COMPUTING THE TUTTE POLYNOMIAL IN VERTEX-EXPONENTIAL TIME

ANDREAS BJÖRKLUND*, THORE HUSFELDT*, PETTERI KASKI†, AND MIKKO KOIVISTO†

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.

An implementation of the algorithm outperforms deletion–contraction also in practice.

1. Introduction

Tutte's motivation for studying what he called the "dichromatic polynomial" was algorithmic. By his own entertaining account [41], 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

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

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 [36] 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 [39, 43, 44, 45].

*Lund University, Department of Computer Science, P.O.Box 118, SE-22100 Lund, Sweden. E-mail: andreas.bjorklund@logipard.com, thore.husfeldt@cs.lu.se.

†Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, P.O.Box 68, FI-00014 University of Helsinki, Finland. E-mail: petteri.kaski@cs.helsinki.fi, mikko.koivisto@cs.helsinki.fi. This research was supported in part by the Academy of Finland, Grants 117499 (P.K.) and 109101 (M.K.).

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 $\exp\bigl(O(n)\bigr)$, that is, in "vertex-exponential" time, while $\tau(G)$ typically is $\exp\bigl(\omega(n)\bigr)$ and can be as large as n^{n-2} [12]. Previously, vertex-exponential running time bounds were known only for evaluations of T_G in special regions of the Tutte plane (x, y), such as for the chromatic polynomial and (using exponential space) the reliability polynomial, or only for special classes of graphs such as planar graphs or bounded-degree graphs. We provide a more detailed overview of such prior work in §2.

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 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 $\sigma(G)n^{O(1)}$;*
(b) *in time $3^n n^{O(1)}$ and polynomial space; and*
(c) *in time $3^{n-s}2^s n^{O(1)}$ and space $2^s n^{O(1)}$ 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. Moreover, even for those curves and points of the Tutte plane where a vertex-exponential time algorithm was known before, our algorithm improves or at least matches their performance, with only a few exceptions (see Figure 1).

For bounded-degree graphs G, the deletion–contraction algorithm itself runs in vertex-exponential time because $\tau(G) = \exp\bigl(O(n)\bigr)$. 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 $\xi_\Delta^n n^{O(1)}$, 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]. It turns out that a directed variant of our main theorem can be established using recent techniques that are by now well understood, we include the precise statement and proof in Appendix C.

1.2. **Overview of techniques.** The Tutte polynomial is, in essence, a sum over connected spanning subgraphs. Managing this connectedness property introduces a computational

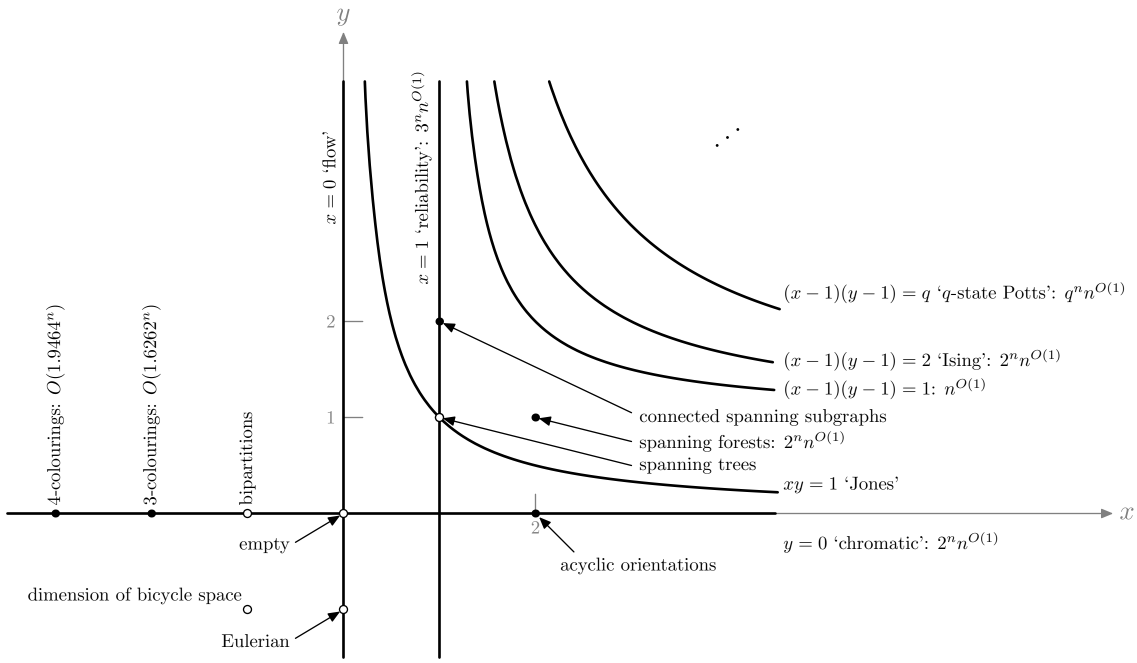


FIGURE 1. An atlas of the Tutte plane (x, y). The five points shown by circles and the points on the hyperbola $(x-1)(y-1) = 1$ are in P, all other points are #P-complete. Those points and lines where algorithms with complexity $\exp\big(O(n)\big)$ were previously known (sometimes only in exponential space), are labelled with their running time; note that the hyperbolas $(x-1)(y-1) = q$ were known to be vertex-exponential only for fixed integer q. See §2.3 for references. Our result is that the entire plane admits algorithms with running time $2^n n^{O(1)}$ and exponential space, or time $3^n n^{O(1)}$ and polynomial space. The only points that are known to admit algorithms with better bounds are the “colouring” points $(-2, 0)$ and $(-3, 0)$, the “Ising” hyperbola $(x-1)(y-1) = 2$, for which a faster algorithm in observed in §2.3, and of course the points in P. (Only the positive branches of the hyperbolas are drawn.)

challenge not present with its specialisations, e.g., with the chromatic polynomial. Neither the dynamic programming algorithm across vertex subsets by Lawler [34] 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 Appendix C 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-states Potts model at suitable points using inclusion–exclusion, which then, by a neat identity due to Fortuin and Kasteleyn [16, 39], 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 [39] 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 §4.1. The smooth time–space tradeoff, Theorem 1(c), is obtained by a new "split transform" technique (Appendix B).

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 (§4.2). 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] Appendix D briefly reports on our experiences with implementing and running this algorithm; it outperforms deletion–contraction in the worst case when $n \geq 13$.

1.3. **Conventions.** For standard graph-theoretic terminology we refer to West [46]. 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, $V = V(G)$ 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 assume $m = n^{O(1)}$ and remark that this assumption is implicit already in Theorem 1. (Without this assumption, all the time bounds require an additional multiplicative term $m^{O(1)}$.) 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 work: Algorithms for the Tutte Polynomial

The direct evaluation of $T_G(x, y)$ based on (1) takes $2^m n^{O(1)}$ 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 $\tau(G)$.

[1] A previous version of this manuscript followed this route, establishing Theorem 1(a).

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 $2^n n^{O(1)}$ [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 formula

$$T_G(x,y) = \begin{cases} 1 & \text{if } G \text{ has no edges;} \\ yT_{G\backslash e}(x,y) & \text{if } e \text{ is a loop;} \\ xT_{G/e}(x,y) & \text{if } e \text{ is a bridge;} \\ T_{G\backslash e}(x,y) + T_{G/e}(x,y) & \text{otherwise.} \end{cases} \tag{3}$$

The deletion–contraction algorithm defined by a direct evaluation of (3) leads to a running time that scales as the Fibonacci sequence, $\big((1+\sqrt{5})/2\big)^{n+m} = O(1.6180^{n+m})$ [47]. Sekine, Imai, and Tani [38] observed that the corresponding computation tree has one leaf for every spanning tree of G, so (3) 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, 38].

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 celebrated Recipe Theorem of Oxley and Welsh [36] (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 [33], 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, algorithms running in time $\exp\big(O(n)\big)$ have been known before. 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 $2^n n^{O(1)}$ 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 $2^{n\omega/3}n^{O(1)}=O(1.732^n)$ and exponential space, where ω is the exponent of matrix multiplication; this is yet a new application of Williams's trick [5, 32, 48].

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* [37]. This is a sum over the configurations of n spins each having q possible states. It can be computed trivially in time $q^n n^{O(1)}$ and, via fast matrix multiplication, in time $q^{n3/\omega}n^{O(1)}$. We will show in §3 that, in fact, time $2^n n^{O(1)}$ 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 $3^n n^{O(1)}$ 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 $2^n n^{O(1)}$ 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 $P_G(t)=(-1)^{n-c}t^c T_G(1-t,0)$ and can be computed in time $2^n n^{O(1)}$ [8]. Vertex-exponential time algorithms were known at least since Lawler [34], 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 $O(1.9464^n)$ [15]. The case $x=0$ is trivial.

To the best knowledge of the authors, no algorithms with running time $\exp\big(O(n)\big)$ 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 $\exp\bigl(O(n)\bigr)$ running time or better:

Planar graphs: If G is planar, then the Tutte polynomial can be computed in time $\exp\bigl(O(\sqrt{n})\bigr)$ [38]. 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 $\exp\bigl(O(n^\alpha \log n)\bigr)$.

Bounded tree-width and branch-width: For k a fixed integer, if G has tree-width k then T_G can be computed in polynomial time [1, 35]. 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 $\exp\bigl(O(n^{1-1/(k+2)})\bigr)$ [18]. A special case of this is the class of cographs (graphs without an induced path of 4 vertices), where the bound becomes $\exp\bigl(O(n^{2/3})\bigr)$.

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 $O\bigl(1.6180^{(1+\Delta/2)n}\bigr)$ directly from the recurrence. Gebauer and Okamoto improve this to $\chi_\Delta^n n^{O(1)}$, 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 algorithms) is bounded by $\tau(G) = O\bigl(\nu_k^n n^{-1} \log n\bigr)$, 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 $\exp\bigl(O(n)\bigr)$ in general, but still faster than by deletion–contraction [17].

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. **Computional complexity.** The study of the computational complexity of the Tutte polynomial begins with Valiant's theory of #P-completeness [42] 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 [43] 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 $\exp\big(O(n/\log n)\big)$.

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 [19]. Very recently, Traxler [40] has shown that already the decision version of $(q,2)$-CSP requires time $\exp\big(\Omega(n\log q)\big)$ 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 and let G be an undirected graph with vertex set $V=\{1,2,\ldots,n\}$ and edge set E, $m=n^{O(1)}$. We allow G to have parallel edges and loops. Associate with each $e\in E$ a ring element $r_e\in R$. The *multivariate Tutte polynomial* [39] of G is the polynomial

$$Z_G(q,r)=\sum_{F\subseteq E} q^{c(F)}\prod_{e\in F} r_e\,, \tag{4}$$

where q is an indeterminate and $c(F)$ denotes the number of connected 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 as a bivariate evaluation of the multivariate polynomial $Z_G(q,r)$ via

$$T_G(x,y)=(x-1)^{-c(E)}(y-1)^{-|V|}Z_G\big((x-1)(y-1),y-1\big)\,. \tag{5}$$

3.1. **The Fortuin–Kasteleyn identity.** At points $q=1,2,\ldots$ the multivariate Tutte polynomial $Z_G(q,r)$ can be represented as an evaluation of the partition function of the q-state Potts model [16, 39].

For a mapping $s:V\to\{1,2,\ldots,q\}$ and an edge $e\in E$ with endvertices x and y, define $\delta_e^s=1$ if $s(x)=s(y)$ and $\delta_e^s=0$ if $s(x)\neq s(y)$. The *partition function* of the q-state Potts model on G is defined by

$$Z_G^{\text{Potts}}(q,r)=\sum_{s:V\to\{1,2,\ldots,q\}}\prod_{e\in E}\big(1+r_e\delta_e^s\big)\,. \tag{6}$$

Theorem 3 (Fortuin and Kasteleyn)**.** *For all $q=1,2,\ldots$ it holds that*

$$Z_G(q,r)=Z_G^{\textit{Potts}}(q,r)\,. \tag{7}$$

3.2. **The multivariate Tutte polynomial via the q-state Potts model.** By virtue of the Fortuin–Kasteleyn identity (7), to compute $Z_G(q,r)$ 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 $Z_G(q,r)$ 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 (6) for a positive integer q appears to require $q^n n^{O(1)}$ ring operations. Fortunately, one can do better. To this end, let us express $Z_G^{\mathrm{Potts}}(q,r)$ in a more convenient form. For $X \subseteq V$, denote by $G[X]$ the subgraph of G induced by X, and let

$$f(X) = \prod_{e \in E(G[X])} (1 + r_e)\,. \tag{8}$$

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

$$Z_G^{\mathrm{Potts}}(q,r) \;=\; \sum_{(U_1,U_2,\ldots,U_q)} f(U_1)f(U_2)\cdots f(U_q)\,, \tag{9}$$

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

We now proceed to develop algorithms for evaluating the Potts partition function in the form (9).

3.3. **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, that is, $f\zeta\mu = f\mu\zeta = f$ for all f. Furthermore, it is known [6] that

$$\big((f\zeta)^q\mu\big)(V) = \sum_{(U_1,U_2,\ldots,U_q)} f(U_1)f(U_2)\cdots f(U_q)\,, \tag{10}$$

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 $\cup_{j=1}^{q} U_j = V$. In particular, $\big((f\zeta)^q\mu\big)(V)$ can be computed directly in $3^n n^{O(1)}$ ring operations by storing $n^{O(1)}$ ring elements. Using the fast zeta and Moebius transforms, $\big((f\zeta)^q\mu\big)(V)$ can be computed in $2^n n^{O(1)}$ ring operations by storing $2^n n^{O(1)}$ ring elements [6].

To use this to evaluate (9), 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 $\big((f_z\zeta)^q\mu\big)(V)$ and look at the coefficient of the monomial $z^{|V|}$, which by virtue of (10) is equal to (9).

This baseline algorithm together with (5), (7), and Lagrangian interpolation establishes that the Tutte polynomial $T_G(x,y)$ can be computed (a) in time and space $2^n n^{O(1)}$; and (b) in time $3^n n^{O(1)}$ and space $n^{O(1)}$. This proves Theorem 1(b). A more careful analysis of $\big((f\zeta)^q\mu\big)(V)$ enables the time–space tradeoff in Theorem 1(c). [[See Appendix B.]]

4. Improvements and variations

4.1. **An algorithm over connected sets.** It is useful to think of $X \subseteq V$ in what follows as the current subset under consideration. We start with a lemma that partitions the subsets of X based on the maximum common suffix. To this end, 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 4 (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 $i = \max X \setminus Y$. $\square$

The intermediate values computed by the algorithm are now defined as follows.

Definition 5. Let $X \subseteq V$, $q = 1, 2, \ldots, n+1$, and $i = 0, 1, \ldots, n$. Let

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

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 $\cup_{j=1}^{q} U_j \equiv_i X$.

Note that $F(V, q, 0) = ((f\zeta)^q \mu)(V)$. Thus, it suffices to compute $F(V, q, 0)$.

We are now ready to describe the algorithm that computes the intermediate values $F(X, q, i)$ in Definition 5. The algorithm considers one set $X \subseteq V$ at a time, starting with the empty set $X = \emptyset$ and proceeding upwards in the subset lattice. It is required that the maximal proper subsets of X have been considered before X itself is considered; for example, we can consider the subsets of V in increasing lexicographic order. The comments delimited by "[[" and "]]" justify the computations in the algorithm.

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

U1: For each $q = 1, 2, 3, \ldots, n+1$, set

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

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

U2: For each $q = 1, 2, 3, \ldots, n+1$ 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 $Y = \cup_{j=1}^{q} U_j$. 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, $F(X, q, i-1) = F(X, q, i) - F(X \setminus \{i\}, q, i-1)$.]]

Assume that f satisfies the following property: for all $X \subseteq V$ it holds that

$$f(X) = f(X_1) f(X_2) \cdots f(X_s) \tag{11}$$

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 (11) is well-defined because of commutativity of R. Also note that (8) satisfies (11).

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

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

The recursion (12) now enables the following top-down evaluation strategy for the intermediate values in Definition 5. Consider a nonempty $X \subseteq V$. If $G[X]$ is not connected, recursively solve the intermediate values of 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 $G[X]$, and assemble the solution using (12). Otherwise; that is, if $G[X]$ is connected, recursively solve the intermediate values of each set $X \setminus \{i\}$, $i \in X$, and assemble the solution using Algorithm U. Call this evaluation strategy Algorithm C.

Algorithm C together with (5), (7), and Lagrangian interpolation establishes that the Tutte polynomial $T_G(x, y)$ can be computed in time and space $\sigma(G)n^{O(1)}$. This proves Theorem 1(a).

4.2. **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 $\emptyset \neq U \subsetneq W$ 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

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

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,

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

The recursion defined by (13) and (14) can now be evaluated for $|W| = 1, 2, \ldots, n$ in total $2^n n^{O(1)}$ 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 analogues of Algorithms U and C running in $\sigma(G)n^{O(1)}$ ring operations can be developed in this context; we describe an implementation of this in Appendix D. However, it is not immediate whether a polynomial-space algorithm for the Tutte polynomial can be developed based on (13) and (14).

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), to appear.

[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, 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] 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.

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

[17] H. Gebauer, Y. Okamoto, *Fast exponential-time algorithms for the forest counting in graph classes*, Theory of Computing 2007, Proceedings of the 13th Computing: The Australasian Theory Symposium (CATS 2007), Ballarat, Victoria, Jan 30–Feb 2, 2007, Conferences in Research and Practice in Information Technology 65, Australian Computer Society, 2007, pp. 63–69.

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

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

[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*, Proceedings of the 39th Annual ACM Symposium on Theory of Computing (San Diego, CA, June 11–13, 2007), Association for Computing Machinery, 2007, pp. 459–468.

[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, and 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] D. E. Knuth, *The Stanford GraphBase: A Platform for Combinatorial Computing*, Association for Computing Machinery, 1993.

[31] 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.

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

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

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

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

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

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

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

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

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

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

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

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

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

Appendix

Appendix A. Proofs

A.1. **Proof of Theorem 3.** This proof of the Fortuin–Kasteleyn identity (7) is well known (e.g. [39]) and is here included only for convenience of verification.

Proof. Expanding the product over E and changing the order of summation,

$$Z_G^{\mathrm{Potts}}(q,r) = \sum_{s:V\to\{1,2,\ldots,q\}} \prod_{e\in E} \left(1 + r_e\delta_e^s\right) = \sum_{F\subseteq E} \sum_{s:V\to\{1,2,\ldots,q\}} \prod_{e\in F} r_e\delta_e^s \,.$$

The right-hand side product evaluates to zero unless s is constant on each connected component of the graph with vertex set V and edge set F. Because there are q choices for the value of s on each connected component,

$$\sum_{F\subseteq E} \sum_{s:V\to\{1,2,\ldots,q\}} \prod_{e\in F} r_e\delta_e^s = \sum_{F\subseteq E} q^{c(F)} \prod_{e\in F} r_e = Z_G(q,r)\,.$$

□

A.2. **Proof of Lemma 6.** It is convenient to start with a preliminary lemma.

Lemma 7. *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 (11),

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

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 (11) 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,

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

□

We now proceed with the proof of Lemma 6.

Proof. Consider an arbitrary q-tuple $(U_1,U_2,\ldots,U_q)$ with $U_1,U_2,\ldots,U_q\subseteq X$ and $\cup_{j=1}^q U_j \equiv_i X$. Because $\{X_1,X_2,\ldots,X_s\}$ is a partition of X, we have $\cup_{j=1}^q U_j\equiv_i X$ if and only if $X_k\cap \cup_{j=1}^q U_j \equiv_i X_k\cap X$ holds for all $k=1,2,\ldots,s$. Put otherwise, we have $\cup_{j=1}^q U_j\equiv_i X$ if and only if $\cup_{j=1}^q (X_k\cap U_j)\equiv_i X_k$ holds for all $k=1,2,\ldots,s$. Using Lemma 7 for each U_j in turn, we have, by commutativity of R, the unique factorisation into pairwise intersections

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

The claim follows because $(U_1,U_2,\ldots,U_q)$ was arbitrary. □

Appendix B. A time–space tradeoff via split transforms

This appendix outlines a "split transform" algorithm that enables a time–space tradeoff in evaluating $\big((f\zeta)^q\mu\big)(V)$ 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

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

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 $\big((f\zeta)^q\mu\big)(V)$, split the outer Moebius transform and the inner zeta transform to get

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

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 $\big((f\zeta)^q\mu\big)(V)$ by storing at most $2^{n_2}n_2$ ring elements and executing at most

$$n^{O(1)}\sum_{Y_1\subseteq V_1}\big(2^{n_2}n_2 + 2^{|Y_1|}2^{n_2}\big) = n^{O(1)}(3^{n_1} + 2^{n_1}n_2)2^{n_2}$$

ring operations. This completes the description and analysis of the split transform algorithm.

The split transform algorithm together with (5), (7), and Lagrangian interpolation proves Theorem 1(c).

APPENDIX C. THE COVER POLYNOMIAL

Let D be a digraph with vertex set $V = \{1, 2, \dots, n\}$. Note that D may have parallel edges and loops. We assume that the number of edges is $n^{O(1)}$. 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* 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 8. *The cover polynomial of an n-vertex directed graph can be computed*

(a) *in time and space $2^n n^{O(1)}$; and*
(b) *in time $3^n n^{O(1)}$ and polynomial space.*

The proof 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, 31], 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,

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

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

We next derive an alternative expression using the principle of inclusion and exclusion. To this end, it is convenient to define for every $U \subseteq V$ the polynomials

$$P(U; z) = \sum_{X \subseteq U} p(X) z^{|X|} \quad \text{and} \quad C(U; z) = \sum_{X \subseteq U} c(X) z^{|X|}$$

in an indeterminate z; if viewed as set functions, $P(U;z)$ and $C(U;z)$ are zeta transforms of the set functions $p(X)z^{|X|}$ and $c(X)z^{|X|}$, respectively. We can now write

$$c_D(i,j) = \frac{1}{i!j!}\sum_{U\subseteq V}(-1)^{|V\setminus U|}\{z^n\}\big(P(U;z)^i C(U;z)^j\big)\,.$$

It remains to show how to compute the $p(X)$ and $c(X)$ for all $X \subseteq V$. 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,

$$p(X) = \sum_{1\le s\le t\le n}\sum_{S\subseteq X}(-1)^{|X\setminus S|}w(S,s,t,|X|-1)\,.$$

Similarly,

$$c(X) = \sum_{S\subseteq X}(-1)^{|X\setminus S|}w(S,s,s,|X|)\,,\quad \text{where } s=\min S\,.$$

Observing that $w(S,s,t,\ell)$ can be computed in time $n^{O(1)}$, we have that $c_D(i,j)$ can be computed in space $n^{O(1)}$ and time $4^n n^{O(1)}$.

To get an algorithm running in $3^n n^{O(1)}$ time and $n^{O(1)}$ space, observe that

$$P(U;z) = \sum_{S\subseteq U}P(U,S;z)$$

where

$$P(U,S;z) = \sum_{1\le s\le t\le n}\sum_{k=0}^{|U\setminus S|}\binom{|U\setminus S|}{k}(-1)^k z^{|S|+k}w(S,s,t,|S|+k-1)$$

and

$$C(U;z) = \sum_{S\subseteq U}C(U,S;z)$$

where

$$C(U,S;z) = \sum_{k=0}^{|U\setminus S|}\binom{|U\setminus S|}{k}(-1)^k z^{|S|+k}w(S,s,s,|S|+k)\,,\quad \text{where } s=\min S\,.$$

This establishes part (b) of the theorem.

For part (a), we show how to evaluate $c_D(i,j)$ in time and space $2^n n^{O(1)}$. Namely, p and c can be computed in time and space $2^n n^{O(1)}$ via fast Moebius inversion. Given p and c, the polynomials P and C can be computed in time and space $2^n n^{O(1)}$ via fast zeta transform. And finally, given P and C, the inclusion–exclusion expression of $c_D(i,j)$ can be evaluated in time $2^n n^{O(1)}$.

Appendix D. Tutte polynomials of concrete graphs

D.1. **Algorithm implementation.** Our implementation of the algorithm described in §4.2 uses a number of extra techniques to reduce the polynomial factors in the time and memory requirements. In what follows we assume that G is a connected graph.

(1) The coefficients t_{ij} of the Tutte polynomial are computed modulo a small integer p; the computation is repeated for sufficiently many different (pairwise coprime) p to enable recovery of the coefficients via the Chinese Remainder Theorem. The number of different p required is determined based on the available word length and

using $\tau(G)$ (computed via the Matrix–Tree Theorem) as an upper bound for the coefficients.

(2) To save a factor of m in memory, instead of direct computation with bivariate polynomials, we compute with univariate evaluations of the polynomials at $z = 0, 1, \ldots, m$, and finally recover only the necessary bivariate polynomials from the evaluations via Lagrange interpolation.

(3) To save a further factor of n^2 in memory, we execute the analogue of Algorithm U for subsets X in a specific order, namely in the lexicographic order. This enables efficient "in-place" computation of the polynomials $F(X, k, i)$ so that, for each X, the polynomials $F(X, k, i)$ need to be stored only for one value of i at the time. Furthermore, we never need all $F(X, k, i)$ for $k = 2, 3, \ldots, n$ explicitly, only a linear combination of them, so we count with this instead; however, we omit the details in this abstract.

The source code of the algorithm implementation is available by request. The implementation uses the GNU Multiple Precision Arithmetic library ⟨`http://gmplib.org/`⟩ for computation with large integers. The computed coefficients t_{ij} are checked for consistency by verifying that $\sum_{i,j} t_{ij} = \tau(G)$ and that $\sum_{i,j} 2^{i+j} t_{ij} = 2^m$.

D.2. **Performance.** The current algorithm implementation uses roughly $2^{n+1}n$ words of memory for an n-vertex graph, which presents a basic obstacle to practical performance. For example, the practical limit is at $n = 25$, assuming 32 GB of main memory and 64-bit words. This makes our polynomial space and time–space tradeoff algorithms from Theorem 1(b,c) interesting also from a practical perspective. At the time of writing, we have implemented the former, but not yet performed large-scale experiments with it.

In terms of running time, the complete graph K_n presents the worst case for n-vertex inputs for our algorithm. On a 3.66GHz Intel Xeon CPU with 1MB cache, computing the Tutte polynomial of K_{17} takes less than an hour, K_{18} takes about three hours, and K_{22} takes 96 hours. In comparison, both deletion–contraction and spanning tree enumeration cease to be practical well below this; for example, $\tau(K_{22}) = 705429498686404044207947776$ and $\tau(K_{16}) = 72057594037927936$; a survey of how to compute T_G in practice [24] reports running times for the complete graph K_{14} in hours. The fastest current program to compute Tutte polynomials [22] is also based on deletion–contraction with isomorphism rejection, but uses many other ideas as well. It processes K_{14} and many sparse graphs with far larger n in a few seconds, but also ceases to be practical for some dense graphs with $n = 16$, see Figure 2.

Two further remarks are in order. First, for (connected) graphs with a small $\tau(G)$, enumeration of spanning trees is faster than our algorithm. Second, graphs with fewer edges are faster to solve using our algorithm. For example, a 3-regular graph on 22 vertices can be solved in about five hours.

D.3. **Tutte polynomials of some concrete graphs.** Even though few readers are likely to derive any insight from the fact that the coefficient of x^2y^2 in the Tutte polynomial of Loupekine's Second Snark is 991226, we feel it germane to our paper to actually compute some Tutte polynomials. We include tables of the nonzero coefficients t_{ij} in the expansion (2) for a number of graphs. Among these, the values for the Petersen graph are well known [4, §13b] and are included here for verification only. For reference, we present the Tutte polynomials of a few other well-known graphs, mostly snarks and cages; however, these graphs are fairly sparse and exhibit symmetries that make them amenable to many of

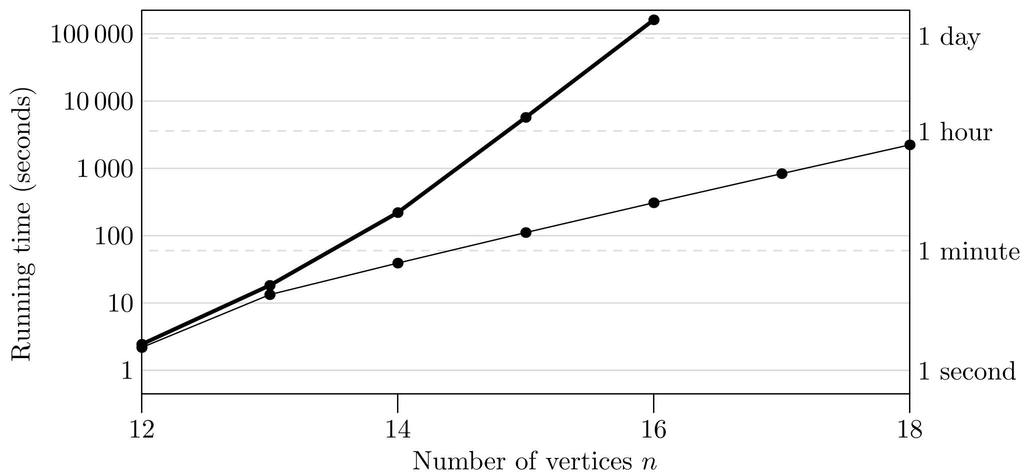
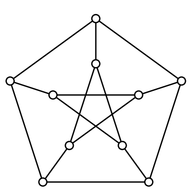


Figure 2. Running times for complements of random 4-regular graphs. The lines show averages of 5 runs on a 3.66GHz Intel Xeon CPU with 1MB cache. The thin line is our algorithm; the thick line is the algorithm of Haggard, Pearce, and Royle [22].

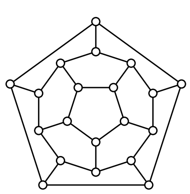
the previously existing techniques. An entertaining example that tests the liminations of our current implementation is from Knuth's Stanford Graph Base [30], based on the encounters between the 23 most important characters in Twain's *Huckleberry Finn*. This graph has 23 vertices, 88 edges, and 54540490752786432 spanning trees; the required solution time is about 50 hours.

Petersen Graph



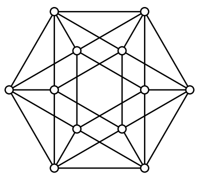
$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
0	36	84	75	35	9	1
36	168	171	65	10		
120	240	105	15			
180	170	30				
170	70					
114	12					
56						
21						
6						
1						

Dodecahedron



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$
0	4 412	17 562	30 686	31 540	21 548	10 439	3 693	950	170	19	1
4 412	38 864	95 646	115 448	82 550	38 322	12 046	2 542	330	20		
25 714	128 918	218 682	185 071	90 860	27 825	5 390	610	30			
72 110	245 880	295 915	174 870	57 735	11 230	1 240	60				
131 380	320 990	275 910	112 365	24 140	2 775	140					
176 968	316 256	193 791	53 350	7 175	468	12					
189 934	250 692	108 884	19 810	1 620	60						
170 690	167 140	50 850	5 870	270							
132 920	96 400	19 980	1 350	30							
91 740	48 710	6 510	220								
56 852	21 530	1 674	20								
31 792	8 198	306									
16 016	2 610	30									
7 216	660										
2 871	120										
989	12										
286											
66											
11											
1											

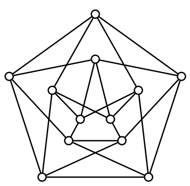
Icosahedron



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$	$j=12$
0	4 412	25 714	72 110	131 380	176 968	189 934	170 690	132 920	91 740	56 852	31 792	16 016
4 412	38 864	128 918	245 880	320 990	316 256	250 692	167 140	96 400	48 710	21 530	8 198	2 610
17 562	95 646	218 682	295 915	275 910	193 791	108 884	50 850	19 980	6 510	1 674	306	30
30 686	115 448	185 071	174 870	112 365	53 350	19 810	5 870	1 350	220	20		
31 540	82 550	90 860	57 735	24 140	7 175	1 620	270	30				
21 548	38 322	27 825	11 230	2 775	468	60						
10 439	12 046	5 390	1 240	140	12							
3 693	2 542	610	60									
950	330	30										
170	20											
19												
1												

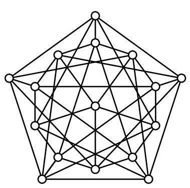
$j=13$	$j=14$	$j=15$	$j=16$	$j=17$	$j=18$	$j=19$
7 216	2 871	989	286	66	11	1
660	120	12				

Chvátal Graph



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$	$j=12$	$j=13$
0	1 994	7 349	12 626	14 115	11 903	8 140	4 642	2 211	869	274	66	11	1
1 994	12 782	25 969	28 952	22 250	13 164	6 202	2 292	636	120	12			
7 427	25 604	32 754	24 116	12 508	4 882	1 386	258	24					
12 339	26 004	20 914	9 804	3 166	672	72							
12 360	15 865	7 568	2 040	319	17								
8 445	6 216	1 552	184	4									
4 191	1 572	158	2										
1 559	240	4											
438	17												
91													
13													
1													

Clebsch Graph

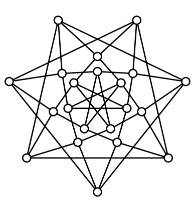


$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$
0	1 872 172	9 112 614	21 717 820	34 847 530	43 384 468	45 431 208	42 011 212	35 302 105	27 382 885
1 872 172	15 110 476	43 880 542	75 108 240	93 048 150	93 485 328	81 408 316	63 725 936	45 628 390	30 079 420
7 870 034	38 438 772	79 492 384	103 270 060	101 400 130	83 435 332	60 699 616	39 921 392	23 866 000	12 946 480
15 033 470	51 332 560	78 503 860	78 511 920	61 562 510	41 488 560	24 896 600	13 374 520	6 392 880	2 691 440
17 576 840	43 215 300	49 009 780	37 661 120	23 461 820	12 724 460	6 070 620	2 514 620	886 920	259 240
14 236 468	25 097 376	20 801 316	12 109 800	5 860 600	2 443 840	859 832	245 520	54 000	8 240
8 544 936	10 555 976	6 205 768	2 654 560	946 240	275 672	61 328	9 200	720	
3 958 696	3 293 168	1 305 736	386 480	90 320	14 800	1 360			
1 451 495	765 300	187 860	34 000	3 860	140				
427 155	130 280	16 860	1 360						
101 355	15 488	720							
19 283	1 152								
2 885	40								
325									
25									
1									

$j=10$	$j=11$	$j=12$	$j=13$	$j=14$	$j=15$	$j=16$	$j=17$	$j=18$	$j=19$	$j=20$	$j=21$	$j=22$	$j=23$	$j=24$
19 759 258	13 306 232	8 367 140	4 907 540	2 678 480	1 355 496	632 942	270 930	105 400	36 840	11 388	3 044	680	120	15
18 276 422	10 217 568	5 236 520	2 446 480	1 033 720	390 712	130 088	37 320	8 920	1 680	224	16			
6 341 100	2 783 100	1 082 640	367 440	106 320	25 320	4 680	600	40						
983 800	305 640	78 080	15 520	2 160	160									
60 100	10 220	1 080	40											
672														

$j=25$
1

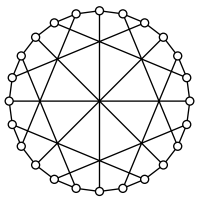
Brinkmann Graph



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$
0	9 135 298	41 648 660	91 803 040	134 604 309	151 187 372	140 741 055	113 681 473	81 746 167
9 135 298	81 926 895	239 055 379	397 141 486	465 831 800	432 446 574	338 941 057	232 203 883	141 329 485
49 413 533	266 495 740	573 449 072	748 656 482	714 722 253	551 889 933	363 367 900	208 849 858	105 525 301
127 008 274	489 646 682	809 702 257	842 806 153	658 050 897	421 620 731	230 681 654	109 043 291	44 426 158
208 645 102	603 283 289	776 194 328	645 651 909	410 955 629	216 136 928	96 269 502	36 267 791	11 403 399
247 964 242	545 064 597	545 163 829	360 374 231	184 948 316	78 214 626	27 429 213	7 843 003	1 773 968
228 346 378	380 867 123	293 316 401	152 129 831	61 696 341	20 281 751	5 314 498	1 066 233	153 321
170 148 325	212 835 902	124 062 007	49 440 398	15 337 490	3 715 740	668 668	81 926	5 749
105 629 121	97 136 821	41 782 196	12 395 327	2 795 009	457 702	48 664	2 667	28
55 758 397	36 640 429	11 223 611	2 363 508	356 678	33 824	1 505	7	
25 384 606	11 470 466	2 380 728	330 351	28 840	1 113			
10 061 144	2 969 085	388 661	31 304	1 155				
3 489 936	626 955	46 361	1 680	7				
1 060 656	105 035	3 626	28					
281 455	13 239	140						
64 604	1 127							
12 601	49							
2 024								
253								
22								
1								

$j=9$	$j=10$	$j=11$	$j=12$	$j=13$	$j=14$	$j=15$	$j=16$	$j=17$	$j=18$	$j=19$	$j=20$	$j=21$	$j=22$
53 017 571	31 181 857	16 645 377	8 048 376	3 509 821	1 371 591	476 045	144 970	38 094	8 435	1 519	210	20	1
76 846 499	37 308 376	16 100 987	6 129 326	2 034 949	579 747	138 453	26 754	3 948	399	21			
46 789 365	18 078 277	6 016 780	1 695 687	394 632	72 996	10 080	924	42					
15 447 215	4 510 996	1 080 072	204 344	28 756	2 688	126							
2 926 959	592 536	89 355	9 009	462									
302 344	35 602	2 401	49										
13 951	588												
140													

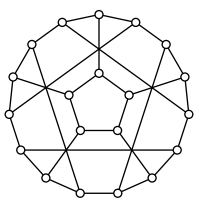
McGee Graph



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$
0	100 424	348 008	546 092	537 899	382 951	210 826	92 060	31 878	8 602	1 748	252
100 424	863 904	1 945 060	2 252 476	1 697 518	930 400	387 550	122 924	28 908	4 764	492	24
616 320	2 984 380	4 833 738	4 237 698	2 455 880	1 027 312	316 166	69 834	10 404	924	36	
1 853 724	6 149 836	7 421 464	4 969 160	2 208 876	694 880	153 316	22 388	1 920	72		
3 683 515	8 896 534	8 101 789	4 145 382	1 401 096	323 908	48 978	4 326	168			
5 484 441	9 867 514	6 792 829	2 639 544	666 144	108 612	10 308	432				
6 563 798	8 854 364	4 577 890	1 330 356	241 640	25 476	1 188					
6 600 622	6 650 972	2 543 854	536 856	65 472	3 720	32					
5 745 907	4 272 590	1 178 731	172 060	12 502	258						
4 420 661	2 377 190	455 693	42 648	1 488							
3 050 680	1 152 488	145 600	7 728	80							
1 908 584	486 960	37 584	904								
1 090 666	178 182	7 488	48								
572 080	55 608	1 048									
276 100	14 380	80									
122 600	2 920										
49 938	418										
18 532	32										
6 188											
1 820											
455											
91											
13											
1											

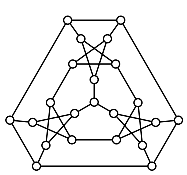
$j=12$	$j=13$
23	1

Flower Snark



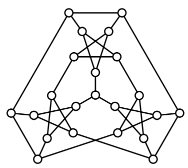
$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$
0	7 878	26 617	39 815	36 190	22 832	10 624	3 704	950	170	19	1
7 878	61 874	129 158	134 515	86 880	38 396	11 938	2 531	330	20		
43 135	187 515	268 795	198 500	90 385	27 215	5 335	610	30			
114 690	332 265	336 780	176 070	55 570	10 965	1 240	60				
200 340	407 935	293 485	107 135	22 700	2 715	140					
261 282	379 816	191 744	47 405	6 285	369	1					
273 073	282 743	97 651	15 540	1 145	20						
239 007	173 800	39 500	3 760	125							
180 402	89 925	12 730	640	5							
119 792	39 505	3 220	65								
70 904	14 713	615									
37 697	4 580	80									
18 052	1 150	5									
7 767	215										
2 977	25										
1 000	1										
286											
66											
11											
1											

Loupekine's First Snark



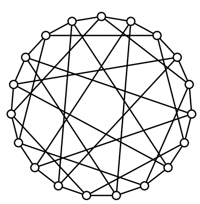
$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$	$j=12$
0	20 724	75 758	124 770	126 268	89 312	46 998	18 864	5 766	1 309	209	21	1
20 724	176 872	412 502	490 516	367 014	190 328	71 133	19 116	3 540	407	22		
121 838	591 662	981 968	862 535	475 649	177 299	45 315	7 638	759	33			
350 010	1 169 865	1 430 045	930 845	373 695	97 491	16 253	1 568	66				
663 435	1 617 133	1 470 305	703 524	201 006	35 166	3 509	154					
941 666	1 709 810	1 153 310	398 793	78 432	8 688	465	6					
1 073 720	1 460 522	723 379	176 425	22 916	1 512	38						
1 028 153	1 044 783	373 894	62 282	5 068	189	3						
852 031	641 221	162 202	17 622	825	15							
623 999	342 967	59 554	3 891	81								
409 765	161 263	18 481	603									
243 580	66 826	4 810	48									
131 786	24 328	1 050										
65 014	7 703	195										
29 187	2 073	30										
11 845	451	3										
4 291	71											
1 359	6											
364												
78												
12												
1												

Loupekine's Second Snark



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$	$j=12$
0	21 156	76 946	126 048	126 968	89 520	47 030	18 866	5 766	1 309	209	21	1
21 156	180 076	417 674	493 856	367 930	190 386	71 117	19 114	3 540	407	22		
124 286	601 016	991 226	865 641	475 731	177 183	45 303	7 638	759	33			
356 730	1 185 628	1 439 086	931 623	373 263	97 431	16 253	1 568	66				
675 496	1 635 022	1 475 257	702 666	200 712	35 160	3 509	154					
957 769	1 724 581	1 154 068	397 725	78 312	8 682	465	6					
1 090 933	1 469 555	721 869	175 617	22 816	1 494	36						
1 043 540	1 048 408	371 796	61 744	5 000	183	3						
863 802	641 304	160 464	17 346	801	15							
631 780	341 544	58 494	3 793	75								
414 216	159 732	17 991	585									
245 775	65 772	4 641	48									
132 710	23 776	1 008										
65 338	7 475	189										
29 277	2 001	30										
11 863	435	3										
4 293	69											
1 359	6											
364												
78												
12												
1												

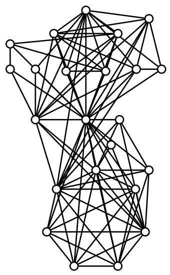
Robertson Graph



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$
0	1 437 372	6 220 100	12 943 266	17 896 018	18 984 001	16 719 144	12 772 526	8 656 420	5 255 640
1 437 372	12 029 428	32 563 392	50 017 890	54 312 237	46 813 703	34 091 282	21 611 154	12 056 388	5 923 384
7 246 700	35 805 218	69 906 011	82 616 390	71 636 757	50 385 897	30 117 376	15 535 199	6 911 373	2 628 658
17 211 692	59 406 320	86 949 004	80 075 156	55 598 473	31 674 483	15 229 199	6 178 619	2 087 824	574 674
25 936 913	65 327 035	72 306 448	51 859 116	28 598 913	12 922 375	4 813 247	1 450 990	342 608	59 983
28 091 119	52 063 835	43 319 066	23 931 700	10 261 301	3 532 549	955 120	193 386	26 967	2 161
23 425 656	31 670 274	19 471 383	8 099 628	2 594 341	634 476	110 940	12 294	642	
15 702 294	15 160 966	6 691 382	2 015 912	449 940	68 992	6 150	204		
8 704 413	5 805 523	1 760 310	360 113	49 420	3 652	72			
4 067 425	1 786 531	348 285	43 247	2 802	36				
1 622 042	438 574	49 497	2 982	36					
555 756	83 920	4 552	72						
163 804	11 910	204							
41 322	1 128								
8 801	54								
1 540									
210									
20									
1									

$j=10$	$j=11$	$j=12$	$j=13$	$j=14$	$j=15$	$j=16$	$j=17$	$j=18$	$j=19$	$j=20$
2 865 546	1 400 474	610 470	235 467	79 458	23 085	5 643	1 121	171	18	1
2 549 764	952 258	303 943	81 073	17 461	2 869	323	19			
841 681	221 559	46 208	7 182	741	38					
124 393	19 969	2 128	114							
7 011	418									
54										

Book("huck", 23, 0, 0, 0, 1, 1, 0)



$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
0	7 644 119 040	80 523 030 528	427 469 042 304	1 534 015 292 832	4 208 425 319 232
7 644 119 040	130 942 365 696	911 365 748 352	3 881 430 693 504	11 932 392 081 456	28 958 733 158 880
58 063 454 208	714 446 189 568	4 072 619 596 896	14 940 911 766 816	40 626 720 391 896	88 689 338 750 268
208 089 907 200	2 122 537 340 160	10 443 363 681 456	33 814 295 549 328	82 361 720 634 568	162 872 521 275 352
468 472 356 864	4 123 092 911 040	17 837 338 328 976	51 446 928 857 016	112 779 195 195 632	202 515 265 428 917
743 860 850 688	5 729 638 717 440	21 934 086 338 856	56 496 623 747 504	111 536 656 897 631	181 822 039 012 474
886 362 588 672	5 999 274 355 776	20 332 420 289 546	46 713 317 299 566	82 898 207 721 802	122 436 175 778 571
823 088 010 752	4 889 305 364 240	14 624 737 351 937	29 857 553 469 301	47 452 347 271 542	63 289 339 398 041
610 456 680 992	3 167 538 981 020	8 313 631 620 163	14 994 835 055 266	21 231 473 215 833	25 460 955 613 478
367 568 054 960	1 653 497 026 704	3 776 758 163 442	5 971 670 244 733	7 483 614 951 821	8 026 773 100 789
181 618 268 880	701 112 177 494	1 378 712 232 797	1 892 644 074 081	2 082 708 405 703	1 985 823 978 447
74 123 982 824	242 350 495 650	404 732 343 621	476 705 833 342	456 394 199 836	384 178 991 077
25 065 464 810	68 266 261 385	95 166 437 501	94 815 567 757	78 141 581 921	57 622 694 859
7 022 616 847	15 600 933 383	17 760 717 539	14 716 870 862	10 314 463 582	6 604 985 046
1 625 058 718	2 866 517 167	2 590 259 291	1 749 939 461	1 028 275 760	565 949 893
308 561 221	417 203 235	288 051 462	155 118 437	75 041 770	35 042 180
47 562 773	46 993 465	23 498 332	9 852 285	3 809 990	1 483 706
5 855 899	3 949 677	1 318 791	422 829	122 002	38 785
562 021	233 095	45 170	11 144	1 895	485
40 503	8 615	704	144		
2 061	150				
66					
1					

$j=6$	$j=7$	$j=8$	$j=9$	$j=10$	$j=11$	$j=12$	$j=13$
9 456 659 968 488	18 198 074 885 356	30 940 731 400 736	47 560 610 835 303	67 267 401 905 395	88 747 842 449 432	110 413 663 746 802	130 666 491 664 749
58 691 487 844 000	103 243 788 227 460	162 102 328 114 667	231 997 488 723 635	307 596 180 791 466	382 634 254 447 952	451 056 497 992 889	507 874 432 372 429
163 670 254 561 170	264 694 234 344 442	385 158 972 944 741	514 385 945 146 530	640 213 989 471 637	751 445 805 849 377	839 500 833 190 661	899 138 156 074 953
274 803 257 750 421	409 580 434 157 718	553 143 680 550 286	689 925 324 841 686	806 372 976 235 758	893 029 971 410 258	945 126 777 109 455	962 134 122 445 120
312 764 825 917 711	429 835 626 654 435	538 846 162 737 722	627 608 139 462 721	688 582 006 779 197	719 045 082 827 230	720 198 797 592 595	695 912 053 117 113
256 878 631 331 512	325 285 245 146 851	378 230 381 624 169	411 029 358 220 644	422 898 675 284 303	415 871 654 487 327	393 582 445 653 266	360 278 181 979 195
157 948 712 034 296	183 994 333 180 127	198 149 002 209 146	200 617 849 105 619	193 255 210 084 387	178 629 421 459 750	159 375 433 931 029	137 832 995 815 031
74 347 451 684 553	79 489 180 483 326	79 122 703 627 690	74 482 208 530 196	67 024 385 760 956	58 078 504 339 287	48 701 630 168 789	39 652 159 130 588
27 140 739 158 980	26 556 188 103 022	24 368 441 736 674	21 270 851 688 207	17 826 935 293 467	14 431 482 593 462	11 328 241 057 760	8 643 948 300 111
7 732 248 143 990	6 900 191 730 740	5 818 100 553 716	4 692 871 669 682	3 648 698 696 018	2 747 096 198 020	2 008 371 282 725	1 428 133 964 932
1 720 587 723 236	1 394 731 774 739	1 076 252 268 769	798 621 154 805	573 141 852 737	399 057 840 816	270 019 505 090	177 714 498 469
297 808 086 966	218 231 222 093	153 327 979 453	104 075 037 503	68 506 947 878	43 803 197 182	27 225 629 630	16 453 312 517
39 723 622 606	26 157 204 332	16 622 431 041	10 246 508 473	6 137 792 945	3 573 570 819	2 022 552 038	1 112 678 888
4 020 794 424	2 360 083 348	1 344 595 660	745 813 853	402 557 913	211 208 919	107 776 845	53 497 484
301 529 428	155 871 340	78 646 816	38 768 901	18 595 383	8 660 607	3 932 249	1 739 850
16 128 290	7 184 647	3 151 857	1 352 207	561 013	224 483	88 116	33 483
575 231	210 107	77 523	27 469	9 090	2 783	859	234
11 796	3 005	862	206	33			
86							

$j=14$	$j=15$	$j=16$	$j=17$	$j=18$	$j=19$	$j=20$	$j=21$
148 114 503 232 862	161 710 933 451 019	170 813 927 289 507	175 183 508 018 224	174 936 105 230 968	170 475 040 838 235	162 410 989 874 779	151 482 203 761 928
549 640 583 945 429	574 575 879 264 822	582 448 562 263 319	574 307 823 880 181	552 153 562 397 919	518 597 408 034 174	476 549 124 971 777	428 948 242 539 842
928 434 404 732 381	928 302 660 980 981	901 802 324 461 765	853 408 768 453 265	788 339 037 061 129	711 981 542 551 881	629 449 797 206 205	545 264 254 521 678
946 802 956 316 230	904 076 789 608 510	840 089 439 442 061	761 337 884 261 647	674 053 687 878 178	583 766 784 630 227	495 042 677 879 502	411 367 624 748 301
651 526 551 686 197	592 898 453 733 529	525 705 387 527 193	454 990 478 155 838	384 901 878 667 513	318 587 168 246 233	258 204 924 355 942	205 018 507 541 185
320 142 255 061 978	276 885 385 561 807	233 534 388 363 802	192 359 454 489 345	154 895 824 102 503	122 026 125 706 545	94 096 567 995 260	71 045 733 735 891
115 885 591 893 630	94 920 219 709 713	75 855 471 078 622	59 206 435 569 975	45 166 410 690 866	33 691 771 782 446	24 580 817 533 069	17 540 994 280 576
31 417 553 021 990	24 262 330 193 502	18 280 905 313 399	13 447 888 149 463	9 661 849 230 137	6 780 631 931 834	4 647 801 211 901	3 110 824 903 986
6 421 813 488 934	4 649 813 570 718	3 283 163 455 432	2 261 157 582 200	1 518 926 901 479	994 941 569 885	635 198 727 558	394 989 616 510
988 695 612 555	666 682 299 140	437 890 395 312	280 092 719 250	174 390 514 342	105 615 119 324	62 160 275 509	35 513 089 086
113 804 068 059	70 899 825 376	42 952 342 270	25 286 305 141	14 453 004 774	8 011 736 241	4 301 315 057	2 232 878 866
9 665 564 617	5 515 962 845	3 055 400 762	1 641 278 936	854 165 572	430 162 989	209 314 543	98 214 601
594 641 211	308 435 706	155 158 798	75 679 090	35 781 667	16 390 015	7 263 816	3 106 323
25 811 941	12 095 111	5 504 691	2 437 347	1 050 538	440 428	178 942	69 836
748 818	312 758	126 758	50 082	19 202	7 108	2 503	808
12 227	4 231	1 375	428	121	31	7	1
53	8						

$j=22$	$j=23$	$j=24$	$j=25$	$j=26$	$j=27$	$j=28$	$j=29$
138 481 043 916 189	124 191 059 706 563	109 337 116 734 795	94 549 658 362 427	80 342 944 707 533	67 106 067 181 304	55 104 744 824 358	44 491 423 252 883
378 551 418 825 180	327 779 450 801 827	278 622 686 027 317	232 599 415 134 395	190 758 701 968 316	153 717 226 791 019	121 719 136 468 979	94 708 476 115 968
463 156 138 345 255	385 979 736 157 458	315 714 271 443 996	253 533 343 091 830	199 919 236 976 320	154 800 992 492 735	117 698 380 405 855	87 858 171 960 160
335 152 391 009 976	267 823 161 239 724	209 968 589 993 354	161 514 900 952 720	121 905 869 974 532	90 270 537 601 981	65 567 617 176 755	46 701 038 037 329
159 540 234 730 431	121 698 403 452 623	91 005 190 493 607	66 709 745 029 275	47 926 985 451 590	33 737 964 980 454	23 261 782 819 227	15 701 786 388 350
52 529 965 418 036	38 034 479 409 317	26 964 153 070 620	18 711 968 201 815	12 706 067 491 372	8 438 244 908 275	5 477 591 996 969	3 473 196 092 061
12 241 838 274 716	8 353 299 039 147	5 570 757 950 696	3 629 009 765 916	2 307 827 423 945	1 431 651 832 106	865 610 892 679	509 618 740 844
2 032 190 992 977	1 294 966 296 360	804 342 907 429	486 565 736 128	286 375 736 913	163 814 154 069	90 962 163 938	48 964 445 869
239 034 177 695	140 635 158 743	80 349 078 122	44 519 508 330	23 887 371 320	12 391 822 214	6 204 115 029	2 991 921 656
19 668 089 937	10 542 647 994	5 459 753 151	2 726 196 170	1 309 543 395	603 613 124	266 217 054	111 984 386
1 118 570 285	539 500 322	249 848 063	110 746 672	46 804 725	18 772 999	7 105 935	2 521 253
44 326 408	19 181 752	7 926 985	3 111 957	1 152 202	398 423	127 018	36 746
1 276 909	502 053	187 522	65 933	21 528	6 401	1 684	380
25 927	9 067	2 949	883	239	56	10	1
231	56	10	1				

$j=30$	$j=31$	$j=32$	$j=33$	$j=34$	$j=35$	$j=36$	$j=37$
35 321 028 354 267	27 569 839 915 839	21 155 277 285 691	15 954 847 768 088	11 823 018 778 121	8 605 270 288 428	6 149 017 377 510	4 311 436 549 811
72 405 228 468 100	54 377 952 292 000	40 108 145 965 420	29 043 492 273 524	20 638 880 703 615	14 385 457 021 771	9 828 892 336 538	6 578 623 394 399
64 373 528 302 495	46 281 448 067 004	32 636 626 427 969	22 562 646 009 720	15 283 132 698 005	10 136 458 047 648	6 577 897 351 181	4 173 019 367 654
32 605 441 633 781	22 303 581 464 318	14 939 530 821 672	9 792 558 415 254	6 276 719 028 345	3 930 854 880 338	2 403 021 466 599	1 432 512 031 399
10 370 366 228 563	6 697 239 653 980	4 226 027 636 156	2 603 398 119 525	1 564 280 125 341	915 803 054 713	521 796 689 085	288 971 661 588
2 149 485 588 138	1 297 255 575 747	762 739 720 158	436 426 275 281	242 715 583 856	131 021 687 166	68 546 193 842	34 695 422 122
291 838 357 248	162 366 310 290	87 644 734 347	45 833 115 884	23 180 300 406	11 316 367 286	5 320 896 693	2 403 511 898
25 513 055 441	12 846 282 872	6 238 791 356	2 915 994 723	1 308 416 720	561 964 518	230 239 072	89 612 569
1 386 738 494	616 213 381	261 761 786	105 936 834	40 682 440	14 753 318	5 022 558	1 593 264
44 763 707	16 929 544	6 025 245	2 004 377	617 953	174 625	44 548	10 031
831 602	252 202	69 220	16 805	3 491	589	73	5
9 466	2 112	386	52	4			
72	11	1					

$j=38$	$j=39$	$j=40$	$j=41$	$j=42$	$j=43$	$j=44$	$j=45$	$j=46$
2 964 474 698 957	1 997 470 496 058	1 317 887 697 673	850 665 829 404	536 654 329 284	330 528 626 041	198 505 264 692	116 089 433 466	66 010 296 070
4 310 052 558 697	2 761 684 207 395	1 728 990 256 047	1 056 510 698 738	629 363 523 349	365 006 585 603	205 792 941 637	112 609 389 066	59 693 833 147
2 585 645 699 396	1 563 111 427 998	920 894 742 411	528 044 208 920	294 272 153 267	159 130 489 574	83 350 267 190	42 202 405 360	20 608 991 452
831 790 371 315	469 846 262 712	257 819 091 781	137 219 085 706	70 712 799 890	35 214 172 506	16 908 923 264	7 809 196 055	3 458 946 792
155 326 381 067	80 906 285 732	40 765 822 575	19 829 960 250	9 291 403 735	4 182 750 513	1 803 797 937	742 649 893	290 757 073
16 957 718 111	7 985 680 153	3 614 180 128	1 567 456 613	649 222 792	255 780 213	95 399 665	33 492 231	10 990 747
1 039 924 955	429 469 516	168 586 789	62 586 269	21 837 068	7 105 191	2 134 376	584 182	143 089
32 969 506	11 395 789	3 672 196	1 092 398	296 134	71 881	15 238	2 717	383
466 505	124 530	29 815	6 261	1 117	161	17	1	
1 925	297	33	2					

$j=47$	$j=48$	$j=49$	$j=50$	$j=51$	$j=52$	$j=53$	$j=54$	$j=55$	$j=56$	$j=57$
36 432 673 762	19 480 581 012	10 069 527 361	5 019 400 819	2 406 133 696	1 105 659 524	485 226 772	202 490 441	79 941 549	29 674 484	10 280 210
30 590 692 119	15 119 155 354	7 187 420 893	3 276 272 681	1 426 882 777	591 250 737	231 936 330	85 622 599	29 531 012	9 430 560	2 757 016
9 681 466 532	4 362 140 777	1 878 612 930	770 201 912	299 178 401	109 478 461	37 477 493	11 898 600	3 465 267	912 617	213 215
1 464 504 260	590 430 015	225 629 513	81 283 454	27 422 357	8 592 757	2 474 897	646 355	150 277	30 308	5 101
107 745 798	37 582 223	12 256 579	3 706 816	1 029 033	258 783	57 926	11 258	1 830	234	21
3 342 402	931 895	234 974	52 641	10 232	1 668	216	20	1		
30 575	5 485	776	77	4						
38	2									

$j=58$	$j=59$	$j=60$	$j=61$	$j=62$	$j=63$	$j=64$	$j=65$	$j=66$
3 293 359	964 438	254 353	59 228	11 848	1 956	250	22	1
727 123	169 643	34 073	5 661	730	65	3		
43 020	7 205	940	85	4				
673	62	3						
1								