems, such as *Independent Set* [77] and *Hamiltonian Cycle* [28] for which algorithms exponential in $O(\sqrt{n})$ are also known for planar graphs. These algorithms all use an approach introduced by Lipton and Tarjan [76, 77], namely, a combination of dynamic programming and the *planar separator theorem*. This theorem states that in a planar graph one can efficiently find a subset of $O(\sqrt{n})$ vertices whose removal splits the graph into components of size at most $2n/3$. Without going into details, the algorithm of Sekine *et al.* computes $T(G; x, y)$ for planar G recursively, using the deletion-contraction expansion of (4.13), and the planar separator theorem is involved in finding a beneficial ordering of edges to be deleted-and-contracted, such that the number of redundant computations can be more effectively reduced compared to the case of a general recursion tree.

6.1.1 Main theorems

The question under consideration here is whether the $2^{O(\sqrt{n})}m^{O(1)}$-time algorithm for *Planar Tutte*(x, y) by Sekine *et al.* has an asymptotically optimal dependency on the number n of vertices, assuming #ETH. As remarked in Section 4.3.2, the number m of edges for a general input graph could in fact be exponential in n, so we need to make sure that any lower bound in terms of n holds for graphs that are not too dense.

In [29], an almost complete map of the exponential-time complexity across the Tutte plane for both multigraphs and simple graphs was presented. This was achieved by finding exact, exponential-time-preserving, versions of reductions in Goldberg and Jerrum's paper [48], concerning NP-hardness of approximation. Motivated by this success, a similar translation of the same authors' reductions for the planar case in [49] was sought. This worked out only for points on the hyperbola H_3, which is somewhat surprising as the upper branch of this hyperbola was not covered by the inapproximability result of [49].

Theorem 6.1. *Let* $(x, y) \in \mathbf{Q}^2$ *with* $(x-1)(y-1) = 3$ *and* $y \neq -1$. *Then, assuming #ETH, there is no* $\exp(o(\sqrt{n}))$*-time algorithm for* Planar Tutte(x, y). *The bound holds especially for input graphs of* $O(n^2)$ *edges.*

Consequently, the algorithm by Sekine *et al.* is indeed optimal for the general problem *Planar Tutte*(x, y), assuming #ETH. Proofs are given in Section 6.3.

Unfortunately, the idea of the reduction in [49] that covers the main regions of the plane seems to not lend itself easily to our purpose; see discussion in section 6.5.2. Instead, the #P-hardness reductions of Vertigan [95] were studied, which led to conditional lower bounds of the type $\exp(\Omega(n^{1/k}))$ for various $k > 2$. The conclusions are contained in the following theorem. These bounds leave the question of exponential-time complexity of *Planar Tutte*(x, y) very much open for most of the Tutte plane, but are included here for a complete picture of the current situation.

Theorem 6.2. *Let* $(x, y) \in \mathbf{Q}^2$ *with* $(x-1)(y-1) \notin \{1, 2, 3\}$. *Then, assuming #ETH, there is no algorithm for* Planar Tutte(x, y) *with running time*

(i) $\exp(o(\sqrt[8]{n}))$ *if* $y = 1$ *and* $x \neq 1$,

(ii) $\exp(o(\sqrt[16]{n}))$ *if* $x = 1$ *and* $y \neq 1$,

(iii) $\exp(o(\sqrt[24]{n}))$ *for any remaining point with* $x \neq 1$ *and* $y \notin \{0, -1\}$,

(iv) $\exp(o(\sqrt[48]{n}))$ *for any remaining point with* $x \neq 1$ *and* $y \in \{0, -1\}$, *even for simple planar graphs.*

The bounds hold especially for input graphs of $O(n^2)$ *edges.*

Proofs are in Section 6.4. The result for the line $y = 1$ is of particular interest, as this seems to be the first concrete superpolynomial lower bound also for the more general problem *Tutte($x, 1$)*. In fact, the only proof of #P-hardness of *Tutte($x, 1$)* seems to rely on the hardness proof for the planar case.

6.1.2 Related work

The stuff of this chapter builds heavily on two papers.

The first, by Goldberg and Jerrum [49], shows hardness of approximation of *Planar Tutte(x, y)* for large regions of the Tutte plane, in the sense of excluding a *fully polynomial randomized approximation scheme* (FPRAS) under the assumption that RP≠NP. This was a follow-up on their previous work [48] on general graphs.

The second paper, by Vertigan [95][1], contains the mentioned result of #P-hardness for every point of the Tutte plane except those on H_1, H_2 and the points $(-1, -1)$, $(0, -1)$, $(-1, 0)$, and $(1, 1)$. In fact, Vertigan's results are explicitly stated for any *algebraic* point (x, y), an extension that we will also use at one point.

6.2 Preliminaries

Reductions in this chapter will repeatedly involve the problem of finding coefficients of various graph polynomial $s \mapsto p(G; s)$ for planar graphs G. We let $[s^d]p(G; s)$ denote the coefficient of s^d in $p(G; s)$. Below we give some basic properties of planar graphs that we will use.

Euler's equation. Euler's equation says that if n, m, f denote the number of vertices, edges and faces, respectively, of a planar graph G, then

$$n - m + f = 2\,. \tag{6.1}$$

In the case that the planar graph G is also *simple*, then each face is bounded by at least three edges, and we get

$$m \leq 3n + 6\,. \tag{6.2}$$

The Tutte polynomial for dual graphs. A given planar graph G may have different non-isomorphic duals G^*, resulting from different planar drawings of G, but the Tutte polynomial is blind to this difference due to the following pleasant property,

$$T(G; x, y) = T(G^*; y, x)\,. \tag{6.3}$$

This identity is classically used for mirroring hardness results over the line $y = x$ in the Tutte plane. However, for exponential-time complexity we must use this with caution, because, unless G is of bounded degree, the number of faces f in G, and thus the number $n^* = f$ of vertices in G^*, may be much larger than n.

[1]This work appeared already in Vertigan's PhD thesis from 1991 [94].

6.3 The 3-state Potts hyperbola

To prove Theorem 6.1 we start out from a coloring problem. The same approach is used in [49, Section 4].

6.3.1 Planar 3-coloring

We give a lower bound for the planar version of the following classical problem.

3-Coloring

Input:	A simple graph G with n vertices and m edges.
Task:	Decide whether there is a proper 3-coloring of G.

For general simple graphs, an edge-exponential lower bound under ETH was given already in [57]. In [30], this result was extended to the counting version *#3-Coloring* under #ETH.

Theorem 6.3 (Dell *et al.* [30])**.** *Assuming #ETH, there is no* $\exp(o(m))$*-time algorithm for* #3-Coloring.

We consider the restriction of this problem to planar graphs, *#Planar 3-Coloring*, and show the following corresponding lower bound.

Theorem 6.4. *Assuming #ETH, there is no* $\exp(o(\sqrt{n}))$*-time algorithm for* #Planar 3-Coloring.

Note that for connected, simple, planar graphs we have $n \leq m \leq 3n - 6$ by Euler's equation, so this bound is equivalent to the same bounds in terms of m. Thus, given Theorem 6.3, it is enough to find a reduction from *#Planar 3-Coloring* to its planar counterpart, such that the number of edges does not increase too much. This is provided by the following lemma.

Lemma 6.1. *There is a mapping reduction from* #3-Coloring *to* #Planar 3-Coloring, *mapping graphs of* m *edges to graphs of* $O(m^2)$ *edges.*

Proof. We show that the claim is satisfied by a standard reduction from *3-Coloring* to *Planar 3-Coloring* in [44], if we replace the crossover gadget used in [44] by the graph W shown in Figure 6.2, which was introduced in [3] to preserve information about the number of colorings.

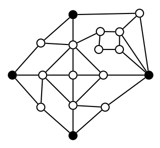


Figure 6.2: W, with connection vertices in black.

Given a simple input graph G to *#3-Coloring*, consider a drawing of G in the plane such that at most two edges cross at any given point. A simple planar graph G' is constructed from this drawing by successively picking an edge that is crossed by other edges, along this edge replacing each crossing by a copy of the gadget W, without introducing new edge intersections; see Figure 6.3. It is shown in [3] that G has k proper 3-colorings if, and only if, G' has $k \cdot 2^{c(G)}$ proper 3-colorings, where $c(G)$ is the number of edge-intersections in the drawing of G. We note that the number of edges of G' satisfies $m' \leq m + 31c(G)$. Since $c(G) < m^2$ for simple graphs G, this means $m' \in O(m^2)$. The claim follows. $\square$

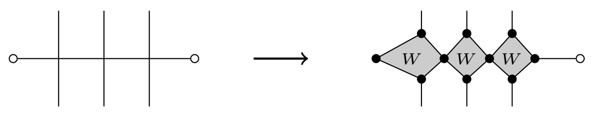


Figure 6.3: Example demonstrating the proper insertion of the crossover gadget W.

6.3.2 Hardness of points on the 3-state Potts hyperbola

We are ready to prove Theorem 6.1, concerning points on the hyperbola H_3. For such points $(x, y) \in H_3$, Fortuin and Kasteleyn's identity (4.9) gives

$$(y-1)^n(x-1)^{\kappa(E)}T(G; x, y) = Z_3(G; w)\,, \tag{6.4}$$

where $w = y - 1$. Thus we can use the formulation of the 3-state Potts polynomial $Z_3(G; w)$. We first collect some useful properties of this polynomial.

By (4.6), we can express $Z_3(G; w)$ in terms of the variable $y = w + 1$ as

$$Z_3(G; y-1) = \sum_{\sigma:V\to\{1,2,3\}} y^{\mathrm{mono}(\sigma,G)}\,, \tag{6.5}$$

where $\mathrm{mono}(\sigma, G)$ is the number of *monochromatic* edges of G under σ, i.e. the number of edges whose end vertices are both assigned the same value by σ. This representation easily gives us a weight-shift identity for thickenings: a k-thickening G_k satisfies $\mathrm{mono}(\sigma, G_k) = k \cdot \mathrm{mono}(\sigma, G)$, and thus

$$Z_3(G_k; y-1) = \sum_{\sigma:V\to\{1,2,3\}} y^{k\cdot\mathrm{mono}(\sigma,G)} = Z_3(G; y^k - 1)\,. \tag{6.6}$$

Proof of Theorem 6.1. The following proposition proves Theorem 6.1. We state a slightly stronger claim, as we will later need this result for a certain irrational point. As remarked in Section 4.3 this is not an issue; see e.g. [59, p.41].

Proposition 6.1. *Let $(x, y) \in \mathbf{R}^2$ be an algebraic point on the hyperbola H_3 with $y \neq -1$. Then, assuming #ETH, there is no $\exp(o(\sqrt{n}))$-time algorithm for the problem* Planar Tutte(x, y) *for input graphs of $O(n^2)$ edges.*

Proof. We first note that the problem *Planar Tutte($-2, 0$)* corresponds precisely to the problem *#Planar 3-Coloring*. To see this, consider (6.5) and note that we may interpret each state configuration $\sigma : V \to \{1, 2, 3\}$ as a 3-coloring of the input graph, so that $\mathrm{mono}(\sigma, G) = 0$ if, and only if, σ is a proper 3-coloring. Thus the value $Z_3(G; -1)$, corresponding to $T(G; -2, 0)$, gives the number of proper 3-colorings of G. It follows from Theorem 6.4 that there is no $\exp(o(\sqrt{n}))$-time algorithm for *Planar Tutte($-2, 0$)* under #ETH for simple input graphs.

Let $(x, y) \in H_3$ with $y \notin \{-1, 0\}$, and let G be a simple, planar graph of n vertices and m edges. As $x, y \neq 1$ on the hyperbola H_3, an algorithm for *Planar Tutte(x, y)* would by (6.4) and (6.6) let us compute

$$T(G_k; x, y) \sim Z_3(G_k; y-1) = Z_3(G; y^k - 1)$$

for $k = 1, \dots, m+1$. Since $y \notin \{0, 1, -1\}$ this gives $m+1$ distinct evaluations of the mth degree polynomial $w \mapsto Z_3(G; w)$, and thus we could get the coefficients by interpolation. This in turn would let us evaluate $Z_3(G; 0)$, which by the above discussion cannot be done in time $\exp(o(\sqrt{n}))$ under #ETH. As G is simple and planar, Euler's equation gives $m \in O(n)$, so the thickened graph G_k will have n vertices and $km \in O(n^2)$ edges for any $k \le m+1$, and the claim follows. □

6.4 The rest of the plane

For the remaining points of the plane we turn to the work done by Vertigan [95] concerning #P-hardness for *Planar Tutte*(x, y). These results rely on a hardness result by Provan [83] of computing *two-terminal network reliability*—a concept related to the all-terminal reliability studied in Chapter 5—which in turn relies on the hardness of counting Hamiltonian cycles in cubic graphs.

6.4.1 Hardness of Hamiltonian cycles in cubic planar graphs

We consider the following problem.

#Planar Cubic Hamiltonian Cycle

Input:	A cubic, planar graph G with n vertices and m edges.
Task:	Find the number of Hamiltonian cycles in G.

The decision version of this problem was shown *NP*-hard in [46] by reduction from *3-Sat*. Provan [83] noted that the same reduction shows #P-hardness of *#Planar Cubic Hamiltonian Cycle* by reduction from *#3-Sat*. In fact, it is made explicit in [46, p.711] that the construction maps a 3-CNF formula φ of n variables to a cubic planar graph G of $O(n^2)$ vertices. Thus we have the following conditional lower bound.

Theorem 6.5. *Assuming #ETH, there is no* $\exp(o(\sqrt{n}))$*-time algorithm for the problem* #Planar Cubic Hamiltonian Cycle.

6.4.2 Hardness of two-terminal reliability

Like the concept of all-terminal network reliability defined in the previous chapter, the two-terminal version also considers a connected graph $G = (V, E)$ with edge failure probability p, but instead of the probability $R(G; p)$ that the full graph G stays connected, we are given two vertices $s, t \in V$ and consider the probability $R_{s,t}(G; p)$ that s and t remain connected by a path. We can express this probability as

$$R_{s,t}(G;p) = \sum_{\substack{A \subseteq E \\ s,t \text{ connected}}} p^{|A|}(1-p)^{|E|-|A|} \,. \tag{6.7}$$

Note that the sum in (6.7) is taken over all edge subsets containing an (s, t)-path; we let $\mathcal{E}_{s,t}(G)$ denote the collection of all such subsets. We will ultimately be interested in the problem of computing the size of this set. From (6.7) we see that

$$|\mathcal{E}_{s,t}(G)| = 2^{|E|} R_{s,t}(G; 1/2)\,. \tag{6.8}$$

For a given probability p, define the problem

Δ3-Planar Path Reliability(p)

Input:	A planar graph $G = (V, E)$ of maximum degree three with n vertices and m edges and $s, t \in V$.
Task:	Compute $R_{s,t}(G; p)$.

In [83], #P-hardness of this problem is shown by reduction from *#Planar Cubic Hamiltonian Cycle*. We obtain the following lower bound.

Proposition 6.2. *Let* $p \in (0, 1)$ *be a fixed rational number. Assuming #ETH, there is no* $\exp(o(\sqrt[4]{n}))$*-time algorithm for the problem* Δ3-Planar Path Reliability(p). *In particular, the bound applies to computation of* $|\mathcal{E}_{s,t}(G)|$.

To prove this, we study the vertex expansion of the reduction in [83].

Provan's reductions. There are two steps to the reduction in [83]. The first step concerns a certain intermediate quantity called $R_{s,t}(G, S, l, k)$, defined for a graph $G = (V, E)$ with terminals $s, t \in V$, an edge subset $S \subseteq E$, and numbers $l, k \in \mathbf{N}$. In [83, p.697], this quantity is related to $R_{s,t}(G; p)$ by the equation

$$R_{s,t}(G^{q,r}; p) = \sum_{i,j} (-p^{q+1})^i (-p^{r+1})^j R_{s,t}(G, S, i, j)\,, \tag{6.9}$$

where $G^{q,r}$ is the graph obtained from G by $(q + 1)$-stretching every edge in S and $(r + 1)$-stretching every edge in $E \setminus S$.

Define the problem

Δ3-Planar Provan

Input:	A planar graph $G = (V, E)$ of maximum degree three, with n vertices, $s, t \in V$, a subset $S \subseteq E$, and numbers $l, k \in \mathbf{N}$.
Task:	Compute $R_{s,t}(G, S, l, k)$.

The first step of the reduction is the following result.

Lemma 6.2 (Theorem 1 in [83]). *There is a mapping reduction from* #Planar Cubic Hamiltonian Cycle *to the problem* Δ3-Planar Provan *that maps graphs of* n *vertices to graphs of* $3n + 2$ *vertices.*

This fact, together with the following lemma and the $\exp(\Omega(\sqrt{n}))$-time lower bound for *#Planar Cubic Hamiltonian Cycle* of Theorem 6.5, proves Proposition 6.2.

Lemma 6.3. *For any probability* $p \in (0, 1)$ *there is a polynomial-time reduction from the problem* Δ3-Planar Provan *to* Δ3-Planar Path Reliability(p) *that maps graphs of* n *vertices to sets of graphs of* $O(n^2)$ *vertices each.*

Proof. We spell out some details omitted in [83], and study the vertex expansion of the reduction suggested in [83, Corollary 2].

Suppose we want to compute $R_{s,t}(G,S,l,k)$ for some planar graph G of maximum degree three with $s,t \in V$ and $S \subseteq E$. To this end we can use (6.9) and an assumed algorithm for *Δ3-Planar Path Reliability*(p), as follows. For fixed r, the sum in (6.9) can be written as $\pi_r(-p^{q+1})$ for a polynomial $\pi_r(x)$ of degree $|S|$. By evaluating $R_{s,t}(G^{q,r};p)$ for $q=0,\ldots,|S|$, we get $|S|+1$ distinct evaluations of the polynomial $x \mapsto \pi_r(x)$, and thus the coefficients by interpolation. Especially we get the lth coefficient, $c_l(r) = [x^l]\pi_r(x)$, which can also be written as

$$c_l(r) = [(-p^{q+1})^l]R_{s,t}(G^{q,r};p) = \sum_j (-p^{r+1})^j R_{s,t}(G,S,l,j)\,.$$

Thus $c_l(r)$ can be seen a polynomial in $(-p^{r+1})$, and we can get the coefficients of this polynomial by interpolation from the values of $c_l(r)$ for $r=0,\ldots,|E\setminus S|$. In particular, we get the kth coefficient $[(-p^{r+1})^k]c_l(r) = R_{s,t}(G,S,l,k)$.

We note that each stretched graph $G^{q,r}$ used in the reduction has $O(n^2)$ vertices. □

6.4.3 Hardness on $y=1$ and $x=1$

We are ready to prove Theorem 6.2, and start by considering the line $y=1$ in the Tutte plane, the line that is left open in [29] for both multigraphs and simple graphs. On this line the Tutte polynomial specializes to the generating function of forests in the graph, weighted by the number of edges,

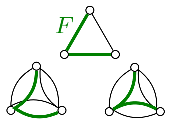
$$F(G;w) = \sum_{\substack{F\subseteq E\\ (V,F)\text{ forest}}} w^{|F|}\,. \tag{6.10}$$

Since $|F| = |V| - \kappa(F)$ whenever (V,F) is a forest, we get the following relation to the Tutte polynomial,

$$F(G;w) = w^{|V|-\kappa(E)}T(G;1+1/w,1)\,. \tag{6.11}$$

Note that for every forest (F,V) in a graph G, there are $k^{|F|}$ corresponding forest of the same size in the k-thickening G_k, since for every edge in F there are k choices for a corresponding edge in G_k. Thus we get the following weight-shift identity for thickenings,

$$F(G_k;w) = F(G;kw)\,. \tag{6.12}$$



Two forests in G_2 corresponding to the same forest F in G.

Hardness of coefficients. We first consider the problem of finding coefficients of the polynomial $w \mapsto F(G;w)$. Vertigan [95] shows #P-hardness of this problem by reduction from *Δ3-Planar Path Reliability*(p); more precisely, from the problem of computing the quantity $|\mathcal{E}_{s,t}(G)|$ defined in (6.8).[2] We study the vertex expansion of this reduction. For convenience, let

$$\hat{F}(G;v) = (1-v)^m F\big(G;\frac{v}{1-v}\big)\,, \tag{6.13}$$

and note that this is a polynomial in v.

Lemma 6.4. *Given a 2-connected, planar graph G with no self-loops and m edges, there is a simple, connected, planar graph $G_{s,t}$ with $m' \in O(m^2)$ edges, such that with $d = m' - 8(m-1)$,*

$$[v^d]\hat{F}(G_{s,t};v) = 4|\mathcal{E}_{s,t}(G)|\,.$$

Proof. In [95], Vertigan constructs a connected, planar graph $G^{2gm}_{*\#} = (V', E')$ from G, such that with $d = |E'| - 8(m-1)$,

$$4|\mathcal{E}_{s,t}(G)| = [v^d]\left(v^{|V'|-\kappa(E')}(1-v)^{|E'|-|V'|+\kappa(E')}T(G^{2gm}_{*\#};1/v,1)\right)\,.$$

This is shown in [95, Lemma 12.6]. Using (6.11) and (6.13), we can write the right hand side here as $[v^d]\hat{F}(G^{2gm}_{*\#};v)$, which shows the last part of the claim. The number of edges of $G^{2gm}_{*\#}$ is given explicitly in [95, (11.11)][3]; in terms of our notation it is

$$|E'| = 32(m-1)(6m-1) \in O(m^2)\,.$$

Thus it only remains to check that the graph $G^{2gm}_{*\#}$ is simple, which is not explicitly stated in [95].

To build $G^{2gm}_{*\#}$, Vertigan first considers a simple, planar construction called the *radial* of G, denoted $G^\diamond$, which has one face for every edge of G, and each of whose faces is bounded by a 4-cycle (including the outer face). Figure 6.4 shows an example. The radial of G is used as a template for forming $G^{2gm}_{*\#}$, by placing copies of the 'tile graphs' depicted in Figure 6.5 side-by-side along their dashed borders, according to the face-pattern of the radial. That is, in each face of $G^\diamond$ we fit one tile graph T, aligning each of the four dashed sides of T with one of the four edges surrounding the given face in $G^\diamond$, and then we remove $G^\diamond$ itself. In this construction the graph T_* is used for one of the tiles, and $\tilde{T}^2_\#$ for the remaining $m-1$ tiles. The tiles are drawn so that the connection vertices of two adjacent tiles pairwise overlap along the shared border in the natural way. The tiles are then glued together by identifying overlapping connection vertices. The resulting graph will clearly be connected, and also planar, because each tile is planar, the connection of tiles at inner faces of $G^\diamond$ will clearly be planar, and the tile placed at the outer face can always be turned 'inside-out' to avoid intersecting edges with the inner tiles. We note that the graph will also be simple, because the tile graphs T_* and $\tilde{T}^2_\#$ are both simple, and no double

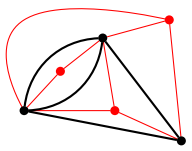


Figure 6.4: A planar graph G (black vertices and edges), and its radial $G^\diamond$ (black and red vertices, red edges only).

[2]In [95] the quantity $|\mathcal{E}_{s,t}(G)|$ is called $T_L(G,g;2,2)$.

[3]The numbers here appear different from those in [95], because Vertigan denotes by n the number of *edges* of G, i.e $n = |E|$, and the number m in [95, (11.11)] is defined as $m = 16(|E|-1)$.

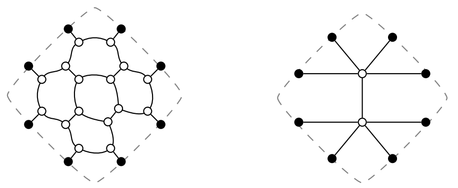


Figure 6.5: Tile graphs $\tilde{T}^2_{\#}$ (left) and T_* (right) with connection vertices in black. The dashed borders are only helping lines, not edges of the graphs.

edges or loops can form by connecting copies of hem at corresponding connection vertices along some dashed side. Finally, to complete the construction of $G^{2gm}_{*\#}$, an m-stretch is applied to a subset of the edges. This preserves connectedness, planarity and also simplicity of the graph.

□

We get the following result.

Lemma 6.5. *Assuming #ETH, there is no* $\exp(o(\sqrt[8]{n}))$*-time algorithm computing the coefficients of* $w \mapsto F(G; w)$ *for simple, planar graphs* G *with* n *vertices.*

Proof. Indeed, suppose there was such an algorithm. Let G be a planar graph of maximum degree three, with n vertices and thus $m \leq 3n/2$ edges. As usual we can also assume that the graph is 2-connected and without loops. The corresponding graph $G_{s,t}$ from Lemma 6.4 would be a connected, simple, planar graph with $n' \leq m' \in O(m^2) = O(n^2)$ vertices. We could then use the assumed algorithm to compute, via (6.13), the coefficients of $v \mapsto \hat{F}(G_{s,t}; v)$, in particular the coefficient of v^d and thus the value of $|\mathcal{E}_{s,t}(G)|$, in time $2^{o(\sqrt[8]{n'})} = 2^{o(\sqrt[4]{n})}$. By Theorem 6.2, there cannot be such an algorithm, assuming #ETH. □

Hardness of evaluations. The following proposition establishes part (i) and (ii) of Theorem 6.2.

Proposition 6.3. *Let* $x, y \neq 1$. *Then, assuming #ETH,*

(i) there is no $\exp(o(\sqrt[8]{n}))$*-time algorithm for* Planar Tutte($x, 1$), *and*

(ii) there is no $\exp(o(\sqrt[16]{n}))$*-time algorithm for* Planar Tutte($1, y$),

for input graphs of $O(n^2)$ *edges.*

Proof. For $x \neq 1$ the problem *Planar Tutte($x, 1$)* is equivalent to the problem of evaluating $F(G; w)$ at $w = 1/(x-1)$ for planar graphs G (this follows from (6.11)). Let G be a simple, planar graph with n vertices and, by Euler's equation, $m \in O(n)$ edges, and suppose we want to compute the coefficients of the mth degree polynomial $w \mapsto F(G; w)$.

(i) By the thickening identity (6.12), an algorithm for *Planar Tutte(x, 1)* could be used to compute $F(G_k; w) = F(G; kw)$ for $k = 1, \ldots, m + 1$, giving $m + 1$ distinct evaluations of $w \mapsto F(G; w)$ and thus the coefficients by interpolation. As G_k has n vertices and $O(m^2) = O(n^2)$ edges for $k \leq m + 1$, the given bound follows from Lemma 6.5.

(ii) By (6.3) we have $T(G_k^*; y, 1) = T(G_k; 1, y)$ for any planar dual G_k^* of G_k. Thus the above interpolation procedure for obtaining the coefficients of $w \mapsto F(G; w)$ could be performed also with an algorithm for *Planar Tutte(1, y)*. We note that G_k^* will be a connected graph with the same number $m' \in O(n^2)$ of edges as G_k, and, by Euler's theorem, with $n^* \in O(m') \subset O(n^2)$ vertices. The claimed bound for *Planar Tutte(1, y)* then follows from the above bound for *Planar Tutte(x, 1)*. □

6.4.4 Hardness for remaining points

Finally we consider points on hyperbolas H_q for remaining rationals $q \notin \{0, 1, 2, 3\}$. For technical simplification we will use the formulation of the multivariate Tutte polynomial defined in (4.10). Recall that $T(G; x, y)$ is computationally equivalent to $Z(G; q, w)$, with $q = (x - 1)(y - 1)$ and $w = y - 1$, whenever $q \neq 0$.

Hardness of coefficients. We first consider the problem of finding coefficients of the polynomial $w \mapsto Z(G; q, w)$ for planar graphs. Vertigan [95] shows #P-hardness also of this problem by reduction from two-terminal reliability. However, this reduction gives a much worse vertex expansion than the corresponding reduction for the forest-counting polynomial. To get a somewhat better superpolynomial lower bound, we will instead use Proposition 6.1 and reduce from a point on the hyperbola H_3. We first use an intermediate reduction from [95].

Let

$$\hat{Z}(G; q, v) = (1 - v)^{|E|} q^{-|V|} Z(G; q, qv/(1 - v)), \tag{6.14}$$

and note that this is a polynomial in v.

Lemma 6.6. *Given a connected, planar graph G with n vertices and m edges, and a natural number k, there is a simple, connected, planar graph $G\langle k \rangle$ with $m' \in O(k^4 m^2)$ edges, such that with $d = m' - 2k^2 m$ and $x_k = 1 + (q - 1)^k$,*

$$[v^d] \hat{Z}(G\langle k \rangle; q, v) = (q - 1)^{k \cdot \kappa(E)} T(G; x_k, x_k).$$

Proof. In [95], a graph called $G_\#^k$ is constructed by glueing together copies of a certain tile graph $\tilde{T}_\#^k$ in the same fashion as described in the proof of Lemma 6.4. The graph $\tilde{T}_\#^k$ is a generalization of the graph $\tilde{T}_\#^2$ in Figure 6.5 to $4k$ connection vertices, with the same grid-like structure. Thus $G_\#^k$ will also be connected, simple and planar.

Let $(V', E') = G^k_{\#}$. From [95, (11.2)][4] it follows that

$$|E'| = 8k^3(k+1)m^2 + 4k^2m \in O(k^4m^2)\,,$$

and it is shown in [95, Lemma 12.3] that with $d = |E'| - 2k^2m$,

$$(q-1)^{k\cdot\kappa(E)}T(G; x_k, x_k) = [v^d]C_vT\left(G^k_{\#}; \frac{1}{v}, 1 + \frac{qv}{1-v}\right)\,,$$

where $C_v = v^{n'-\kappa(E')}(1-v)^{m'-n'+\kappa(E')}$. The expression on the right hand side can also be written as $[v^d]\hat{Z}(G^k_{\#}; q, v)$. □

We get the following result.

Lemma 6.7. *Let $q \notin \{0,1,2\}$. Then, assuming #ETH, there is no $\exp(o(\sqrt[24]{n}))$-time algorithm computing the coefficients of $w \mapsto Z(G; q, w)$ for simple, planar graphs G of n vertices.*

Proof. Let $\alpha = 1+\sqrt{3}$, and note that $(\alpha, \alpha) \in H_3$ and that this is an algebraic point. It follows from Proposition 6.1 that there can be no $\exp(o(\sqrt{n}))$-time algorithm for *Planar Tutte(α, α)* under #ETH, even when restricted to graphs of $m \in O(n^2)$ edges. As (α, α) is a point on the line $y = x$ in the Tutte plane, the same lower bound must then apply to the problem of computing the coefficients of $x \mapsto T(G; x, x)$ for planar graphs G with $O(n^2)$ edges. This is the starting point.

Now suppose there was an $\exp(o(\sqrt[24]{n}))$-time algorithm computing the coefficients of $w \mapsto Z(G; q, w)$ for planar graph with $O(n^2)$ edges. Let G be such a graph, let $k \le m+1$, and consider the corresponding connected, simple, planar graph $G\langle k\rangle$ as defined in Lemma 6.6. Since G has $m \in O(n^2)$ edges, the graph $G\langle k\rangle$ has $n' \le m' \in O(m^2k^4) \subseteq O(n^{12})$ vertices, according to Lemma 6.6. We could then use the assumed algorithm to compute, via (6.14), the coefficients of $v \mapsto \hat{Z}(G\langle k\rangle; q, v)$. In particular, we would get the coefficient of v^d and thus the value of $T(G; x_k, x_k)$, with $x_k = 1 + (q-1)^k$ and d as defined in Lemma 6.6, since $q \neq 1$. Repeating this procedure for $k = 1, \ldots, m+1$ would provide $m+1$ evaluations of the mth degree polynomial $x \mapsto T(G; x, x)$, and these evaluations would all be distinct since $q \notin \{0,1,2\}$. Hence we could get the full polynomial $x \mapsto T(G; x, x)$ by interpolation. The claim then follows from the above discussion. □

Hardness of evaluations. The following proposition establishes part (iii) and (iv) of Theorem 6.2.

Proposition 6.4. *Let (x, y) be a nontrivial point in the Tutte plane with $q = (x-1)(y-1) \notin \{0,1,2\}$. Then, assuming #ETH, there is no algorithm for* Planar Tutte(x, y) *with running time*

[4]Again, in [95] Vertigan denotes by n the number of *edges* of G, i.e $n = |E|$, and the number m in [95, (11.2)] is defined as $m = 4k^2n$.

(i) $\exp(o(\sqrt[24]{n}))$ *if* $x \neq 1$ *and* $y \notin \{0, -1\}$ *for input graphs of* $O(n^2)$ *vertices,*

(ii) $\exp(o(\sqrt[48]{n}))$ *if* $x \neq 1$ *and* $y \in \{0, -1\}$ *for simple input graphs.*

Proof. Let G be a connected, simple, planar graph with n vertices and $m \in O(n)$ edges (by Euler's equation), and suppose we want to compute the coefficients of the mth degree polynomial $w \mapsto Z(G; q, w)$. Recall that for $q \neq 0$ the problem *Planar Tutte*(x, y) is computationally equivalent to evaluation of $Z(G; q, w)$ at $w = y - 1$ for planar graphs G.

(i) For $y \notin \{0, -1\}$, corresponding to $w \notin \{-1, -2\}$, we use thickening and interpolation. By (4.19) a k-thickening for the multivariate Tutte polynomial corresponds to the weight shift $w \mapsto (1 + w)^k - 1$, so we get distinct weight shifts for distinct values of k for the given restriction on w. Thus we could apply an algorithm for *Planar Tutte*(x, y) to input graphs G_k for $k = 1, \ldots, m + 1$, to get $m + 1$ distinct evaluations $Z(G_k; q, w) = Z(G; q, w_k)$, and then the coefficients of $w \mapsto Z(G; q, w)$ by interpolation. As G_k has n vertices and $O(m^2) \subset O(n^2)$ edges for all $k \leq m + 1$, the result follows from Lemma 6.7.

(ii) For $y \in \{0, -1\}$, corresponding to $w \in \{-1, -2\}$ we use the same argument with stretches instead of thickenings. Note that since G is simple and planar, the graph G^k is also simple and planar with $O(m^2) = O(n^2)$ vertices for any $k \leq m+1$. By (4.18), a k-stretch for the multivariate Tutte polynomial corresponds to a weight shift $w \mapsto w'$, where $(1 + q/w') = (1 + q/w)^k$. We must ensure that distinct values of k gives gives distinct weight shifts. For $w = -1$ this is true provided $q \notin \{1, 2\}$, which holds by assumption. For $w = -2$ this is true provided $q \notin \{2, 4\}$, which also holds by assumption since $(w, q) = (-2, 4)$ corresponds to the point $(x, y) = (-1, -1)$, which is a trivial point in the Tutte plane and thus excluded by assumption. The result then follows from Lemma 6.7. □

6.5 Discussion

From Theorem 6.1 we have an asymptotically tight lower bound for the problem *Planar Tutte*(x, y) for points on the hyperbola H_3. Compared to this result, the bounds for other regions of the plane provided by Theorem 6.2 certainly leave substantial gaps to the $O(\sqrt{n})$-exponential upper bound. There is no apparent reason why points on the hyperbola H_3 should present much harder problems than points on any other curve in the Tutte plane, so it seems likely that the lower bounds of Theorem 6.2 can be significantly improved. In this section we look at some approaches to this problem, as well as obstacles that were encountered.

6.5.1 The chromatic line

On the *chromatic line* $y = 0$, the Tutte polynomial specializes to the chromatic polynomial $\chi(G; x)$ defined in (4.3). For points on this line we only have a lower bound exponential in $\Omega(\sqrt[48]{n})$ under #ETH from Theorem 6.2(iii), except for the point $(-2, 0)$ where a tight lower bound exponential in $\Omega(\sqrt{n})$ follows from Theorem 6.1. For both multigraphs and simple graphs this one point $(-2, 0)$ determines the exponential-time complexity of any other point on the chromatic line, by a well-known reduction due to Linial [75], based on the following identity:

Theorem 6.6 (Linial [75])**.** *Let $i \in \{1, 2, \dots\}$ and let r be any real number. Then*

$$\chi(G + K_i; r) = r(r-1)\dots(r-i+1) \cdot \chi(G; r-i)\,, \tag{6.15}$$

where $G + K_i$ is the graph consisting of G and a clique K_i on i vertices, each of which is adjacent to every vertex of G.

For planar graphs we cannot use this reduction, as the graphs $G + K_i$ are very nonplanar in general. It would be interesting to find a planarity-preserving reduction for the chromatic line.

The status of line is particularly intriguing as there can only be a finite number of points on it where the tight lower bound does not hold, considering the remark in Section 4.3.2.

6.5.2 Coefficients via *#3-Terminal Min-Cut*

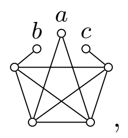
The lower bounds of Theorem 6.2 (iii) and (iv) are particularly far from the upper bound. This is due to the $\exp(\Omega(\sqrt[24]{n}))$-time lower-bound of Lemma 6.7 for computing coefficients of $w \mapsto Z(G; q, w)$ for simple planar graphs. We can compare this to Lemma 5.3 of the previous chapter, which gives a lower bound of $\exp(\Omega(m))$ under #ETH, for the same problem for general simple graphs. That bound, as well as the approximation-hardness result of [48], rests ultimately on the hardness of the following problem

Simple #3-Terminal Min-Cut

Input:	A simple graph $G = (V, E)$ with n vertices, m edges and terminals $t_1, t_2, t_3 \in V$.
Task:	Find the number of cuts of minimum size such that t_1, t_2 and t_3 are in distinct components.

Dahlhaus *et al.* [26] showed that the decision version of this problem is NP-complete, and a lower bound of $\exp(\Omega(m))$ for the counting version under #ETH follows in [29, Theorem C.1]. However, Dahlhaus *et al.* also showed that the decision problem *Planar 3-Terminal Min-Cut* is in P, which leaves the status of the counting version open for planar graphs. It may very well be in FP.

Moreover, even if *Simple #3-Terminal Min-Cut* would be hard also for planar graphs, the reduction to the coefficient problem used for Lemma 5.3 is not planarity preserving. The first step of this reduction is the proof of [30, Lemma 1], where three new edges are added to the input graph G to *Simple #3-Terminal Min-Cut.* Then, for Lemma 5.3, these newly added edges would be removed again using deletions and contractions, and this does not necessarily make the resulting graphs planar even if G was. For example, the following graph is planar,



but if we add edges ab, bc and ac, and then contract them, we get the nonplanar graph K_5.

6.5.3 An alternative starting point

While Goldberg and Jerrum used the problem *Simple #3-Terminal Min-Cut* as the the starting point for the approximation-hardness results in [48], concerning general graphs, they chose a different problem for the planar version in [49], namely

Δ3-Planar Vertex Cover

Input:	A simple, planar graph G of maximum degree three, with n vertices, m edges, and a number $k \in \mathbf{N}$.
Task:	Decide whether G has a vertex cover of size at most k.

Except for the hyperbola H_3, all other regions covered in [49] are treated by the same, somewhat elaborate, reduction from this problem. As it stands, that construction increases the number of vertices by an unspecified polynomial factor, and since no fixed degree can be given on the polynomial expansion, no concrete superpolynomial lower bound follows. So far, no alternative reduction from the same problem has been found. However, as the following result shows, such a reduction would indeed provide some superpolynomial lower bound under ETH.

Theorem 6.7. *Assuming ETH, there is no* $\exp(o(\sqrt{n}))$*-time algorithm for* Δ3-Planar Vertex Cover.

This result follows from Lemma 6.8 – 6.10 below. The starting point is the following hardness result for the nonplanar version of the problem by Cai and Juedes [19].

Lemma 6.8 (Theorem 4.2 in [19]). *Assuming ETH, there is no* $\exp(o(n))$*-time algorithm for* Δ3-Vertex Cover.[5]

We then use a reduction from *Δ3-Vertex Cover* to *Planar Vertex Cover* given in the same paper. In fact, the construction never produces vertices of degree more than 6. We state this stronger result here, and clarify some details of the proof for completeness.

Lemma 6.9 (Lemma 5.1 in [19]). *There is a mapping reduction from* Δ3-Vertex Cover *to* Δ6-Planar Vertex Cover, *mapping graphs G of maximum degree three and n vertices to planar graphs G' of maximum degree six and $O(n^2)$ vertices.*

[5]In [19] the bound is stated in terms of the size k of the vertex cover, but the given proof is for the stronger claim with $k = n$.

Proof. Given a graph G of maximum degree three, consider a drawing of G in the plane such that at most two edges cross at any given point. We construct a planar graph G' by inserting a small copy of the graph X shown in Figure 6.6 at any point in in the drawing where two edges intersect, as demonstrated in Figure 6.7, so that no new edge intersections are formed.

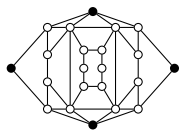


Figure 6.6: The crossover gadget X, with connection vertices in black.

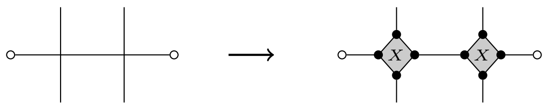


Figure 6.7: Example demonstrating the proper insertion of the crossover gadget X.

It is shown in [45] that the graph G has a vertex cover of size k if, and only if, G' has a vertex cover of size $k' = k + 13c(G)$, where $c(G)$ is the number of edge-intersections in the drawing of G. Since G has maximum degree three, it follows that $c(G) \leq (3n/2)^2$, so the number of vertices in G' is $n' = n + 22c(G) \in O(n^2)$. From the fact that G has maximum degree three and X has maximum degree six, with connection vertices of maximum degree four, it follows that G' will have maximum degree six. □

Finally, we consider [43, Lemma 1], which gives a reduction from *Planar Vertex Cover* to $\Delta 3$-*Planar Vertex Cover* of quadratic vertex expansion. By a very slight adjustment, we can use this to reduce from $\Delta 6$-*Planar Vertex Cover* instead (in fact, any degree-bounded version would do), with only linear vertex expansion, and Theorem 6.7 follows.

Lemma 6.10. *For any fixed $d > 3$, there is a mapping reduction from Δd-*Planar Vertex Cover *to $\Delta 3$-*Planar Vertex Cover*, mapping graphs G of maximum degree d and n vertices to planar graphs G' of maximum degree 3 and $O(dn)$ vertices.*

Proof. We follow the construction of [43, Lemma 1]. Given a planar graph $G = (V, E)$ of maximum degree d and n vertices, consider a fixed planar drawing of G. Pick $v \in V$, and let $u_1, \ldots, u_s \in V$ be the neighbors of v in clockwise order. Expand v into an alternating red-black cycle C_v of length $2d$ (instead of length n as proposed in [43]), as demonstrated in Figure 6.8, so that u_i and u_{i+1} become neighbors of consecutive black vertices in C_v. Add a vertex z_v in the center of C_v, and an edge between z_v and one of the red vertices in C_v, which we denote by r_v.

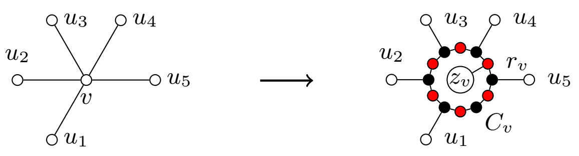


Figure 6.8: Expansion of a vertex v of degree 5 in a graph of maximum degree $d = 6$.

Repeat this for every $v \in V$. The resulting graph $G' = (V', E')$ is planar, of maximum degree three, and with $|V'| = (2d+1)n$. Moreover, if G has a vertex cover $\mathcal{C}$ of size k then G' has the following vertex cover of size $k' = k + dn$, with C_v^B resp. C_v^R denoting the set of black resp. red vertices in C_v,

$$\mathcal{C}' = \{r_v \,:\, v \in \mathcal{C}\} \cup \{C_v^B \,:\, v \in \mathcal{C}\} \cup \{C_v^R \,:\, v \notin \mathcal{C}\}\,.$$

Conversely, suppose G' has a vertex cover $\mathcal{C}'$ of size $k' = k + dn$. We can assume w.l.o.g. that $\mathcal{C}'$ contains the vertex r_v for all $v \in V$; otherwise there is another vertex cover of at most the same size with this property. As every edge in G corresponds to an edge in G' with black end vertices, the following gives a vertex cover in G,

$$\mathcal{C} = \{v \in V \,:\, C_v^B \cap \mathcal{C}' \neq \emptyset\}\,.$$

To estimate the size of this set, we note that $\mathcal{C}'$ must contain at least d vertices from every cycle C_v, i.e. $|C_v \cap \mathcal{C}'| \geq d$. Together with the fact than

$$k + dn = |\mathcal{C}'| = \sum_{v \in V} |C_v \cap \mathcal{C}'|\,,$$

it follows that all except at most k cycles C_v has exactly d vertices in $\mathcal{C}'$. Since $r_v \in \mathcal{C}'$, this means that all except at most k cycles C_v contribute only red vertices to $\mathcal{C}'$. Thus $|\mathcal{C}| \leq k$, which concludes the proof. □

Chapter 7

Open Problems

In Chapter 3, a $2^k n^{O(1)}$-time Monte Carlo algorithm was presented for the *K-Cycle* problem. This algorithm can find a *shortest* solution of given parity, but more than that we cannot specify the length of the output. This could be something for future investigations. It would also be interesting to know how close to this time bound a deterministic algorithm could get, as the current best deterministic algorithm is still doubly exponential in k^{10} [62]. Finally, it would of course be nice to find a relevant application of this result.

As for the lower bounds in the Tutte plane, the line $y = 1$, except for the point $(1, 1)$, is still left completely open for simple graphs, and even for multigraphs the best we have on this line is the $\exp(\Omega(\sqrt[8]{n}))$-bound from the planar case. For simple graphs, only the chromatic line has a lower bound matching the upper bound of $\exp(O(n))$, all other results being of type $\exp(\Omega(n/\log^k n))$, where k depends on the size expansion of the given reduction. It does not seem unreasonable that this tight bound of the chromatic line could be extended to further points in the plane, or at least that some of the lower bounds could be sharpened by improved reductions. Even more so, the lower bounds of type $\exp(\Omega(\sqrt[k]{n}))$ reported for the planar case are for most points very far from the upper bound $\exp(O(\sqrt{n}))$, as discussed in Section 6.5.

Appendix A

List of named problems

Below is a list of the statements of computational problems mentioned in the thesis. Other problems are variations or specializations of these problems, according to the following conventions, where Q denotes a given problem:

#Q	is the standard counting version of Q (a decision problem),
Simple Q	is the restriction of Q to simple input graphs,
Planar Q	is the restriction of Q to planar input graphs,
Cubic Q	is the restriction of Q to cubic input graphs, and
Δd-*Q*	is the restriction of Q to input graphs of maximum degree d.

3-Coloring

Input:	A graph G with n vertices and m edges.
Task:	Decide whether G has a proper 3-coloring.

3-Sat

Input:	A 3-CNF formula φ with n variables and m clauses.
Task:	Decide whether φ is satisfiable.

3-Terminal Min-Cut

Input:	A graph $G = (V, E)$ with n vertices, m edges, three terminals $t_1, t_2, t_3 \in V$ and a number $k \in \mathbf{N}$.
Task:	Decide whether there is a cut of size k such that t_1, t_2 and t_3 are in distinct components.

Clique

Input:	A graph G with n vertices, and a number $k \in \mathbf{N}$.
Task:	Decide whether G contains a clique of k vertices.

Disjoint Paths

Input:	A simple graph $G = (V, E)$ with n vertices, and k vertex pairs $(s_1, t_1), \dots, (s_k, t_k)$.
Task:	Decide whether G contains k mutually disjoint paths $P_1, \dots, P_k$ such that P_i connects s_i to t_i.

Disjoint Factors

Input:	A finite string s of digits from $\{1, 2, \dots, k\}$.
Task:	Decide whether there are k disjoint substrings $s_1, \dots, s_k$ of s such that s_i starts and end with the digit i.

Exact Hitting Set

Input:	A hypergraph $\mathcal{H}$ consisting of m subsets of a finite set V.
Task:	Decide whether there is a subset $S \subset V$ such that $\lvert S \cap E \rvert = 1$ for all $E \in \mathcal{H}$.

Hamiltonian Cycle (Path)

Input:	A graph G with n vertices and m edges.
Task:	Decide whether G contains a Hamiltonian cycle (path).

Hitting Set

Input:	A hypergraph $\mathcal{H}$ consisting of m subsets of a finite set V.
Task:	Decide whether there is a subset $S \subset V$ such that $\lvert S \cap E \rvert \neq 0$ for all $E \in \mathcal{H}$.

Independent Set

Input:	A graph G with n vertices, and a number $k \in \mathbf{N}$.
Task:	Decide whether G contains an independent set of k vertices.

K-Cycle

Input:	A simple graph $G = (V, E)$ with n vertices, and a subset $K \subseteq V \cup E$ of size k.
Task:	Decide whether G contains a K-cycle.

k-Cycle (-Path)

Input:	A graph G with n vertices and m edges, and a number $k \in \mathbf{N}$.
Task:	Decide whether G contains a cycle (path) of length k.

l-K-Hole

Input:	A simple graph $G = (V, E)$ with n vertices, a subset $K \subset V$ of size $k = (\log n)^{O(1)}$, and a number $l \geq 3$.
Task:	Decide whether G contains an induced cycle of length l, containing all vertices in K.

Monochrome K-Cycle

Input:	An simple, edge-colored graph $G = (V, E)$ with n vertices, and a subset $K \subseteq V \cup E$ of size k.
Task:	Decide whether G contains a *monochrome* K-cycle.

Steiner Tree

Input:	A graph $G = (V, E)$ with n vertices, a set of s terminals $t_1, \ldots, t_s \in V$, and a number $k \in \mathbf{N}$.
Task:	Decide whether G contains a tree connecting all terminals and including at most k other vertices.

Tutte(x, y)

Input:	A multigraph G with n vertices and m edges.
Task:	Compute $T(G; x, y)$, as defined in (4.2).

Path Repiability(p)

Input:	A graph $G = (V, E)$ with n vertices, m edges, and terminals $s, t \in V$.
Task:	Compute $R_{s,t}(G; p)$, as defined in (6.7).

Vertex Cover

Input:	A simple graph G with n vertices, m edges, and $k \in \mathbf{N}$.
Task:	Decide whether G has a vertex cover of size at most k.

Bibliography

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[2] M. O. Ball, C. J. Colbourn, and J. S. Provan. Network Reliability. In *Network Models*, volume 7 of *Handbooks in operations research and management science*, pages 673–762. Elsevier Science, 1995.

[3] R. Barbanchon. On unique graph 3-colorability and parsimonious reductions in the plane. *Theor. Comput. Sci.*, 319(1–3):455–482, 2004.

[4] E. T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Inf. Process. Lett.*, 47(4):203–207, 1993.

[5] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[6] A. Björklund. Determinant sums for undirected Hamiltonicity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 173–182. IEEE Computer Society, 2010.

[7] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *Proceedings of the 47th annual IEEE Symposium on Foundations of Computer Science, FOCS 2008*, pages 677–686, 2008.

[8] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.

[9] A. Björklund, T. Husfeldt, and N. Taslaman. Shortest cycle through specified elements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 1747–1753. SIAM, 2012.

[10] M. Bläser, H. Dell, and J. A. Makowsky. Complexity of the bollobás-riordan polynomial. exceptional points and uniform reductions. *Theor. Comput. Syst.*, 46(4):690–706, 2010.

[11] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Revised Selected Papers*, volume 5917 of *Lect. Notes Comput. Sci.*, pages 17–37, 2009.

[12] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

[13] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 165–176, 2011.

[14] H. L. Bodlaender, S. Thomassé, and A. Yeo. Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical report, Department of Information and Computing Sciences, Utrecht University, 2008.

[15] H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

[16] J. Bondy and U. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, 2008.

[17] T. Brylawski. The Tutte polynomial. I. General theory. In *Matroid theory and its applications*, C.I.M.E. Summer Schools, pages 125–275. C.I.M.E., Ed. Liguori, Napoli & Birkhäuser, 1982.

[18] J. A. Buzacott. A recursive algorithm for finding reliability measures related to the connection of nodes in a graph. *Networks*, 10(4):311–327, 1980.

[19] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):789–807, 2003.

[20] M. Chudnovsky, G. Cornuéjols, X. Liu, P. D. Seymour, and K. Vuskovic. Recognizing berge graphs. *Combinatorica*, 25(2):143–186, 2005.

[21] M. Chudnovsky, K. Kawarabayashi, and P. D. Seymour. Detecting even holes. *J. Graph Theor.*, 48(2):85–111, 2005.

[22] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006.

[23] M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vuskovic. Even-hole-free graphs part i: Decomposition theorem. *J. Graph Theor.*, 39(1):6–49, 2002.

[24] M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vuskovic. Even-hole-free graphs part ii: Recognition algorithm. *J. Graph Theor.*, 40(4):238–266, 2002.

[25] H. H. Crapo. The Tutte polynomial. *Aequationes Mathematicae*, 3:211–229, 1969.

[26] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[27] N. Dean. Open problems. In *Graph Structure Theory: Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference 1991*, volume 147 of *Contemporary Mathematics*, pages 677–688. AMS, 1991.

[28] V. G. Deineko, B. Klinz, and G. J. Woeginger. Exact algorithms for the Hamiltonian cycle problem in planar graphs. *Oper. Res. Lett.*, 34(3):269–274, 2006.

[29] H. Dell, T. Husfeldt, D. Marx, N. Taslaman, and M. Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Trans. Algorithms*, to appear.

[30] H. Dell, T. Husfeldt, and M. Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Proceedings, Part I*, volume 6198 of *Lect. Notes Comput. Sci.*, pages 426–437. Springer, 2010.

[31] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and ids. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Proceedings, Part I*, volume 5555 of *Lect. Notes Comput. Sci.*, pages 378–389. Springer, 2009.

[32] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[33] A. Drucker. New limits to classical and quantum instance compression. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, Lect. Notes Comput. Sci. Springer, 2012. To appear.

[34] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[35] M. R. Fellows. The Robertson-Seymour theorems: A survey of applications. In *Graphs and Algorithms: Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference 1989*, volume 89 of *Contemporary Mathematics*, pages 1–18. AMS, 1989.

[36] H. Fleischner and G. J. Woeginger. Detecting cycles through three fixed vertices in a graph. *Inf. Process. Lett.*, 42(1):29–33, 1992.

[37] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[38] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

[39] C. M. Fortuin and P. W. Kasteleyn. Phase transitions in lattice systems with random local properties. *J. Phys. Soc. Japan*, 26(Suppl.):11–14, 1969.

[40] C. M. Fortuin and P. W. Kasteleyn. On the random-cluster model: I. Introduction and relation to other models. *Physica*, 57(4):536–564, 1972.

[41] S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[42] A. Frank. Packing paths, cuts and circuits - a survey. In *Paths, Flows, and VLSI-Layout*, volume 9 of *Algorithms and Combinatorics*, pages 49–100. Springer, 1990.

[43] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32:826–834, 1977.

[44] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[45] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[46] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.

[47] C. Godsil and G. Royle. *Algebraic Graph Theory*. Graduate Texts in Mathematics. Springer, 2001.

[48] L. A. Goldberg and M. Jerrum. Inapproximability of the Tutte polynomial. *Information and Computation*, 206(7):908–929, 2008.

[49] L. A. Goldberg and M. Jerrum. Inapproximability of the Tutte polynomial of a planar graph. *CoRR*, abs/0907.1724, 2009.

[50] L. A. Goldberg and M. Jerrum. The complexity of computing the sign of the Tutte polynomial (and consequent #P-hardness of approximation). *CoRR*, abs/1202.0313, 2012.

[51] R. Haas and M. Hoffmann. Chordless paths through three vertices. *Theor. Comput. Sci.*, 351(3):360–371, 2006.

[52] G. Haggard, D. J. Pearce, and G. Royle. Computing Tutte polynomials. *ACM Trans. Math. Softw.*, 37(3), 2010.

[53] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[54] C. Hoffmann. Exponential time complexity of weighted counting of independent sets. In *Proceedings of the 5th International Symposium on Parameterized and Exact Complexity, IPEC 2010*, volume 6478 of *Lect. Notes Comput. Sci.*, pages 180–191. Springer, 2010.

[55] T. Husfeldt. Invitation to algorithmic uses of inclusion-exclusion. In *Proceedings of the 38th international conference on Automata, languages and programming, ICALP 2011, Part II*, volume 6756 of *Lect. Notes Comput. Sci.*, pages 42–59. Springer, 2011.

[56] T. Husfeldt and N. Taslaman. The exponential time complexity of computing the probability that a graph is connected. In *Proceedings of the 5th International Symposium on Parameterized and Exact Complexity, IPEC 2010*, volume 6478 of *Lect. Notes Comput. Sci.*, pages 192–203. Springer, 2010.

[57] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[58] S. Istrail. Statistical mechanics, three-dimensionality and NP-completeness: I. Universality of intractability for the partition function of the Ising model across non-planar lattices. In *Proceedings of the 32nd annual ACM Symposium on Theory of Computing, STOC 2000*, pages 87–96. ACM, 2000.

[59] F. Jaeger, D. L. Vertigan, and D. J. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Math. Proc. Cambridge Philos. Soc.*, 108(1):35–53, 1990.

[60] M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.

[61] R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1(2):49–51, 1982.

[62] K. Kawarabayashi. An improved algorithm for finding cycles through elements. In *Proceedings of Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008*, pages 374–384, 2008.

[63] K. Kawarabayashi and Y. Kobayashi. Algorithms for finding an induced cycle in planar graphs. *Combinatorica*, 30(6):715–734, 2010.

[64] K. Kawarabayashi, Z. Li, and B. A. Reed. Recognizing a totally odd K_4-subdivision, parity 2-disjoint rooted paths and a parity cycle through specified elements. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 318–328, 2010.

[65] G. Kirchhoff. Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird. *Ann. Phys. Chem.*, 72:497–508, 1847.

[66] D. E. Knuth. *Combinatorial Algorithms, Part 1*, volume 4A of *The Art of Computer Programming*. Addison-Wesley Professional, 2011.

[67] S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In *ACM '77 Proceedings of the 1977 annual conference*, pages 294–300. ACM, 1977.

[68] I. Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part I*, volume 5125 of *Lect. Notes Comput. Sci.*, pages 575–586. Springer, 2008.

[69] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Proceedings, Part I*, volume 5555 of *Lect. Notes Comput. Sci.*, pages 653–664. Springer, 2009.

[70] S. Kratsch and M. Wahlström. Preprocessing of min ones problems: A dichotomy. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Proceedings, Part I*, volume 6198 of *Lect. Notes Comput. Sci.*, pages 653–665. Springer, 2010.

[71] K. Kutzkov. New upper bound for the #3-SAT problem. *Inf. Process. Lett.*, 105(1):1–5, 2007.

[72] A. LaPaugh. The subgraph isomorphism problem. Technical report, Massachusetts Institute of Technology, Laboratory for Computer Science TM 99, 1978.

[73] A. S. LaPaugh and R. L. Rivest. The subgraph homeomorphism problem. *J. Comput. Syst. Sci.*, 20(2):133–149, 1980.

[74] B. Lévêque, D. Y. Lin, F. Maffray, and N. Trotignon. Detecting induced subgraphs. *Discrete Appl. Math.*, 157(17):3540–3551, 2009.

[75] N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM J. Algebra. Discr.*, 7(2):331–335, 1986.

[76] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[77] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[78] D. Lokshtanov. Private communication, February 2012.

[79] L. Lovász. On determinants, matchings, and random algorithms. In *Proceedings of Conference on Fundamentals of Computing Theory*, pages 565–574. Akademia-Verlag, 1979.

[80] B. Monien. How to find long paths efficiently. In *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 239–254. North-Holland, 1985.

[81] L. Onsager. Crystal statistics. I. A two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, 1944.

[82] J. Oxley and D. Welsh. The Tutte polynomial and percolation. In *Graph Theory and Related Topics*, pages 329–339. Academic Press, 1979.

[83] J. S. Provan. The complexity of reliability computations in planar and acyclic graphs. *SIAM J. Comput.*, 15(3):694–702, 1986.

[84] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[85] N. Robertson and P. D. Seymour. Graph minors .XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[86] J. Scott, T. Ideker, R. M. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comput. Biol.*, 13(2):133–144, 2006.

[87] K. Sekine, H. Imai, and S. Tani. Computing the Tutte polynomial of a graph of moderate size. In *Algorithms and Computations: 6th International Symposium, ISAAC '95, Proceedings*, volume 1004 of *Lect. Notes Comput. Sci.*, pages 224–233. Springer, 1995.

[88] T. Shlomi, D. Segal, E. Ruppin, and R. Sharan. QPath: a method for querying pathways in a protein-protein interaction network. *BMC Bioinformatics*, 7(1):1–9, 2006.

[89] A. D. Sokal. The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In *Surveys in Combinatorics*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 173–226. Cambridge University Press, 2005.

[90] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[91] W. Tutte. The factorization of linear graphs. *J. Lond. Math. Soc.*, 22:459–474, 1947.

[92] W. Tutte. A contribution to the theory of chromatic polynomials. *Canad. J. Math.*, 6:80–91, 1953.

[93] W. Tutte. Graph-polynomials. *Advances in Applied Mathematics*, 32(1–2):5–9, 2004.

[94] D. L. Vertigan. *On the Computational Complexity of Tutte, Jones, Homfly and Kauffman Invariants.* PhD thesis, University of Oxford, 1991.

[95] D. L. Vertigan. The computational complexity of Tutte invariants for planar graphs. *SIAM J. Comput.*, 35(3):690–712, 2005.

[96] H. S. Wilf. *Algorithms and complexity.* Prentice Hall, 1986.

[97] R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.