Computing the Tutte Polynomial in Vertex-Exponential Time

Andreas Björklund[1] Thore Husfeldt[1,2] Petteri Kaski[3] Mikko Koivisto[3]

Abstract

The deletion–contraction algorithm is perhaps the most popular method for computing a host of fundamental graph invariants such as the chromatic, flow, and reliability polynomials in graph theory, the Jones polynomial of an alternating link in knot theory, and the partition functions of the models of Ising, Potts, and Fortuin–Kasteleyn in statistical physics. Prior to this work, deletion–contraction was also the fastest known general-purpose algorithm for these invariants, running in time roughly proportional to the number of spanning trees in the input graph.

Here, we give a substantially faster algorithm that computes the Tutte polynomial—and hence, all the aforementioned invariants and more—of an arbitrary graph in time within a polynomial factor of the number of connected vertex sets. The algorithm actually evaluates a multivariate generalization of the Tutte polynomial by making use of an identity due to Fortuin and Kasteleyn. We also provide a polynomial-space variant of the algorithm and give an analogous result for Chung and Graham's cover polynomial.

1 Introduction

Tutte's motivation for studying what he called the "dichromatic polynomial" was algorithmic. By his own entertaining account [40], he was intrigued by the variety of graph invariants that could be computed with the deletion–contraction algorithm, and "playing" with it he discovered a bivariate polynomial that we can define as

$$T_G(x,y) = \sum_{F\subseteq E} (x-1)^{c(F)-c(E)}(y-1)^{c(F)+|F|-|V|}. \qquad (1)$$

Here, G is a graph with vertex set V and edge set E; by $c(F)$ we denote the number of connected components in the graph with vertex set V and edge set F. Later, Oxley and Welsh [35] showed in their celebrated Recipe Theorem that, in a very strong sense, the *Tutte polynomial* T_G is indeed the most general graph invariant that can be computed using deletion–contraction.

Since the 1980s it has become clear that this construction has deep connections to many fields outside of computer science and algebraic graph theory. It appears in various guises and specialisations in enumerative combinatorics, statistical physics, knot theory, and network theory. It subsumes the chromatic, flow, and reliability polynomials, the Jones polynomial of an alternating link, and, perhaps most importantly, the models of Ising, Potts, and Fortuin–Kasteleyn, which appear in tens of thousands of research papers. A number of surveys written for various audiences present and explain these specialisations [38, 42, 43, 44].

Computing the Tutte polynomial has been a very fruitful topic in theoretical computer science, resulting in seminal work on the computational complexity of counting, several algorithmic breakthroughs both classical and quantum, and whole research programmes devoted to the existence and nonexistence of approximation algorithms. Its specialisation to graph colouring has been one of the main benchmarks of progress in exact algorithms.

The deletion–contraction algorithm computes T_G for a connected G in time within a polynomial factor of $\tau(G)$, the number of spanning trees of the graph, and no essentially faster algorithm was known. In this paper we show that the Tutte polynomial—and hence, by virtue of the Recipe Theorem, *every* graph invariant admitting a deletion–contraction recursion—can be computed in time within a polynomial factor of $\sigma(G)$, the number of vertex subsets that induce a connected subgraph. Especially, the algorithm runs in time $\exp(O(n))$, that is, in "vertex-exponential" time, while $\tau(G)$ typically is $\exp(\omega(n))$ and can be as large as n^{n-2} [12].

1.1 Result and consequences

By "computing the Tutte polynomial" we mean computing the coefficients t_{ij} of the monomials $x^i y^j$ in $T_G(x,y)$ for a

[1]Lund University, Department of Computer Science, P.O.Box 118, SE-22100 Lund, Sweden.

[2]IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark.

[3]Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, P.O.Box 68, FI-00014 University of Helsinki, Finland.

[4]This research was supported in part by the Academy of Finland, Grants 117499 (P.K.) and 109101 (M.K.) and by the Swedish Research Council, project "Exact Algorithms" (A.B., T.H.).

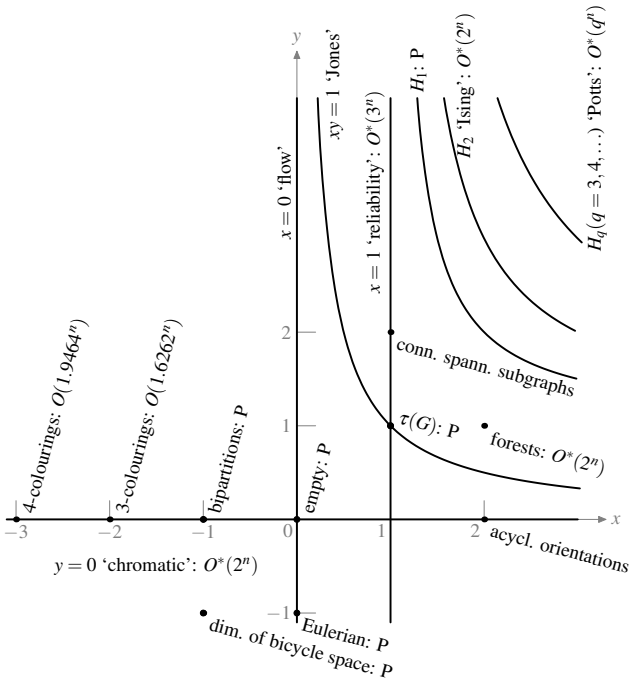

Figure 1. Tutte plane with prior complexities.

graph G given as input. Of course, the coefficients also enable the efficient evaluation of $T_G(x, y)$ at any given point (x, y). Our main result is as follows.

Theorem 1 *The Tutte polynomial of an n-vertex graph G can be computed*

(a) in time and space $O^*(\sigma(G))$;

(b) in time $O^*(3^n)$ and polynomial space; and

(c) in time $O^*(3^{n-s}2^s)$ and space $O^*(2^s)$ for any integer s, $0 \le s \le n$.

Especially, the Tutte polynomial can be evaluated everywhere in vertex-exponential time. In some sense, this is both surprising and optimal, a claim that we solidify under the Exponential Time Hypothesis in §2.5.

Prescribing the point of evaluation (x, y) leads to the more restricted task of computing the value $T_G(x, y)$ for a given G. In this setting vertex-exponential (or faster) algorithms were known before for a number of points (x, y); see Figure 1. The hyperbolas H_q are defined by $(x-1)(y-1) = q$, only their positive branches are drawn. The points $(0, 0)$, $(-1, 0)$, $(0, -1)$, $(-1, -1)$, $(1, 1)$ and the hyperbola H_1 are in P, all other points are #P-complete. Those points and lines where algorithms with complexity $\exp(O(n))$ were previously known (sometimes only in exponential space), are labelled with their running time; the hyperbolas H_q were known to be vertex-exponential only for positive integer q. See §2.3 for references. The only points that are known to admit algorithms with better bounds than our main result are the "colouring" points $(-2, 0)$ and $(-3, 0)$, the "Ising" hyperbola H_2, for which a faster algorithm in observed in §2.3, and of course the points in P.

For bounded-degree graphs G, the deletion–contraction algorithm itself runs in vertex-exponential time because $\tau(G) = \exp(O(n))$. Theorem 1 still gives a better bound because it is known that $\sigma(G) = O((2-\epsilon)^n)$ for bounded degree [7, Lemma 6], while $\tau(G)$ grows faster than 2.3^n already for 3-regular graphs (see §2.4). The precise bound is as follows:

Corollary 2 *The Tutte polynomial of an n-vertex graph with maximum vertex degree Δ can be computed in time $O^*(\xi_\Delta^n)$, where $\xi_\Delta = (2^{\Delta+1} - 1)^{1/(\Delta+1)}$.*

The question about solving deletion–contraction based algorithmic problems in vertex-exponential time makes sense in *directed* graphs as well. Here, the most successful attempt to define an analogue of the Tutte polynomial is Chung and Graham's *cover polynomial*, which satisfies directed analogues to the deletion–contraction operations [13].

Let D be a digraph, possibly with parallel edges and loops. Denote by $c_D(i, j)$ the number of ways of disjointly covering all the vertices of D with i directed paths and j directed cycles. The *cover polynomial* of D is defined as

$$C_D(x, y) = \sum_{i,j} c_D(i, j) x^{\underline{i}} y^j ,$$

where $x^{\underline{i}} = x(x-1)\cdots(x-i+1)$ and $x^{\underline{0}} = 1$. It is known that $C_D(x, y)$ is #P-complete to evaluate except at a handful of points (x, y) [9].

In analogy to Theorem 1, we can show that C_D can be computed in vertex-exponential time:

Theorem 3 *The cover polynomial of an n-vertex directed graph can be computed*

(a) in time and space $O^*(2^n)$; and

(b) in time $O^*(3^n)$ and polynomial space.

1.2 Overview of techniques

The Tutte polynomial is, in essence, a sum over connected spanning subgraphs. Managing this connectedness property introduces a computational challenge not present with its specialisations, e.g., with the chromatic polynomial. Neither the dynamic programming algorithm across vertex subsets by Lawler [33] nor the recent inclusion–exclusion algorithm [8], which apply for counting k-colourings, seems

to work directly for the Tutte polynomial. Perhaps surprisingly, they do work for the cover polynomial, even though the application is quite involved; the details are in §5 and can be seen as an attempt to explain just how far these concepts get us.

For the Tutte polynomial, we take a detour via the Potts model. The idea is to evaluate the partition function of the q-state Potts model at suitable points using inclusion–exclusion, which then, by a neat identity due to Fortuin and Kasteleyn [17, 38], enables the evaluation of the Tutte polynomial at any given point by polynomial interpolation. Finally, another round of polynomial interpolation yields the desired coefficients of the Tutte polynomial. Each step can be implemented using only polynomial space. Moreover, the approach readily extends to the multivariate Tutte polynomial of Sokal [38] which allows the incorporation of arbitrary edge weights; that generalisation can be communicated quite concisely using the involved high-level framework, which we do in §3. To finally arrive at the main result of this paper—reducing the running time to within a polynomial factor of $\sigma(G)$—requires manipulation at the level of the fast Moebius transform "inside" the algorithm, which can be found in §3.3. The smooth time–space tradeoff, Theorem 1(c), is obtained by a new "split transform" technique.

Our approach highlights the algorithmic significance of the Fortuin–Kasteleyn identity, and suggests a more general technique: to compute a polynomial, it may be advisable to look at its evaluations at integral (or otherwise special) points, with the objective of obtaining new combinatorial or algebraic interpretations that then enable faster reconstruction of the entire polynomial. (For example, the multiplication of polynomials via the fast Fourier transform can be seen as an instantiation of this technique.)

We also give another vertex-exponential time algorithm that does not rely on interpolation (§4). It is based on a new recurrence formula that alternates between partitioning an induced subgraph into components and a subtraction step to solve the connected case. The recurrence can be solved using fast subset convolution [6] over a multivariate polynomial ring. However, an exponential space requirement seems inherent to that algorithm.

1.3 Conventions

For standard graph-theoretic terminology we refer to West [45]. All graphs we consider are undirected and may contain multiple edges and loops. For a graph G, we write $n = n(G)$ for the number of vertices, $m = m(G)$ for the number of edges, $V = V(G) = \{1, 2, \ldots, n\}$ for the vertex set, $E = E(G)$ for the edge set, $c = c(G)$ for the number of connected components, $\tau(G)$ for the number of spanning trees, and $\sigma(G)$ the *number of connected sets*, i.e., the number of vertex subsets that induce a connected graph.

To simplify running time bounds, we use the notation O^* to suppress a polynomial factor (always in n), so $O^*(f(r))$ means $O(f(r)n^k)$ for some constant k. We also assume m is bounded by a polynomial in n and remark that this assumption is implicit already in Theorem 1. (Without this assumption, all the time bounds require an additional multiplicative term polynomial in m.) For a set of vertices $U \subseteq V(G)$, we write $G[U]$ for the subgraph induced by U in G. A subgraph H of G is *spanning* if $V(H) = V(G)$. For a proposition P, we use Iverson's bracket notation $[P]$ to mean 1 if P is true and 0 otherwise.

2 Prior algorithms for the Tutte polynomial

The direct evaluation of $T_G(x, y)$ based on (1) takes $O^*(2^m)$ steps and polynomial space, but many other expansions have been studied in the literature.

2.1 Spanning tree expansion

If we expand and collect terms in (1) we arrive at

$$T_G(x, y) = \sum_{i,j} t_{ij} x^i y^j \,. \tag{2}$$

In fact, this is Tutte's original definition. The coefficients t_{ij} of this expansion are well-studied: assuming that G is connected, t_{ij} is the number of spanning trees of G having "internal activity" i and "external activity" j. What these concepts mean need not occupy us here (for example, see [4, §13]), for our purposes it is sufficient to know that they can be efficiently computed for a given spanning tree. Thus (2) can be evaluated directly by iterating over all spanning trees of G, which can be accomplished with polynomial delay [27]. The resulting running time is within a polynomial factor of $\tau(G)$.

Some of the coefficients t_{ij} have an alternative combinatorial interpretation, and some can be computed faster than others. For example, $t_{00} = 0$ holds if $m > 0$, and $t_{01} = t_{10}$ if $m > 1$. The latter value, the *chromatic invariant* $\theta(G)$, can be computed from the chromatic polynomial, and thus can be found in time $O^*(2^n)$ [8].

The computational complexity of computing individual coefficients t_{ij} has also been investigated. In particular, polynomial-time algorithms exist for $t_{n-1-k,j}$ for constant k and all $j = 0, 1, \ldots, m - n + 1$. In general, the task of computing t_{ij} is #P-complete [2].

2.2 Deletion–contraction

The classical algorithm for computing T_G is the following *deletion–contraction* algorithm. It is based on two graph transformations involving an edge e. The graph $G\backslash e$ is obtained from G by *deleting* e. The graph G/e is obtained from

G by *contracting* e, that is, by identifying the endvertices of e and then deleting e.

With these operations, one can establish the recurrence

$$T_G(x,y) = \begin{cases} 1 & \text{if } E = \emptyset; \\ yT_{G\setminus e}(x,y) & \text{if } e \text{ is a loop;} \\ xT_{G/e}(x,y) & \text{if } e \text{ is a bridge;} \\ T_{G\setminus e}(x,y) + T_{G/e}(x,y) & \text{otherwise.} \end{cases}$$

The deletion–contraction algorithm defined by a direct evaluation of the above recurrence leads to a running time that scales as the Fibonacci sequence, $((1+\sqrt{5})/2)^{n+m} = O(1.6180^{n+m})$ [46]. Sekine, Imai, and Tani [37] observed that the corresponding computation tree has one leaf for every spanning tree of G, so the above recurrence is yet another way to evaluate T_G in time within a polynomial factor of $\tau(G)$. In practice one can speed up the computation by identifying isomorphic graphs and using dynamic programming to avoid redundant recomputation [22, 24, 37].

The deletion–contraction algorithm is known to compute many different graph parameters. For example, the number of spanning trees admits an analogous recursion, as does the number of acyclic orientations, the number of colourings, the dimension of the bicycle space, and so forth [20, §15.6–8]. This is no surprise: all these graph parameters are evaluations of the Tutte polynomial at certain points. But not only is every specialisation of T_G expressible by deletion–contraction, the converse holds as well: *every* graph parameter that can be expressed as a deletion–contraction recursion turns out to be a valuation of T_G, according to the Recipe Theorem of Oxley and Welsh [35] (cf. [10, Theorem X.2]).

Besides deletion–contraction, many other expansions are known (in particular for restrictions of the Tutte polynomial; see [4]), even a convolution over the set of edges [32], but none leads to vertex-exponential time.

2.3 Regions of the Tutte plane

The question at which points (x, y) the Tutte polynomial can be computed exactly and efficiently was completely settled in the framework of computational complexity in the seminal paper of Jeager, Vertigan, and Welsh [26]: They presented a complete classification of points and curves where the problem is polynomial-time computable, and where it is #P-complete. This result shows us where we probably need to resign ourselves to a superpolynomial-time algorithm.

For most of the #P-hard points, the algorithms from §2.1 and §2.2 were best known. However, for certain regions of the Tutte plane, faster algorithms were known. We attempt to summarise these algorithms here, including the polynomial-time cases; see Figure 1.

Trivial hyperbola. On the hyperbola $(x-1)(y-1) = 1$ the terms of (1) involving $c(F)$ cancel, so $T_G(x,y) = (x-1)^{n-c}y^m$, which can be evaluated in polynomial time.

Ising model. On the hyperbola $H_2 \equiv (x-1)(y-1) = 2$, the Tutte polynomial gives the partition function of the *Ising model*, a sum of easily computable weights over the 2^n configurations of n two-state spins. This can be trivially computed in time $O^*(2^n)$ and polynomial space. By dividing the n spins into three groups of about equal size and using fast matrix multiplication, one can compute the sum in time $O^*(2^{n\omega/3}) = O(1.732^n)$ and exponential space, where ω is the exponent of matrix multiplication; this is yet a new application of Williams's trick [5, 31, 47].

Potts model. More generally, for any integer $q \geq 2$, the Tutte polynomial on the hyperbola $H_q \equiv (x-1)(y-1) = q$ gives the partition function of the *q-state Potts model* [36]. This is a sum over the configurations of n spins each having q possible states. It can be computed trivially in time $O^*(q^n)$ and, via fast matrix multiplication, in time $O^*(q^{n3/\omega})$. We will show in §3 that, in fact, time $O^*(2^n)$ suffices, which result will be an essential building block in our main construction.

Reliability polynomial. The reliability polynomial $R_G(p)$, which is the probability that no component of G is disconnected after independently removing each edge with probability $1-p$, satisfies $R_G(p) = p^{m-n+c}(1-p)^{n-c}T_G(1,1/p)$ and can be evaluated in time $O^*(3^n)$ and exponential space [11].

Number of spanning trees. For connected G, $T_G(1,1)$ equals the number $\tau(G)$ of spanning trees, and is computable in polynomial time as the determinant of a maximal principal submatrix of the Laplacian of G, a result known as Kirchhoff's Matrix–Tree Theorem.

Number of spanning forests. The number of spanning forests, $T_G(2,1)$, is computable in time $O^*(2^n)$ by first using the Matrix–Tree Theorem for each induced subgraph and then assembling the result one component (that is, tree) at a time via inclusion–exclusion [8]. (This observation is new to the present work, however.)

Dimension of the bicycle space. $T_G(-1,-1)$ computes the dimension of the bicycle space, in polynomial time by Gaussian elimination.

Number of nowhere-zero 2-flows. $T_G(0,-1) = 1$ if G is Eulerian (in other words, it "admits a nowhere-zero 2-flow"), and $T_G(0,-1) = 0$ otherwise. Thus $T_G(0,-1)$ is computable in polynomial time.

Chromatic polynomial. The chromatic polynomial $P_G(t)$, which counts the number of proper t-colourings of the vertices of G, satisfies $P_G(t) = (-1)^{n-c}t^cT_G(1-t,0)$ and can be computed in time $O^*(2^n)$ [8]. Vertex-exponential time algorithms were known at least since Lawler [33], and a vertex-exponential, polynomial-space algorithm was found only recently [5]. Other approaches to the chromatic

polynomial are surveyed by Anthony [3]. At $t = 2$ (equivalently, $x = -1$) this is polynomial-time computable by breadth-first search (every connected component of a bipartite graph has exactly two proper 2-colourings). The cases $t = 3, 4$ are well-studied benchmarks for exact counting algorithms, the current best bounds are $O(1.6262^n)$ and $O(1.9464^n)$ [16]. The case $x = 0$ is trivial.

To the best knowledge of the authors, no algorithms with running time $\exp(O(n))$ have been known for other real points. If we allow x and y to be complex, there are four more points (x, y) at which T_G can be evaluated in polynomial time [26].

2.4 Restricted graph classes

Explicit formulas for Tutte polynomial have been derived for many elementary families of graphs, such as $T_{C_n}(x, y) = y + x + x^2 + \cdots + x^{n-1}$ for the n-cycle graph C_n. We will not give an overview of these formulas here (see [4, §13]); most of them are applications of deletion–contraction.

For well-known graph classes, the authors know the following results achieving $\exp(O(n))$ running time or better:

Planar graphs. If G is planar, then the Tutte polynomial can be computed in time $\exp(O(\sqrt{n}))$ [37]. This works more generally, with a slight overhead: in classes of graphs with separators of size n^α, the Tutte polynomial can be computed in time $\exp(O(n^\alpha \log n))$.

Bounded tree- and branch-width. For k a fixed integer, if G has tree-width k then T_G can be computed in polynomial time [1, 34]. This can be generalised to branch-width [23].

Bounded clique-width and cographs. For k a fixed integer, if G has clique-width k then T_G can be computed in time $\exp(O(n^{1-1/(k+2)}))$ [19]. A special case of this is the class of cographs (graphs without an induced path of 4 vertices), where the bound becomes $\exp(O(n^{2/3}))$.

Bounded-degree graphs. If Δ is the maximum degree of a vertex, the deletion–contraction algorithm and $2m \leq n\Delta$ yield the vertex-exponential running time bound $O(1.6180^{(1+\Delta/2)n})$ directly from the recurrence. Gebauer and Okamoto improve this to $O^*(\chi_\Delta^n)$, where $\chi_\Delta = 2(1 - \Delta 2^{-\Delta})^{1/(\Delta+1)}$ (for example, $\chi_3 = 2.5149$, $\chi_4 = 3.7764$, and $\chi_5 = 5.4989$). For k-regular graphs with $k \geq 3$ a constant independent of n, the number of spanning trees (and hence, within a polynomial factor, the running time of the deletion–contraction algorithm) is bounded by $\tau(G) = O(\nu_k^n n^{-1} \log n)$, where $\nu_k = (k-1)^{k-1}/(k^2 - 2k)^{k/2-1}$ (for example, $\nu_3 = 2.3094$, $\nu_4 = 3.375$, and $\nu_5 = 4.4066$), and this bound is tight [14].

Interval graphs. If G is an interval graph, then T_G can be computed in time $O(1.9706^m)$, which is not $\exp(O(n))$ in general, but still faster than by deletion–contraction [18].

What we cannot survey here is the extensive literature that studies algorithms that simultaneously specialise T_G and restrict the graph classes, often with the goal of developing a polynomial-time algorithm. A famous example is that for Pfaffian orientable graphs, which includes the class of planar graphs, the Tutte polynomial is polynomial-time computable on the hyperbola H_2 [29]. Within computer science, the most studied specialisation of this type is most likely graph colouring for restricted graph classes.

2.5 Computational complexity

The study of the computational complexity of the Tutte polynomial begins with Valiant's theory of #P-completeness [41] and the exact complexity results of Jaeger, Vertigan, and Welsh [26]. The study of the approximability of the values of T_G has been a very fruitful research direction, an overview of which is again outside the scope of this paper. In this regard we refer to Welsh's monograph [42] and to the recent paper of Goldberg and Jerrum [21] for a survey of newer developments.

For our purposes, the most relevant hardness results have been established under the Exponential Time Hypothesis [25] (ETH). First, deciding whether a given graph can be 3-coloured requires $\exp(\Omega(n))$ time under ETH, and since 3-colourability can be decided by computing $T_G(-2, 0)$ we see that evaluating the Tutte polynomial requires vertex-exponential time under ETH. Thus, it would be surprising if our results could be significantly improved, for example to something like $\exp(O(n/\log n))$.

Second, it is by no means clear that the entire Tutte plane should admit such algorithms. Many specialisations of the Tutte polynomial can be understood as constraint satisfaction problems. For example, graph colouring is an instance of $(q, 2)$-CSP, the class of constraint satisfaction problems with pairwise constraints over q-state variables. Similarly, the partition function for the Potts model can be seen as a weighted counting CSP [15]. Very recently, Traxler [39] has shown that already the decision version of $(q, 2)$-CSP requires time $\exp(\Omega(n \log q))$ under ETH, even for some very innocent-looking restrictions, and even for bounded degree graphs. Thus in general, these CSPs are not vertex-exponential under ETH.

3 The multivariate Tutte polynomial via the q-state Potts model

Let R be a multivariate polynomial ring over a field. Associate with each $e \in E$ a ring element $r_e \in R$. The *multivariate Tutte polynomial* [38] of G is the polynomial

$$Z_G(q, r) = \sum_{F \subseteq E} q^{c(F)} \prod_{e \in F} r_e \,, \tag{3}$$

where r denotes a vector with components r_e with $e \in E$, q is an indeterminate, and $c(F)$ denotes the number of con-

nected components in the graph with vertex set V and edge set F. The product over an empty set always evaluates to 1.

The classical Tutte polynomial $T_G(x, y)$ can be recovered from the multivariate polynomial $Z_G(q, r)$ by the identity [38, (2.26)]

$$T_G(x, y) = (x - 1)^{-c(E)}(y - 1)^{-|V|} Z_G(q, r)\,, \tag{4}$$

where $q = (x - 1)(y - 1)$ and $r_e = y - 1$ for all $e \in E$.

For a mapping $s : V \to \{1, 2, \ldots, q\}$ and an edge $e \in E$ with endvertices $e_1, e_2 \in V$ (in arbitrary order), the *partition function* of the q-state Potts model on G is defined by

$$Z_G^{\text{Potts}}(q, r) = \sum_{s:V\to\{1,2,\ldots,q\}} \prod_{e\in E} (1 + r_e[s(e_1) = s(e_2)])\,. \tag{5}$$

Theorem 4 (Fortuin and Kasteleyn [17]) *For integer values $q = 1, 2, \ldots$ it holds that*

$$Z_G(q, r) = Z_G^{\text{Potts}}(q, r)\,. \tag{6}$$

For fixed G and r it is clear from (3) that $Z_G(q, r)$, viewed as a polynomial in q, has degree at most n. Thus, it suffices to evaluate

$$Z_G^{\text{Potts}}(1, r),\ Z_G^{\text{Potts}}(2, r),\ \ldots,\ Z_G^{\text{Potts}}(n + 1, r)$$

and then recover the value $Z_G(q, r)$ via Lagrangian interpolation. For the interpolation to succeed, it is necessary to assume that the coefficient field of R has a large enough characteristic so that $1, 2, \ldots, n$ have multiplicative inverses.

At first sight the evaluation of (5) for a positive integer q appears to require $O^*(q^n)$ ring operations. Fortunately, one can do better. To this end, let us express $Z_G^{\text{Potts}}(q, r)$ in a more convenient form. For $X \subseteq V$, denote by $G[X]$ the subgraph of G induced by X, and let

$$f(X) = \prod_{e\in E(G[X])} (1 + r_e)\,. \tag{7}$$

For $q = 1, 2, \ldots$, we have

$$Z_G^{\text{Potts}}(q, r) \;= \sum_{(U_1,U_2,\ldots,U_q)} f(U_1)f(U_2)\cdots f(U_q)\,, \tag{8}$$

where the sum is over all q-tuples $(U_1, U_2, \ldots, U_q)$ with $U_1, U_2, \ldots, U_q \subseteq V$ such that $\bigcup_{i=1}^{q} U_i = V$ and $U_j \cap U_k \neq \emptyset$ for all $1 \leq j < k \leq q$.

We now prove Theorem 1 by presenting various algorithms for evaluating the Potts partition function in the form (8). Part (b) of the theorem is established in §3.1, Part (c) in §3.2, and Part (a) in §3.3.

3.1 The baseline algorithm

Let $f : 2^V \to R$ be a function that associates a ring element $f(X) \in R$ with each subset $X \subseteq V$.

The *zeta transform* $f\zeta : 2^V \to R$ is defined for all $Y \subseteq V$ by $f\zeta(Y) = \sum_{X\subseteq Y} f(X)$. The *Moebius transform* $f\mu : 2^V \to R$ is defined for all $X \subseteq V$ by $f\mu(X) = \sum_{Y\subseteq X}(-1)^{|X\setminus Y|} f(Y)$.

It is a basic fact that the zeta and Moebius transforms are inverses of each other. Furthermore, it is known [6] that

$$((f\zeta)^q\mu)(V) = \sum_{(U_1,U_2,\ldots,U_q)} f(U_1)f(U_2)\cdots f(U_q)\,, \tag{9}$$

where the sum is over all q-tuples $(U_1, U_2, \ldots, U_q)$ with $U_1, U_2, \ldots, U_q \subseteq V$ and $\bigcup_{j=1}^{q} U_j = V$. In particular, $((f\zeta)^q\mu)(V)$ can be computed directly in $O^*(3^n)$ ring operations by storing a number of ring elements polynomial in n. Using the fast zeta and Moebius transforms, $((f\zeta)^q\mu)(V)$ can be computed in $O^*(2^n)$ ring operations by storing $O^*(2^n)$ ring elements [6].

To use this to evaluate (8), adjoin a new indeterminate z into R to obtain the polynomial ring $R[z]$. Replace f with $f_z : 2^V \to R[z]$ defined for all $X \subseteq V$ by $f_z(X) = f(X)z^{|X|}$. Now evaluate the z-polynomial $((f_z\zeta)^q\mu)(V)$ and look at the coefficient of the monomial $z^{|V|}$, which by virtue of (9) is equal to (8). This proves Theorem 1(b).

3.2 A time–space tradeoff via split transforms

This section presents a "split transform" algorithm that enables a time–space tradeoff in evaluating $((f\zeta)^q\mu)(V)$ for a given function $f : 2^V \to R$ and $q = 1, 2, \ldots, n + 1$.

Split the ground set $V = \{1, 2, \ldots, n\}$ into two parts, $V_1 \subseteq V$ and $V_2 \subseteq V$, such that $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$. Let $n_1 = |V_1|$ and $n_2 = |V_2|$. For a subset $X \subseteq V$, we use subscripts to indicate the parts of the subset in V_1 and V_2; that is, we let $X_1 = X \cap V_1$ and $X_2 = X \cap V_2$. It is also convenient to split the function notation accordingly, that is, we write $f(X_1, X_2)$ for $f(X_1 \cup X_2) = f(X)$. In the context of zeta and Moebius transforms, we use X for a subset in the "spatial" (original) domain and Y for a subset in the "frequency" (transformed) domain.

An elementary observation is now that both the zeta and Moebius transforms split, that is,

$$f\zeta(Y) = \sum_{X\subseteq Y} f(X) = \sum_{X_1\subseteq Y_1}\sum_{X_2\subseteq Y_2} f(X_1, X_2) = \sum_{X_1\subseteq Y_1} f\zeta_2(X_1, Y_2) = f\zeta_2\zeta_1(Y_1, Y_2)$$

and

$$f\mu(X) = \sum_{Y\subseteq X}(-1)^{|X\setminus Y|} f(Y) = \sum_{X_1\subseteq Y_1}(-1)^{|X_1\setminus Y_1|}\sum_{X_2\subseteq Y_2}(-1)^{|X_2\setminus Y_2|} f(Y_1, Y_2) = \sum_{X_1\subseteq Y_1}(-1)^{|X_1\setminus Y_1|} f\mu_2(Y_1, X_2) = f\mu_2\mu_1(X_1, X_2)\,.$$

Also note that $f\zeta = f\zeta_2\zeta_1 = f\zeta_1\zeta_2$ and $f\mu = f\mu_2\mu_1 = f\mu_1\mu_2$.

To arrive at the split transform algorithm for computing $((f\zeta)^q\mu)(V)$, split the outer Moebius transform and the inner zeta transform to get

$$((f\zeta)^q\mu)(V) = \sum_{Y_1 \subseteq V_1} (-1)^{|V_1 \setminus Y_1|} \sum_{Y_2 \subseteq V_2} (-1)^{|V_2 \setminus Y_2|} (f\zeta_1\zeta_2(Y_1, Y_2))^q \ .$$

Now let Y_1 be fixed and consider the inner sum. To evaluate the inner sum for a fixed Y_1, it suffices to have $f\zeta_1\zeta_2(Y_1, Y_2)$ available for each $Y_2 \subseteq V_2$. By definition,

$$f\zeta_1\zeta_2(Y_1, Y_2) = \sum_{X_2 \subseteq Y_2} f\zeta_1(Y_1, X_2) \, .$$

Observe that if we have $f\zeta_1(Y_1, X_2)$ stored for each $X_2 \subseteq V_2$, then we can evaluate $f\zeta_1\zeta_2(Y_1, Y_2)$ for each $Y_2 \subseteq V_2$ simultaneously using the fast zeta transform. This takes in total at most $2^{n_2} n_2$ ring operations and requires one to store at most $2^{n_2} n_2$ ring elements.

For fixed Y_1 and X_2, we can evaluate and store

$$f\zeta_1(Y_1, X_2) = \sum_{X_1 \subseteq Y_1} f(X_1, X_2)$$

by plain summation in at most $2^{|Y_1|}$ ring operations. Thus, for fixed Y_1, we can evaluate $f\zeta_1(Y_1, X_2)$ for each $X_2 \subseteq V_2$ in total at most $2^{|Y_1|} 2^{n_2}$ ring operations.

Considering each $Y_1 \subseteq V_1$ in turn, we can thus evaluate $((f\zeta)^q\mu)(V)$ by storing at most $2^{n_2} n_2$ ring elements and executing a number of ring operations at most within a polynomial factor of

$$\sum_{Y_1 \subseteq V_1} (2^{n_2} n_2 + 2^{|Y_1|} 2^{n_2}) = (3^{n_1} + 2^{n_1} n_2) 2^{n_2} \, .$$

This completes the description and analysis of the split transform algorithm, proving Theorem 1(c).

3.3 An algorithm over connected sets

We now show how to compute $((f\zeta)^q\mu)(V)$ for each $q = 1, 2, \ldots, n+1$ in time $O^*(\sigma(G))$. We assume that f satisfies the following property: for all $X \subseteq V$ it holds that

$$f(X) = f(X_1) f(X_2) \cdots f(X_s) \tag{10}$$

where $G[X_1], G[X_2], \ldots, G[X_s]$ are the connected components of $G[X]$. For convenience we also assume that $f(\emptyset) = 1$. Note that the factorisation (10) is well-defined because of commutativity of R. Also note that (7) satisfies (10).

Our idea will be to compute $((f\zeta)^q\mu)(X)$ recursively, starting at $X = V$, and guided by two rules. If the graph $G[X]$ is not connected, the recursion continues with each of the vertex sets $X_1, X_2, \ldots, X_s$ of the connected components $G[X_1], G[X_2], \ldots, G[X_s]$ of $G[X]$. Otherwise; that is, if $G[X]$ is connected, the recursion considers the maximal proper subsets $X \setminus \{i\}$, $i \in X$.

For this idea to work, the recursion actually computes certain intermediate values $F(X, q, i)$ rather than the values $((f\zeta)^q\mu)(X)$ themselves. To define these intermediate values, we first need to partition the subsets of X based on the maximum common suffix. Let $Y \equiv_i X$ be a shorthand for $Y \cap \{i+1, i+2, \ldots, n\} = X \cap \{i+1, i+2, \ldots, n\}$.

Lemma 5 (Suffix partition) *Let $Y \subseteq X \subseteq \{1, 2, \ldots, n\}$. Then, either $Y = X$ or there exists a unique $i \in X$ such that $Y \equiv_{i-1} X \setminus \{i\}$.*

Proof. Either $Y = X$ or $i = \max X \setminus Y$. ∎

Now define, for $X \subseteq V$, $q = 1, 2, \ldots, n+1$, and $i = 0, 1, \ldots, n$, the values

$$F(X, q, i) = \sum_{(U_1, U_2, \ldots, U_q)} \prod_{j=1}^{q} f(U_j) \, , \tag{11}$$

where the sum is over all q-tuples $(U_1, U_2, \ldots, U_q)$ such that both $U_1, U_2, \ldots, U_q \subseteq X$ and $\bigcup_{j=1}^{q} U_j \equiv_i X$. Note that $((f\zeta)^q\mu)(V) = F(V, q, 0)$.

We now turn to the details of our two rules for computing the intermediate values in (11). We consider first the case when $G[X]$ is not connected.

Lemma 6 *Let $G[X_1], G[X_2], \ldots, G[X_s]$ be the connected components of $G[X]$ and let $U \subseteq X$. Then,*

$$f(U) = f(U \cap X_1) f(U \cap X_2) \cdots f(U \cap X_s) \, .$$

Proof. Let $G[U_1], G[U_2], \ldots, G[U_t]$ be the connected components of $G[U]$. Then, by (10),

$$f(U) = f(U_1) f(U_2) \cdots f(U_t) \, .$$

Because $U \subseteq X$ holds, for every U_i there is a unique $h(i) \in \{1, 2, \ldots, s\}$ such that $U_i \subseteq X_{h(i)}$. Moreover, since $\{U_1, U_2, \ldots, U_t\}$ is a partition of U, we have that $\{U_i : i \in h^{-1}(j)\}$ is a partition of $U \cap X_j$ for all $j = 1, 2, \ldots, s$. Thus, by (10) we have $f(U \cap X_j) = \prod_{i \in h^{-1}(j)} f(U_i)$ for all $j = 1, 2, \ldots, s$. In particular, by commutativity of R,

$$f(U) = \prod_{i=1}^{t} f(U_i) = \prod_{j=1}^{s} \prod_{i \in h^{-1}(j)} f(U_i) = \prod_{j=1}^{s} f(U \cap X_j) \, . \quad ∎$$

This enables a factorisation of the intermediate values over connected components:

Lemma 7 *Let $G[X_1], G[X_2], \ldots, G[X_s]$ be the connected components of $G[X]$. Then,*

$$F(X, q, i) = \prod_{k=1}^{s} F(X_k, q, i) \, . \tag{12}$$

Proof. Consider an arbitrary q-tuple $(U_1, U_2, \ldots, U_q)$ with $U_1, U_2, \ldots, U_q \subseteq X$ and $\bigcup_{j=1}^{q} U_j \equiv_i X$. Because $\{X_1, X_2, \ldots, X_s\}$ is a partition of X, we have $\bigcup_{j=1}^{q} U_j \equiv_i X$ if and only if $X_k \cap \bigcup_{j=1}^{q} U_j \equiv_i X_k \cap X$ holds for all $k = 1, 2, \ldots, s$. Put otherwise, we have $\bigcup_{j=1}^{q} U_j \equiv_i X$ if and only if $\bigcup_{j=1}^{q} (X_k \cap U_j) \equiv_i X_k$ holds for all $k = 1, 2, \ldots, s$. Using Lemma 6 for each U_j in turn, we have, by commutativity of R, the unique factorisation over pairwise intersections

$$f(U_1)f(U_2)\ldots f(U_q) = \prod_{j=1}^{q}\prod_{k=1}^{s} f(U_j \cap X_k) = \prod_{k=1}^{s}\prod_{j=1}^{q} f(U_j \cap X_k)\,.$$

We are done because $(U_1, U_2, \ldots, U_q)$ was arbitrary. ∎

In the case when $G[X]$ is connected, we apply the following algorithm. The comments delimited by "[[" and "]]" justify the computations in the algorithm.

Algorithm U. (*Up-step.*)
Input: A subset $X \subseteq V$ and the values $F(X \setminus \{i\}, q, i-1)$ for each $q = 1, 2, \ldots, n+1$ and $i \in X$.
Output: The value $F(X, q, i)$ for each $q = 1, 2, \ldots, n+1$ and $i = 0, 1, \ldots, n$.

U1: For each $q = 1, 2, \ldots, n+1$, set

$$F(X, q, n) = \Bigl(f(X) + \sum_{i \in X} F(X \setminus \{i\}, 1, i-1)\Bigr)^q\,.$$

[[By the suffix partition lemma, $\sum_{Y \subseteq X} f(Y) = \sum_{i \in X} F(X \setminus \{i\}, 1, i-1)$. Adding $f(X)$ and taking powers, we obtain $F(X, q, n)$.]]

U2: For each $q = 1, 2, \ldots, n+1$ and $i = n, n-1, \ldots, 1$, set

$$F(X, q, i-1) = F(X, q, i) - [i \in X]F(X \setminus \{i\}, q, i-1)\,.$$

[[There are two cases to consider to justify correctness. First, assume that $i \notin X$. Consider an arbitrary q-tuple $(U_1, U_2, \ldots, U_q)$ with $U_1, U_2, \ldots, U_q \subseteq X$. Let $Y = \bigcup_{j=1}^{q} U_j$. Clearly, $Y \subseteq X$. Because $i \notin X$ and $Y \subseteq X$, we have $Y \equiv_{i-1} X$ if and only if $Y \equiv_i X$. Thus, $F(X, q, i-1) = F(X, q, i)$. Second, assume that $i \in X$. In this case we have $Y \equiv_i X$ if and only if either $Y \equiv_{i-1} X$ or $Y \equiv_{i-1} X \setminus \{i\}$ (the former case occurs if $i \in Y$, the latter if $i \notin Y$). In the latter case, $Y \subseteq X \setminus \{i\}$ and hence $U_1, U_2, \ldots, U_q \subseteq X \setminus \{i\}$. Thus, $F(X, q, i-1) = F(X, q, i) - F(X \setminus \{i\}, q, i-1)$.]]

It remains to describe the main recursion. At each recursive invocation with input $X \subseteq V$, we return the value $F(X, q, i)$ for each $q = 1, 2, \ldots, n+1$ and $i = 0, 1, \ldots, n$. To this end, we first find the connected components $G[X_1], G[X_2], \ldots, G[X_s]$ of $G[X]$. If $s > 1$, we recursively compute the values $F(X_k, q, i)$ for each $k = 1, 2, \ldots, s$ and return their product according to Lemma 7. Otherwise, we recursively compute the values $F(X \setminus \{i\}, q, i-1)$ for each $i \in X$ and use Algorithm U to compute the values $F(X, q, i)$.

We observe that the recursion starts at $X = V$ and thereafter considers only sets that induce a connected subgraph, or maximal proper subsets of such sets. Thus, the number of different recursive invocations is $O^*(\sigma(G))$. Moreover, each recursive invocation takes time bounded by a polynomial in n. We use memoisation to avoid redundant recomputation of previously computed values, so the running time and space usage are as claimed by Theorem 1(a).

4 An alternative recursion

We derive an alternative recursion for $Z_G(q, r)$ based on induced subgraphs and fast subset convolution. Let R be a commutative ring. Associate a ring element $r_e \in R$ with each $e \in E$. For $k = 1, 2, \ldots, n$, let

$$S_G(k, r) = \sum_{\substack{F \subseteq E \\ c(F) = k}} \prod_{e \in F} r_e$$

and observe that $Z_G(q, r) = \sum_{k=1}^{n} q^k S_G(k, r)$. Thus, to determine $Z_G(q, r)$, it suffices to compute $S_G(k, r)$ for all $k = 1, 2, \ldots, n$.

To this end, the values $S_G(k, r)$ can be computed using the following recursion over induced subgraphs of G. Let $W \subseteq V$ and consider the subgraph $G[W]$ induced by W in G. Suppose that $S_{G[U]}(k, r)$ has been computed for all $\emptyset \neq U \subseteq W$ and $k = 1, 2, \ldots, |U|$.

To compute $S_{G[W]}(k, r)$ for $k = 2, 3, \ldots, |W|$, observe that a disconnected subgraph of $G[W]$ partitions into connected components. Thus, for $k \geq 2$ we have

$$S_{G[W]}(k, r) = \frac{1}{k} \sum_{\emptyset \neq U \subseteq W} S_{G[U]}(1, r) S_{G[W \setminus U]}(k-1, r)\,. \quad (13)$$

For the connected case, that is, for $k = 1$, it suffices to observe that we can subtract the disconnected subgraphs from the set of all subgraphs to obtain the connected graphs; put otherwise,

$$S_{G[W]}(1, r) = \prod_{e \in E(G[W])} (1 + r_e) - \sum_{k \geq 2} S_{G[W]}(k, r)\,. \quad (14)$$

The recursion defined by (13) and (14) can now be evaluated for $|W| = 1, 2, \ldots, n$ in total $O^*(2^n)$ ring operations using fast subset convolution [6]. As a technical observation we remark that (13) assumes that k has a multiplicative inverse in R; this assumption can be removed, but we omit the details from this extended abstract. We also note that an algorithm running in $O^*(\sigma(G))$ ring operations can be developed in this context, matching Theorem 1(a). However, it is not immediate whether a polynomial-space algorithm for the Tutte polynomial can be developed based on (13) and (14).

5 The cover polynomial

The proof of Theorem 3 involves several inclusion–exclusion-based arguments with different purposes and in a nested fashion, so we first give a high-level overview of the concepts involved. One readily observes that the cover polynomial can be expressed as a sum over partitionings of the vertex set, each vertex subset appropriately weighted, so the inclusion–exclusion technique [8] applies. Computing the weights for all possible vertex subsets is again a hard problem, but the fast Moebius inversion algorithm [7] can be used to compute the necessary values beforehand. This leads to an exponential-space algorithm. Finally, to use inclusion–exclusion to reduce the space to polynomial [28, 30], we apply the mentioned transforms in a nested manner and switch the order of certain involved summations.

We turn to the details of the proof. For $X \subseteq V$, denote by $p(X)$ the number of spanning directed paths in $D[X]$, and denote by $c(X)$ the number of spanning directed cycles in $D[X]$. Define $p(\emptyset) = c(\emptyset) = 0$. Note that for all $x \in V$ we have $p(\{x\}) = 1$ and that $c(\{x\})$ is the number of loops incident with x.

By definition,

$$c_D(i,j) = \frac{1}{i!\,j!} \sum_{\substack{X_1,X_2,\ldots,X_i,\\ Y_1,Y_2,\ldots,Y_j}} p(X_1)\cdots p(X_i)c(Y_1)\cdots c(Y_j)\,,$$

where the sum is over all $(i+j)$-tuples $(X_1, X_2, \ldots, X_i, Y_1, Y_2, \ldots, Y_j)$ such that $\{X_1, X_2, \ldots, X_i, Y_1, Y_2, \ldots, Y_j\}$ is a partition of V.

We derive an alternative expression for $c_D(i,j)$ using the principle of inclusion and exclusion. Let z be a polynomial indeterminate. Define for every $U \subseteq V$ the polynomials

$$P_U(z) = \sum_{X\subseteq U} p(X) z^{|X|} \quad \text{and} \quad C_U(z) = \sum_{Y\subseteq U} c(Y) z^{|Y|}\,. \tag{15}$$

Viewed as set functions, $P_U(z)$ and $C_U(z)$ are zeta transforms of the set functions $p(X)z^{|X|}$ and $c(Y)z^{|Y|}$, respectively. By inclusion–exclusion,

$$c_D(i,j) = \frac{1}{i!\,j!} \sum_{U\subseteq V} (-1)^{|V\setminus U|} \{z^n\}(P_U(z)^i C_U(z)^j)\,. \tag{16}$$

It remains to show how to compute $p(X)$ and $c(Y)$. For $S \subseteq V$ let $w_S(s,t,\ell)$ denote the number of directed walks of length ℓ from vertex s to vertex t in $D[S]$. Define $w_S(s,t,\ell) = 0$ if $s \notin S$ or $t \notin S$. By inclusion–exclusion, again,

$$\begin{aligned} p(X) &= \sum_{1\le s,t\le n} \sum_{S\subseteq X} (-1)^{|X\setminus S|} w_S(s,t,|X|-1)\,,\\ c(Y) &= \frac{1}{|Y|} \sum_{s=1}^{n} \sum_{S\subseteq Y} (-1)^{|Y\setminus S|} w_S(s,s,|Y|)\,. \end{aligned} \tag{17}$$

Observing that $w_S(s,t,\ell)$ can be computed in polynomial time, a direct evaluation of (16), (15), and (17) computes $c_D(i,j)$ in time $O^*(4^n)$ and polynomial space.

To improve to $O^*(3^n)$ and polynomial space, observe that we can substitute (17) to (15) and collect terms to obtain

$$P_U(z) = \sum_{S\subseteq U} P_{U,S}(z) \quad \text{and} \quad C_U(z) = \sum_{S\subseteq U} C_{U,S}(z)$$

where

$$P_{U,S}(z) = \sum_{1\le s,t\le n} \sum_{k=0}^{|U\setminus S|} \tbinom{|U\setminus S|}{k} (-1)^k z^{|S|+k} w_S(s,t,|S|+k-1)\,,$$

$$C_{U,S}(z) = \sum_{s=1}^{n} \sum_{k=0}^{|U\setminus S|} \frac{1}{|S|+k} \tbinom{|U\setminus S|}{k} (-1)^k z^{|S|+k} w_S(s,s,|S|+k)\,.$$

This establishes part (b) of the theorem.

For part (a), we show how to evaluate $c_D(i,j)$ in time and space $O^*(2^n)$. Namely, p and c can be computed in time and space $O^*(2^n)$ via fast Moebius inversion. Given p and c, the polynomials P and C can be computed in time and space $O^*(2^n)$ via fast zeta transform. And finally, given P and C, (16) can be evaluated in time $O^*(2^n)$.

References

[1] A. Andrzejak, *An algorithm for the Tutte polynomials of graphs of bounded treewidth*, Discrete Math. **190** (1998), 39–54.

[2] J. D. Annan, *The complexities of the coefficients of the Tutte polynomial*, Discrete Appl. Math. **57** (1995), 93–103.

[3] M. H. G. Anthony, *Computing chromatic polynomials*, Ars Combinatoria **29** (1990), 216–220.

[4] N. Biggs, *Algebraic Graph Theory*, 2nd ed., Cambridge University Press, 1993.

[5] A. Björklund, T. Husfeldt, *Exact algorithms for exact satisfiability and number of perfect matchings*, Algorithmica, 2007, doi:10.1007/s00453-007-9149-8.

[6] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, *Fourier meets Möbius: fast subset convolution*, Proceedings of the 39th Annual ACM Symposium on Theory of Computing (San Diego, CA, June 11–13, 2007), Association for Computing Machinery, 2007, pp. 67–74.

[7] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, *The travelling salesman problem in bounded degree graphs*, Proceedings of the 35th International Colloquium on Automata, Languages and Programming (Reykjavik, Iceland, July 6–13, 2008), Lecture Notes in Computer Science 5125, Springer, 2008, pp. 198–209.

[8] A. Björklund, T. Husfeldt, M. Koivisto, *Set partitioning via inclusion–exclusion*, SIAM J. Computing, to appear.

[9] M. Bläser, H. Dell, *Complexity of the cover polynomial*, Proceedings of the 34th International Colloquium on Automata, Languages and Programming (Wroclaw, Poland, July 9–13, 2007), Lecture Notes in Computer Science 4596, Springer, 2007, pp. 801–812.

[10] B. Bollobás, *Modern Graph Theory*, Graduate Texts in Mathematics 184, Springer, 1998.

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

[12] A. Cayley, *A theorem on trees*, Quart. J. Math. **23** (1889), 376–378.

[13] F. R. K. Chung, R. L. Graham, *On the cover polynomial of a digraph*, J. Combin. Theory Ser. B **65** (1995), 273–290.

[14] F. Chung, S.-T. Yau, *Coverings, heat kernels, and spanning trees*, Electron. J. Combinatorics **6** (1999) #R12, 21 pp.

[15] M. Dyer, L.A. Goldberg, M. Jerrum, *The complexity of weighted Boolean #CSP*, arXiv:0704.3683v1 [cs.CC] (Apr, 2007).

[16] F. V. Fomin, S. Gaspers, S. Saurabh, *Improved exact algorithms for counting 3- and 4-colorings*, Computing and Combinatorics, 13th Annual International Conference (COCOON), Banff, Canada, July 16–19, 2007, Lecture Notes in Computer Science 4598, Springer, 2007, pp. 65–74.

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

[18] H. Gebauer, Y. Okamoto, *Fast exponential-time algorithms for the forest counting and the Tutte polynomial computation in graph classes*, Int. J. Found. Comput. S. (2008), to appear.

[19] O. Giménez, P. Hliněný, M. Noy, *Computing the Tutte polynomial on graphs of bounded clique-width*, SIAM J. Discrete Math. **20** (2006), 932–946.

[20] C. Godsil, G. Royle, *Algebraic Graph Theory*, Graduate Texts in Mathematics 207, Springer, 2001.

[21] L. A. Goldberg, M. Jerrum, *Inapproximability of the Tutte polynomial*, Inform. Comput. **206** (2008), 908–929.

[22] G. Haggard, D. Pearce, G. Royle, *Computing Tutte polynomials*, Technical Report, Victoria University of Wellington, NZ, in preparation.

[23] P. Hliněný, *The Tutte polynomial for matroids of bounded branch-width*, Combin. Probab. Comput. **15** (2006), 397–409.

[24] H. Imai, *Computing the invariant polynomials of graphs, networks, and matroids*, IEICE T. Inf. Syst. **E93–D** (2000), 330–343.

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

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

[27] S. Kapoor, H. Ramesh, *Algorithms for enumerating all spanning trees of undirected and weighted graphs*, SIAM J. Comput. **24** (1995), 247–265.

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

[29] P. W. Kasteleyn, *The statistics of dimers on a lattice: I. The number of dimer arrangements on a quadratic lattice*, Physica **27** (1961), 1209–1225.

[30] S. Kohn, A. Gottlieb, M. Kohn, *A generating function approach to the traveling salesman problem*, Proceedings of the 1977 Annual Conference (ACM'77), Association for Computing Machinery, 1977, pp. 294–300.

[31] M. Koivisto, *Optimal 2-constraint satisfaction via sum-product algorithms*, Inform. Process. Lett. **98** (2006), 22–24.

[32] W. Kook, V. Reiner, D. Stanton, *A convolution formula for the Tutte polynomial*, J. Combin. Theory Ser. B **76** (1999), 297–300.

[33] E. L. Lawler, *A note on the complexity of the chromatic number problem*, Inf. Process. Lett. **5** (1976), 66–67.

[34] S. D. Noble, *Evaluating the Tutte polynomial for graphs of bounded tree-width*, Combin. Probab. Comput. **7** (1998), 307–321.

[35] J. G. Oxley, D. J. A. Welsh, *The Tutte polynomial and percolation*, Graph Theory and Related Topics (J. A. Bondy and U. S. R. Murty, Eds.), Academic Press, 1979, pp. 329–339.

[36] R. B. Potts, *Some generalized order-disorder transformations*, Proceedings of the Cambridge Philosophical Society **48** (1952), 106–109.

[37] K. Sekine, H. Imai, S. Tani, *Computing the Tutte polynomial of a graph of moderate size*, Algorithms and Computation, 6th International Symposium (ISAAC '95), Cairns, Australia, December 4–6, 1995, Lecture Notes in Computer Science 1004, Springer, 1995, pp. 224–233.

[38] A. D. Sokal, *The multivariate Tutte polynomial (alias Potts model) for graphs and matroids*, Surveys in Combinatorics, 2005, London Mathematical Society Lecture Note Series 327, Cambridge University Press, 2005, pp. 173–226.

[39] P. Traxler, *The time complexity of constraint satisfaction*, Proceedings of the 3rd International Workshop on Exact and Parameterized Computation (Victoria, BC, Canada, May 14–16, 2008), to appear.

[40] W. T. Tutte, *Graph-polynomials*, Adv. Appl. Math. **32** (2004), 5–9.

[41] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput. **8** (1979), 410–421.

[42] D. J. A. Welsh, *Complexity: Knots, Colourings and Counting*, London Mathematical Society Lecture Note Series 186, Cambridge University Press, 1993.

[43] D. J. A. Welsh, *The Tutte polynomial*, Random Structures Algorithms **15** (1999), 210–228.

[44] D. J. A. Welsh, C. Merino, *The Potts model and the Tutte polynomial*, J. Math. Phys. **41** (2000), 1127–1152.

[45] D. B. West, *Introduction to Graph Theory*, 2nd ed., Prentice–Hall, 2001.

[46] H. S. Wilf, *Algorithms and Complexity*, Prentice–Hall, 1986.

[47] R. Williams, *A new algorithm for optimal constraint satisfaction and its implications*, Theoret. Comput. Sci. **348** (2005), 357–365.