COMPUTING THE TUTTE POLYNOMIAL
IN VERTEX-EXPONENTIAL TIME

ANDREAS BJÖRLUND*, 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

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

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 $\exp(O(n))$, that is, in “vertex-exponential” time, while $\tau(G)$ typically is $\exp(\omega(n))$ and can be as large as $n^{n-2}$ [12]. 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 $3^n - s^{2s} n^{O(1)}$ and space $2^{s} n^{O(1)}$ for any integer $s$, $0 \leq s \leq n$.

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 $\tau(G) = \exp(O(n))$. Theorem 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 $\Delta$ 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 \(\exp(O(n))\) 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 \(\S 2.3\) 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 \((x - 1)(y - 1) = 2\), 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 \(\cite{34}\) nor the recent inclusion–exclusion algorithm \(\cite{8}\), 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 \(\cite{16, 39}\), 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 \(\cite{39}\) 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. 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 $m = nO(1)$ and remark that this assumption is implicit already in Theorem 1. (Without this assumption, all the time bounds require an additional multiplicative term $mO(1)$.) 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 $2^m nO(1)$ 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

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

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 $\tau(G)$.

1A 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 $2^n n^{O(1)}$.

The computational complexity of computing individual coefficients $t_{ij}$ has also been investigated. In particular, polynomial-time algorithms exist for $t_{n-k,j}$ 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. 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 $G \setminus e$ 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

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

The deletion–contraction algorithm defined by a direct evaluation of (3) leads to a running time that scales as the Fibonacci sequence, $((1 + \sqrt{5})/2)^{n+m} = O(1.6180^{n+m})$. Sekine, Imai, and Tani 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.

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. 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 (cf. [10, Theorem X.2]).

Besides deletion–contraction, many other expansions are known (in particular for restrictions of the Tutte polynomial; see [21]), even a convolution over the set of edges, 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: 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 $\exp(O(n))$ 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 \(\|\) involving \(c(F)\) cancel, so \(T_G(x, y) = (x - 1)^n y^m\), 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 \(2^{\omega/3} n^{O(1)} = O(1.732^n)\) and exponential space, where \(\omega\) is the exponent of matrix multiplication; this is yet a new application of Williams’s trick \([3, 32, 48]\).

**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 \(q^{n/\omega} n^{O(1)}\). 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 \(R_G(p) = p^{n-\tau} + (1 - p)^{n - \tau} T_G(1, 1/p)\) 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 \(P_G(t) = (-1)^{n-\tau} t^\tau T_G(1 - t, 0)\) 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 \(\exp(O(n))\) 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 \( \exp(O(\sqrt{n})) \) [33]. This works more generally, with a slight overhead: in classes of graphs with separators of size \( n^k \), the Tutte polynomial can be computed in time \( \exp(O(n^k \log n)) \).

**Bounded tree-width and branch-width:** For a fixed integer, if \( G \) has tree-width \( k \) then \( T_G \) can be computed in polynomial time [11 33]. This can be generalised to branch-width [23].

**Bounded clique-width and cographs:** For a fixed integer, if \( G \) has clique-width \( k \) then \( T_G \) can be computed in time \( \exp(O(n^{1/(k+2)})) \) [18]. 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 \( \Delta \) 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 \( \chi^\Delta n^{O(1)} \), 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 \( \tau(G) = O(\nu_k^{\nu_k^{-1}} n^{-1} \log n) \), 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 [11].

**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 [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. **Computational 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 $\exp\left(O\left(n/\log n\right)\right)$.

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 $\exp\left(\Omega(n \log q)\right)$ 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 \cite{[39]} of $G$ is the polynomial

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

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

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

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 \cite{[16, 39]}.

For a mapping $s : V \rightarrow \{1, 2, \ldots, q\}$ 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

$$Z^\text{Potts}_G(q, r) = \sum_{s:V \rightarrow \{1,2,\ldots,q\}} \prod_{e \in E} \left(1 + r_e \delta^e_s\right).$$

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

$$Z_G(q, r) = Z^\text{Potts}_G(q, r).$$

3.2. The multivariate Tutte polynomial via the $q$-state Potts model. By virtue of the Fortuin–Kasteleyn identity \cite{[16]}, to compute $Z_G(q, r)$ it suffices to evaluate

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

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 \(Z_G^{\text{Potts}}(q, r)\) in a more convenient form. For \(X \subseteq V\), denote by \(G[X]\) the subgraph of \(G\) induced by \(X\), and let
\[(8)\]
\[f(X) = \prod_{e \in E(G[X])} (1 + r_e).\]
For \(q = 1, 2, \ldots\), we have
\[(9)\]
\[Z_G^{\text{Potts}}(q, r) = \sum_{(U_1, U_2, \ldots, U_q)} f(U_1)f(U_2) \cdots f(U_q),\]
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 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 \(f : 2^V \to R\) is defined for all \(Y \subseteq V\) by \(f_\zeta(Y) = \sum_{X \subseteq Y} f(X)\). The Moebius transform \(f : 2^V \to R\) is defined for all \(X \subseteq V\) by \(f_\mu(X) = \sum_{Y \subseteq X} (-1)^{\abs{X \setminus Y}} f(Y)\).

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
\[(10)\]
\[\left((f_\zeta)^q \mu\right)(V) = \sum_{(U_1, U_2, \ldots, U_q)} f(U_1)f(U_2) \cdots f(U_q),\]
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 \(3^n n^{O(1)}\) ring operations by storing \(n^{O(1)}\) ring elements. Using the fast zeta and Moebius transforms, \((f_\zeta)^q \mu(V)\) 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 \(f_z(X) = f(X)z^\abs{X}\). Now evaluate the \(z\)-polynomial \((f_z_\zeta)^q \mu(V)\) and look at the coefficient of the monomial \(z^\abs{V}\), 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 \((f_\zeta)^q \mu(V)\) 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 \(i = \max X \setminus Y\). □
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

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

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

Note that $F(V, q, 0) = ((f \zeta \mu)(V))$. 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

$$F(X, q, n) = \left(f(X) + \sum_{i \in X} F(X \setminus \{i\}, 1, i-1)\right)^q.$$[[ 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 $q = 1, 2, 3, \ldots, n+1$ 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 $Y = \bigcup_{j=1}^{q} U_j$. 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 $Y \equiv X$ 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 (11)

$$f(X) = f(X_1)f(X_2) \cdots f(X_s)$$

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,

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

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

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 $2^n n^{O(1)}$ 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,

\[
Z_G^{\text{Potts}}(q,r) = \sum_{s:V \rightarrow \{1,2,\ldots,q\}} \prod_{e \in E} (1 + r_e \delta_e^s) = \sum_{F \subseteq E} \sum_{s:V \rightarrow \{1,2,\ldots,q\}} \prod_{e \in F} r_e \delta_e^s.
\]

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, \[\sum_{F \subseteq E} \sum_{s:V \rightarrow \{1,2,\ldots,q\}} \prod_{e \in F} r_e \delta_e^s = \sum_{F \subseteq E} q^{|F|} \prod_{e \in F} r_e = Z_G(q,r).\] □

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 \(\bigcup_{j=1}^{q} U_j \equiv X\). Because \(\{X_1, X_2, \ldots, X_s\}\) is a partition of \(X\), we have \(\bigcup_{j=1}^{q} U_j \equiv X_k\) if and only if \(X_k \cap \bigcup_{j=1}^{q} U_j \equiv X_k\) holds for all \(k = 1,2,\ldots,s\). Put otherwise, we have \(\bigcup_{j=1}^{q} U_j \equiv X\) if and only if \(\bigcup_{j=1}^{q} (X_k \cap U_j) \equiv X_k\) 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

\[
f(U_1)f(U_2) \cdots f(U_q) = \prod_{j=1}^{q} \prod_{k=1}^{s} f(U_j \cap X_k) = \prod_{k=1}^{s} \prod_{j=1}^{q} f(U_j \cap X_k).
\]

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 \( (f_\zeta)^q \mu(V) \) 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 \( f(X) \) for \( f(X_1) \) 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,
\[
f_\zeta(Y) = \sum_{X \subseteq Y} f(X) = \sum_{X_1 \subseteq Y_1} \sum_{X_2 \subseteq Y_2} f(X_1, X_2) = \sum_{X_1 \subseteq Y_1} f_\zeta_2(X_1, Y_2) = f_\zeta \zeta_1(Y_1, Y_2)
\]
and
\[
f_\mu(X) = \sum_{Y \subseteq X} (-1)^{|X|\setminus|Y|} f(Y) = \sum_{X_1 \subseteq Y_1} \sum_{X_2 \subseteq Y_2} (-1)^{|X_1 \setminus Y_1|} \sum_{X_2 \subseteq Y_2} (-1)^{|X_2 \setminus Y_2|} f(Y_1, Y_2) = \sum_{X_1 \subseteq Y_1} (-1)^{|X_1 \setminus Y_1|} f_\mu_2(Y_1, X_2) = f_\mu \mu_1(X_1, X_2).
\]

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
\[
( (f_\zeta)^q \mu(V) = \sum_{Y_1 \subseteq V_1} (\sum_{Y_2 \subseteq V_2} (\sum_{X_2 \subseteq V_2} (f_\zeta_1 \zeta_2(Y_1, Y_2)))^q).
\]

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 at most
\[
n^{O(1)} \sum_{Y_1 \subseteq V_1} (2^{n_2} n_2 + 2^{|Y_1|} 2^{n_2}) = n^{O(1)} (3^{n_1} + 2^{n_1} n_2) 2^{n_2}
\]
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 \( V = \{1, 2, \ldots, n\} \). 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

\[
C_D(x, y) = \sum_{i,j} c_D(i, j)x^iy^j,
\]

where \( x^i = x(x-1)\cdots(x-i+1) \) and \( x^0 = 1 \). 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,

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

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

We 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

$$c_D(i,j) = \frac{1}{i!j!} \sum_{U \subseteq V} (-1)^{|V\setminus U|} \left(\sum_{S,s,t,\ell} P(U; z) C(U; z)^\ell \right).$$

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 $\ell$ 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,

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

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 $4n^{O(1)}$.

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

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

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 Möbius 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 $\sum_{i,j} 2^{i+j} t_{ij} = 2^n$.

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 [3, §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
the previously existing techniques. An entertaining example that tests the limitations 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 545409752786432 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 |
|-------|-------|-------|-------|-------|-------|-------|
| 36    | 120   | 180   | 170   | 140   | 114   | 114   |
| 114   | 114   | 114   | 114   | 114   | 114   | 114   |
| 114   | 114   | 114   | 114   | 114   | 114   | 114   |

**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     | 441   | 3060  | 3194  | 5066  | 11534 | 21548 | 17670 | 6640  | 1550  | 1230   | 1100   |
| 1     | 14     | 85    | 151   | 340   | 868   | 2028  | 1295  | 381   | 107   | 79     | 96     |
| 2     | 400    | 125   | 250   | 625   | 1300  | 2500  | 1000  | 300   | 90    | 45     | 45     |
| 3     | 771    | 421   | 1294  | 2147  | 5294  | 9618  | 3012  | 802   | 200   | 100    | 100    |
| 4     | 1320   | 645   | 1935  | 3300  | 7500  | 12750 | 4012  | 1001  | 250   | 125    | 125    |
| 5     | 21548  | 1088  | 3462  | 5718  | 10944 | 17776 | 5718  | 1275  | 300   | 150    | 150    |
| 6     | 35000  | 1800  | 5390  | 8760  | 16740 | 26310 | 5390  | 8760  | 2000  | 1000   | 1000   |
| 7     | 51460  | 2700  | 6738  | 10720 | 19440 | 31440 | 10720 | 19440 | 2700  | 2700   | 2700   |
| 8     | 6640   | 1800  | 4320  | 6320  | 9240  | 14700 | 1800  | 4320  | 6320  | 6320   | 6320   |
| 9     | 8140   | 2700  | 5460  | 7680  | 11520 | 17280 | 5460  | 7680  | 7680  | 7680   | 7680   |
| 10    | 9640   | 3600  | 6900  | 9240  | 13040 | 18700 | 6900  | 9240  | 9240  | 9240   | 9240   |
| 11    | 11140  | 4500  | 8100  | 10720 | 14560 | 20340 | 8100  | 10720 | 10720 | 10720  | 10720  |
| 12    | 12640  | 5400  | 9600  | 12320 | 16160 | 21960 | 9600  | 12320 | 12320 | 12320  | 12320  |
| 13    | 14140  | 6300  | 11400 | 14080 | 17920 | 23760 | 11400 | 14080 | 14080 | 14080  | 14080  |
| 14    | 15640  | 7200  | 13500 | 16720 | 20560 | 26400 | 13500 | 16720 | 16720 | 16720  | 16720  |
| 15    | 17140  | 8100  | 15600 | 19320 | 23160 | 29040 | 15600 | 19320 | 19320 | 19320  | 19320  |
| 16    | 18640  | 9000  | 17700 | 22080 | 25920 | 31800 | 17700 | 22080 | 22080 | 22080  | 22080  |
| 17    | 20140  | 9900  | 19800 | 24840 | 28720 | 34600 | 19800 | 24840 | 24840 | 24840  | 24840  |
| 18    | 21640  | 10800 | 21900 | 27600 | 31520 | 37400 | 21900 | 27600 | 27600 | 27600  | 27600  |
| 19    | 23140  | 11700 | 24000 | 30360 | 34320 | 40200 | 24000 | 30360 | 30360 | 30360  | 30360  |
| 20    | 24640  | 12600 | 26100 | 33120 | 38240 | 44100 | 26100 | 33120 | 33120 | 33120  | 33120  |

**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].
Loupekine’s Second Snark

| j = 0 | j = 1 | j = 2 | j = 3 | j = 4 | j = 5 | j = 6 | j = 7 | j = 8 | j = 9 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 21 156 | 76 945 | 126 340 | 145 064 | 105 820 | 47 030 | 10 866 | 5 763 | 1 309 |
| 1     | 21 156 | 180 076 | 417 674 | 493 056 | 387 930 | 190 386 | 71 117 | 19 114 | 3 540 |
| 2     | 124 298 | 601 016 | 981 226 | 885 441 | 475 731 | 177 183 | 45 303 | 7 638 | 759 |
| 3     | 356 730 | 1 185 620 | 1 439 086 | 931 623 | 373 263 | 97 431 | 16 253 | 1 560 | 66 |
| 4     | 675 496 | 1 635 022 | 1 475 257 | 702 666 | 200 712 | 35 160 | 3 509 | 154 |
| 5     | 957 769 | 1 724 581 | 1 154 064 | 397 725 | 78 312 | 86 862 | 485 | 6 |
| 6     | 1 180 933 | 1 489 755 | 721 889 | 175 617 | 22 828 | 1 494 | 36 |
| 7     | 1 045 640 | 1 048 408 | 371 796 | 61 744 | 5 000 | 183 | 3 |
| 8     | 863 002 | 641 304 | 160 464 | 17 346 | 80 1 | 15 |
| 9     | 631 780 | 341 344 | 36 444 | 3793 | 75 |
| 10    | 414 216 | 159 732 | 17 991 | 585 |
| 11    | 245 775 | 65 772 | 4 441 | 48 |
| 12    | 132 710 | 23 776 | 1 008 |
| 13    | 65 338 | 7 475 | 109 |
| 14    | 29 277 | 2 001 | 30 |
| 15    | 11 683 | 45 | 3 |
| 16    | 429 | 69 |
| 17    | 1 599 | 6 |
| 18    | 364 |
| 19    | 78 |
| 20    | 12 |

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 | 6 230 100 | 12 943 366 | 17 880 016 | 18 984 001 | 18 984 001 | 17 880 016 | 12 943 366 | 6 230 100 |
| 1     | 1 437 372 | 12 029 428 | 32 563 992 | 50 917 980 | 54 312 237 | 46 815 703 | 34 081 354 | 21 631 154 | 12 058 586 |
| 2     | 7 346 700 | 35 605 218 | 65 908 511 | 62 618 190 | 73 856 757 | 50 388 897 | 30 117 376 | 15 535 196 | 6 911 375 |
| 3     | 17 711 682 | 59 408 320 | 86 949 004 | 80 075 156 | 55 190 473 | 32 674 483 | 15 229 199 | 6 179 619 | 2 087 824 |
| 4     | 25 936 913 | 65 327 035 | 72 386 448 | 51 039 118 | 28 598 913 | 12 922 375 | 4 413 247 | 1 450 990 | 342 608 |
| 5     | 28 091 119 | 52 063 385 | 45 319 066 | 23 931 700 | 10 261 301 | 35 324 549 | 955 120 | 193 386 | 26 987 |
| 6     | 23 425 656 | 31 672 276 | 28 471 383 | 8 099 620 | 2 593 641 | 634 476 | 110 940 | 12 294 | 642 |
| 7     | 15 702 294 | 15 160 966 | 6 891 382 | 2 015 912 | 499 490 | 60 900 | 6 50 | 204 |
| 8     | 6 704 413 | 5 805 523 | 1 780 310 | 380 112 | 49 432 | 3 582 | 12 |
| 9     | 4 067 425 | 1 796 531 | 348 285 | 43 247 | 2 802 | 36 |
| 10    | 1 632 042 | 436 574 | 49 487 | 2 982 | 36 |
| 11    | 555 756 | 69 290 | 4 552 | 72 |
| 12    | 1 836 804 | 11 910 | 204 |
| 13    | 41 322 | 119 | 8 801 |
| 14    | 1 540 | 8 |
| 15    | 210 | 1 |
| 16    | 20 | 1 |

| j = 20 | j = 19 | j = 18 | j = 17 | j = 16 | j = 15 | j = 14 | j = 13 | j = 12 | j = 11 | j = 10 | j = 9 | j = 8 | j = 7 | j = 6 | j = 5 | j = 4 | j = 3 | j = 2 | j = 1 | j = 0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 2 549 764 | 952 258 | 303 943 | 81 073 | 17 461 | 2 089 | 323 | 19 |
| 841 881 | 213 590 | 46 208 | 7 182 | 74 | 30 |
| 124 393 | 19 960 | 2 128 | 114 |
| 7 011 | 418 | 54 |
| \(j = 38\) | \(j = 39\) | \(j = 40\) | \(j = 41\) | \(j = 42\) | \(j = 43\) | \(j = 44\) | \(j = 45\) | \(j = 46\) |
|---|---|---|---|---|---|---|---|---|
| 2 964 474 698 957 | 4 310 253 548 097 | 2 781 684 207 389 | 1 728 980 236 047 | 1 056 510 699 738 | 829 363 523 349 | 385 056 583 603 | 205 782 941 637 | 12 608 389 086 |
| 1 997 470 496 058 | 1 561 111 427 998 | 920 894 742 411 | 469 846 262 712 | 80 906 285 732 | 7 985 680 153 | 429 469 516 | 11 395 789 | 124 530 |
| 1 317 887 697 673 | 1 728 990 256 047 | 920 894 742 411 | 257 819 091 781 | 40 765 822 575 | 3 614 180 128 | 168 586 789 | 3 672 196 | 29 815 |
| 850 665 829 404 | 1 056 510 698 738 | 528 044 208 920 | 137 219 085 706 | 19 829 960 250 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 536 654 329 284 | 629 363 523 349 | 294 272 153 267 | 70 712 799 890 | 9 291 403 735 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 330 528 626 041 | 365 006 585 603 | 159 130 489 574 | 35 214 172 506 | 4 182 750 513 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 198 505 264 692 | 205 792 941 637 | 83 350 267 190 | 16 908 923 264 | 1 803 797 937 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 116 089 433 466 | 112 609 389 066 | 42 202 405 360 | 7 809 196 055 | 742 649 893 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 66 010 296 070 | 59 693 833 147 | 20 608 991 452 | 3 458 946 792 | 95 399 665 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 36 432 673 762 | 19 490 581 042 | 10 869 357 395 | 485 287 734 | 1 803 797 937 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 30 590 692 119 | 15 119 155 364 | 7 187 420 893 | 327 627 261 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 207 745 788 | 37 302 223 | 12 256 247 | 5 706 016 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 3 342 402 | 931 895 | 254 974 | 52 641 | 1 567 456 613 | 62 586 269 | 1 092 398 | 6 261 |
| 30 575 | 546 | 776 | 4 | 1 |