Computing Tutte Polynomials

GARY HAGGARD
Department of Computer Science, Bucknell University
and
DAVID J. PEARCE
School of Engineering and Computer Science, Victoria University of Wellington, NZ
and
GORDON ROYLE
School of Mathematics and Statistics, University of Western Australia

The Tutte polynomial of a graph is a 2-variable polynomial graph invariant of considerable importance in both combinatorics and statistical physics. It contains several other polynomial invariants, such as the chromatic polynomial and flow polynomial as partial evaluations, and various numerical invariants such as the number of spanning trees as complete evaluations. However, despite its ubiquity, there are no widely-available effective computational tools able to compute the Tutte polynomial of a general graph of reasonable size. In this paper we describe the implementation of an algorithm that exploits isomorphisms in the computation tree to extend the range of graphs for which it is feasible to compute their Tutte polynomials, and we demonstrate the utility of the program by finding counterexamples to a conjecture of Welsh on the location of the real flow roots of a graph.

Categories and Subject Descriptors: G.4 [**MATHEMATICAL SOFTWARE**]: Algorithm design and Analysis; G.2.2 [**GRAPH THEORY**]: Graph Algorithms; G.2.1 [**COMBINATORICS**]: Counting Problems

General Terms: Algorithms, Design

Additional Key Words and Phrases: Chromatic Polynomial, Flow Polynomial, Graph Polynomial, Graph Theory, Tutte Polynomial

Thanks go to the Isaac Newton Institute for funding the Workshop on Combinatorics and Statistical Mechanics in 2008. This enabled the authors to gather together in the same hemisphere and discuss the problem in person! Special thanks in particular go to Alan Sokal and Klas Markström for many interesting discussions. Gary Haggard was also supported by a National Science Foundation (NSF) grant (#0737730) during this time.

Authors' Addresses: Professor Gary Haggard, Dept. of Computer Science, Bucknell University, Lewisburg, PA 17837, e-mail: haggard@bucknell.edu; Dr David J. Pearce, School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand, e-mail: david.pearce@ecs.vuw.ac.nz; Professor Gordon Royle, The University of Western Australia, 35 Stirling Highway, Crawley, Perth, Western Australia 6009, e-mail: gordon@maths.uwa.edu.au.

1. INTRODUCTION

The *Tutte polynomial* of an undirected graph G (which may have loops and multiple edges) is a 2-variable polynomial $T(G;x,y)$ that encodes a significant amount of information about the graph — indeed in a strong sense it "contains" *every* graphical invariant that can be computed by deletion and contraction. In particular, the Tutte polynomial can be explicitly evaluated at particular points (x,y) to give numerical graphical invariants such as the number of spanning trees, the number of forests, the number of connected spanning subgraphs, the dimension of the bicycle space, and many more. In addition, the Tutte polynomial specialises to a variety of single-variable graphical polynomials of independent combinatorial interest, including the *chromatic polynomial*, the *flow polynomial*, and the *reliability polynomial* (see the next section for details).

The Tutte polynomial also plays an important role in the field of statistical physics where it appears as the partition function of the q-state Potts model $Z(G;q,v)$ (see [Sokal 2005]). In fact, if G is a graph on n vertices with c connected components, then

$$T(G;x,y) = (x-1)^{-c}(y-1)^{-n}Z(G;(x-1)(y-1),(y-1))$$

and so the partition function of the q-state Potts model is simply the Tutte polynomial expressed in different variables. There is a very substantial physics literature involving the calculation of the partition function for specific families of graphs, usually sequences of increasingly large subgraphs of various infinite lattices and other graphs with some sort of repetitive structure.

In knot theory, the Tutte polynomial appears in yet another guise as the Jones polynomial of an alternating knot. However, computing the Jones polynomial of a non-alternating knot, which may have some application in analysing knotted strands of DNA requires the use of a signed Tutte polynomial, which is considerably more involved. A more complete discussion of the theory and applications of Tutte polynomials can be found in [Bollobás 1998; Brylawski and Oxley 1992; Ellis-Monaghan and Merino 2009a; 2009b].

Of all the invariants associated with the Tutte polynomial, the chromatic polynomial plays a special role in both combinatorics and statistical physics. In statistical physics, the chromatic polynomial occurs as a special limiting case, namely the zero-temperature limit of the anti-ferromagnetic Potts model, while in combinatorics its relationship to graph colouring and historical status as perhaps the earliest graph polynomial has given it a unique position. As a result, particularly in the combinatorics literature, far more is known about the chromatic polynomial than about the Tutte polynomial or any of its other univariate specialisations, such as the flow polynomial, and there are still fundamental unresolved questions in these areas.

The computation of the Tutte polynomial is NP-hard — indeed, even evaluating $T(G;x,y)$ for specific values (x,y) is #P-complete, except for a few special pairs [Jaeger. et al. 1990]. However, although we cannot expect to get efficient (i.e. polynomial time) algorithms to compute the Tutte Polynomial, currently the only widely-available general purpose implementations are the naive implementations found in computer algebra systems such as Maple and Mathematica [Pemmaraju and Skiena 2003]. These naive implementations can only deal with very small graphs indeed and, therefore, practical experimentation with the Tutte polynomial has been hampered by the lack of any effective general purpose computational tool.

Earlier work of the first author [Haggard and Read 1993; Haggard and Mathies 1999] describes a tool for the chromatic polynomial of a graph, where the task is considerably simpler as the chromatic polynomial is univariate and all the graphs can be taken to be simple. Dealing with a bivariate polynomial and manipulating graphs that may include loops and multiple edges introduces a range of different issues that must be resolved. In [Sekine et al. 1995], an algorithm is described that will compute Tutte polynomials of graphs with no more than 14 vertices. The algorithm performs a breadth-first search of the computation tree, whilst identifying and pruning equivalent graphs at each level. To determine equivalence, they employ a notion referred to as *2-isomorphism.* More specifically, two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ are 2-isomorphic if there is a bijection $\phi : E_1 \longrightarrow E_2$, such that $\phi(T)$ is a spanning tree of G_2 iff T is a spanning tree of G_1. Sekine et al. use a fixed ordering of the edges and delete/contract the edges in this order along every branch of the tree; then, they check for pairs of graphs for which the identity mapping is a 2-isomorphism.

The algorithm given in [Read 1987] for computing chromatic polynomials was extended in [Royle 1988] to compute Tutte polynomials of moderate sized graphs, but is not effective much beyond 14 vertices. By comparison, our algorithm can process graphs with 14 vertices in a matter of seconds (as shown in Section 6). In [Björklund et al. 2008a] an algorithm is given which computes the Tutte polynomial using $2^n n^{O(1)}$ time and space; a variation is also given which runs in time $3^n n^{O(1)}$, but requires only polynomial space. Some experimental data is reported for an implementation of the former, but the memory constraints limited its practicality (see [Björklund et al. 2008b] App. D).

In this paper, we describe the implementation of an effective algorithm for computing Tutte polynomials. The algorithm is based on the idea of caching intermediate graphs and their Tutte polynomials and using graph isomorphism to avoid unnecessary recomputation of subtrees of the computation. We present some experimental results using this algorithm and a discussion of some factors affecting its performance. In addition, as an example of its practical use, we present counterexamples to a conjecture of Welsh on the location of the roots of the flow polynomial of a graph, by finding for the first time graphs with real flow roots larger than 4. Finally, our implementation also supports the practical computation of chromatic and flow polynomials, based on the same techniques presented in this paper, and the code can be obtained from `http://ecs.victoria.ac.nz/~djp/tutte`.

2. PRELIMINARIES

Let $G = (V, E)$ be an *undirected multi-graph*; that is, V is a set of vertices and E is a multi-set of unordered pairs (v, w) with $v, w \in V$. An edge (v, v) is called a *loop*. If an edge (u, v) occurs more than once in E it is called a *multi-edge*. The *underlying graph* of G is obtained by removing any duplicate entries in E.

The Tutte polynomial can be defined in terms of two operations on graphs. These are: deleting an edge, denoted by $G - e$; and contracting an edge, denoted by G/e (see Figure 1).

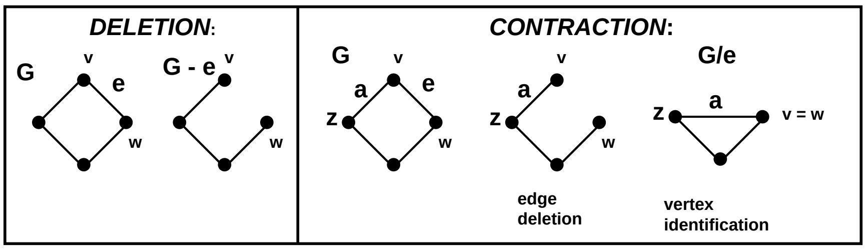


Fig. 1. Deletion and Contraction of an Edge

Definition 1. *The* Tutte polynomial *of a graph* $G = (V, E)$ *is a two-variable polynomial defined as follows:*

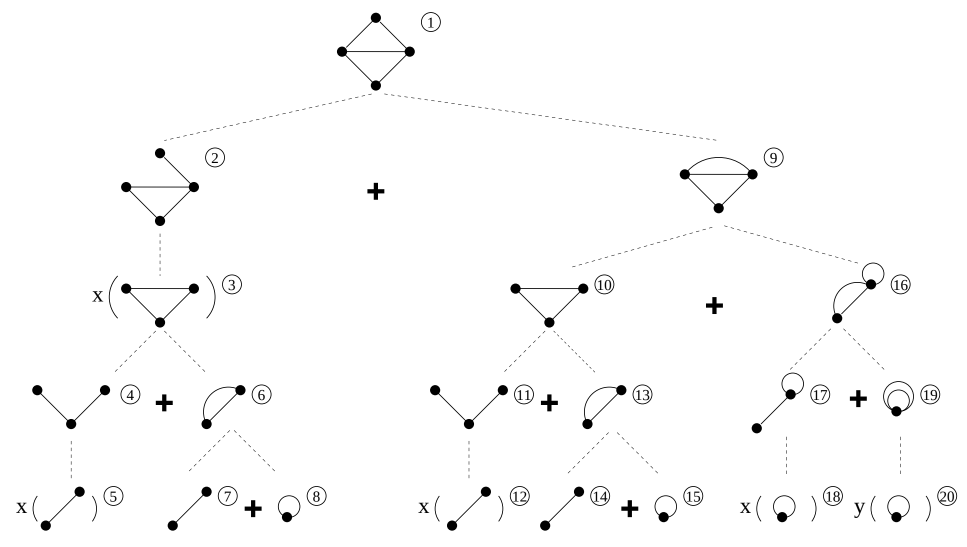
$$T(G; x, y) = \begin{cases} 1 & E(G) = \emptyset \\ xT(G/e; x, y) & e \in E \textit{ and } e \textit{ is a bridge} \\ yT(G - e; x, y) & e \in E \textit{ and } e \textit{ is a loop} \\ T(G - e; x, y) + T(G/e; x, y) & \textit{otherwise} \end{cases}$$

This definition of the Tutte polynomial, which Tutte showed is independent of the edge chosen [Tutte 1954], outlines a simple recursive procedure for computing it. Thus, we are free to apply its rules in whatever order we wish, and to choose any edge to operate on at each stage. Figures 2 and 3 illustrate this recursive procedure applied to a simple graph to give the final polynomial. It should be clear from these figures that the structure of the computation corresponds to a tree.

The order in which the rules of Definition 1 are applied significantly affects the size of the computation tree. An “efficient” order can reduce work in a number of ways. For example, there are two situations where an edge is associated with a factor directly: if the edge is a loop, the factor is y; likewise, if the edge is a bridge, the factor is x. Eliminating such edges as soon as possible and storing the factor for later incorporation into the answer reduces work by lowering the cost of operations (e.g. contracting, connectedness testing, etc.) on graphs in the subtrees below. In Figure 2, for example, the loop present in G_{16} is not reduced immediately and, instead, is propagated to the bottom of the computation tree; removing it immediately reduces, amongst other things, the cost of duplicating the graph when the branch forks further down.

Within a single computation tree, it often arises that a graph occurs more than once (including those isomorphic to it). Thus, recomputing its Tutte polynomial from scratch each time is wasteful and should be avoided when possible. For example, the triangle occurs twice in Figure 2, both as G_3 and G_{10}. Thus, we can simplify the tree by simply reusing the result from $T(G_3)$ in place of $T(G_{10})$. This optimisation has a significant effect on the performance of our algorithm in practice (as shown in Section 6).

The choice of edge for a delete/contract operation can also greatly affect the size of the computation tree. In particular, it affects the likelihood of reaching a graph isomorphic to one already seen. For example, selecting $(4, 2)$ when evaluating $T(G_9)$ in Figure 3 yields the triangle (as shown); choosing any of the other edges, however, does not. In [Pearce et al. 2009], we initiate the exploration of different



$$\left(x\left(x^2 + (x + y)\right) + \left(x^2 + (x + y)\right) + \left(xy + y^2\right)\right)$$

Fig. 2. Illustrating the application of Definition 1 to a small graph. Observe that vertices are not drawn once they become isolated, as they play no further role. Also, each graph is given a unique number to aid identification.

$$\begin{aligned}
T(G_1 = \{(1,2),(2,3),(3,4),(4,1),(2,4)\}) &= T(G_2 = G_1 - (4,1)) + T(G_9 = G_1/(4,1)) \\
T(G_2 = \{(1,2),(2,3),(3,4),(2,4)\}) &= x \cdot T(G_3 = G_2 - (1,2)) \\
T(G_3 = \{(2,3),(3,4),(2,4)\}) &= T(G_4 = G_3 - (2,4)) + T(G_6 = G_3/(2,4)) \\
T(G_4 = \{(2,3),(3,4)\}) &= x \cdot T(G_5 = G_4 - (3,4)) \\
T(G_5 = \{(2,3)\}) &= x \cdot T(G_5 - (2,3) = \emptyset) = x \cdot 1 \\
T(G_6 = \{(2,3),(3,2)\}) &= T(G_7 = G_6 - (2,3)) + T(G_8 = G_6/(2,3)) \\
T(G_7 = \{(2,3)\}) &= x \\
T(G_8 = \{(2,2)\}) &= y \\
T(G_9 = \{(4,2),(2,3),(3,4),(2,4)\}) &= T(G_{10} = G_9 - (4,2)) + T(G_{16} = G_9/(4,2)) \\
T(G_{10} = \{(2,3),(3,4),(2,4)\}) &= T(G_{11} = G_{10} - (4,2)) + T(G_{13} = G_9/(4,2)) \\
T(G_{11} = \{(2,3),(3,4)\}) &= x \cdot T(G_{12} = G_{11} - (3,4)) \\
T(G_{12} = \{(2,3)\}) &= x \cdot T(G_{12} - (2,3) = \emptyset) = x \cdot 1 \\
T(G_{13} = \{(2,3),(3,2)\}) &= T(G_7 = G_{14} - (2,3)) + T(G_{15} = G_6/(2,3)) \\
T(G_{14} = \{(2,3)\}) &= x \\
T(G_{15} = \{(2,2)\}) &= y \\
T(G_{16} = \{(2,3),(3,2),(2,2)\}) &= T(G_{17} = G_{16} - (2,3)) + T(G_{19} = G_9/(2,3)) \\
T(G_{17} = \{(3,2),(2,2)\}) &= x \cdot T(G_{18} = G_{16} - (2,3)) \\
T(G_{18} = \{(2,2)\}) &= y \\
T(G_{19} = \{(2,2),(2,2)\}) &= y \cdot T(G_{20} = G_{19} - (2,2)) \\
T(G_{20} = \{(2,2))\}) &= y \cdot T(G_{19} - (2,2) = \emptyset) = y \cdot 1
\end{aligned}$$

Fig. 3. Illustrating an algebraic proof of the computation illustrated in Figure 2. Observe that the graph numbers given (e.g. G_1) align with those given in Figure 2.

edge selection heuristics, although much remains to be done here.

Finally, an efficient algorithm for computing Tutte polynomials can be used immediately to compute chromatic, flow, and reliability polynomials. For example, for the chromatic polynomial $P(G;\lambda)$ and flow polynomial $F(G;\lambda)$ of a graph with n vertices, e edges and c connected components are derived as follows from the Tutte polynomial:

$$\begin{aligned} P(G;\lambda) &= (-1)^{n-c}\lambda^c \cdot T(G;(1-\lambda),0) \\ F(G;\lambda) &= (-1)^{e-n+c} \cdot T(G;0,(1-\lambda)) \end{aligned}$$

If G is a planar graph with planar dual G^*, then $T(G^*;x,y) = T(G;y,x)$ and so, up to factors of λ, the flow polynomial of a planar graph is the chromatic polynomial of its planar dual.

3. ROOTS OF FLOW POLYNOMIALS - WELSH'S CONJECTURE

Our primary motivation for developing an effective algorithm was to extend the range for which computational exploration of questions relating to Tutte polynomials is feasible, as there are a number of long-standing open questions for which the computational evidence is extremely limited.

Several of these problems relate to the location of the *roots* of the various single-variable specialisations of the Tutte polynomial mentioned earlier. In particular, the roots of the chromatic polynomial, or *chromatic roots* have been extensively studied while much less is known about the roots of the flow polynomial. One fundamental question about which very little is known is whether there is an upper bound on the value of real flow roots. As there are graphs (such as the Petersen graph) with no nowhere-zero 4-flows, the strongest possible result would be that there are no real flow roots larger than 4.

CONJECTURE 1 DOMINIC WELSH. *If G is a bridgeless graph with flow polynomial $F(G;\lambda)$, then $F(G;r) > 0$ for all $r \in (4,\infty)$.*

This well-known (but unpublished) conjecture is essentially a planar dual version of the famous Birkhoff-Lewis conjecture that planar graphs have no *chromatic* roots in $[4,\infty)$ which has been proved for $r = 4$ (the four-colour theorem) and for $[5,\infty)$.

In prior study of chromatic roots, cubic graphs of high girth have played an important role as they seem to exhibit qualitatively extremal behaviour (this is a deliberately imprecise statement) and, for this reason, they are a natural class to examine for other questions related to Tutte polynomials. In this vein, we computed the Tutte polynomials of cubic graphs of girth at least 7 on 24–32 vertices with the intention of testing a variety of conjectures against this data set.

An immediate positive outcome of this experiment was the discovery of a number of counterexamples to Welsh's conjecture. A specific example is the generalised Petersen graph $P(16,6)$ which is a 32-vertex cubic graph of girth 7 shown in Figure 4 with flow polynomial $(\lambda-1)(\lambda-2)(\lambda-3)Q(G;\lambda)$ where

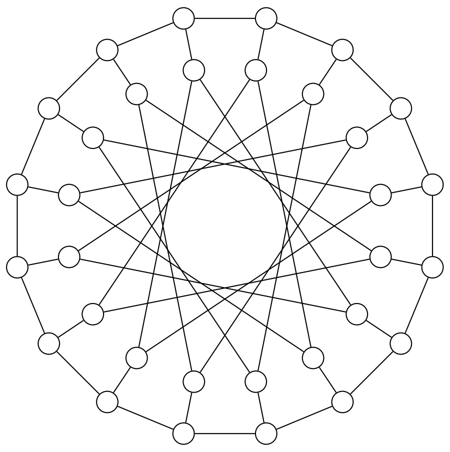


Fig. 4. The generalised Petersen graph $P(16, 6)$

$$\begin{aligned}
Q(G; \lambda) &= \lambda^{14} - 42\,\lambda^{13} + 833\,\lambda^{12} - 10358\,\lambda^{11} + 90393\,\lambda^{10} - 587074\,\lambda^{9} \\
&+ 2934917\,\lambda^{8} - 11515364\,\lambda^{7} + 35798907\,\lambda^{6} - 88275860\,\lambda^{5} \\
&+ 171273551\,\lambda^{4} - 256034548\,\lambda^{3} + 282089291\,\lambda^{2} \\
&- 207662412\,\lambda + 77876944.
\end{aligned}$$

This has real roots at two values $\lambda_1 \approx 4.0252205$ and $\lambda_2 \approx 4.2331455$ thereby demonstrating that 4 is not the upper limit for flow roots. There are a variety of other examples on 28 and 36 vertices, but the smaller ones are more difficult to describe. The common features of the examples found are that the flow polynomial is a polynomial of reasonably high *odd* degree that has a negative derivative at $\lambda = 4$ and is strongly positive at $\lambda = 5$. The graphs on 30 and 34 vertices that were examined have flow polynomials of even degree with positive derivative at $\lambda = 4$ and values that just keep increasing as λ increases.

Given that 4 is not an upper limit for flow roots, what would be an appropriate replacement for Welsh's conjecture? The nature of these examples suggests that they will not give flow roots above 5, and yet there seems to be no strong reason to choose any value strictly between 4 and 5. Therefore, we propose the following conjecture:

CONJECTURE 2. *If G is a bridgeless graph with flow polynomial $F(G; \lambda)$, then $F(G; r) > 0$ for all $r \in [5, \infty)$.*

The case $r = 5$ is simply Tutte's 5-flow conjecture and so the truth of this conjecture (and the Birkhoff-Lewis conjecture) would give an appealing parallel between the flow roots of general graphs and chromatic roots of planar graphs.

Finally, we have sanity checked the Tutte polynomial computed for $P(16, 6)$ by evaluating it at several known points (see Section 5.2).

4. ALGORITHMIC OBSERVATIONS

In this section, we begin by detailing several well-known theorems about the Tutte polynomial and explain how these can be exploited to improve computational performance.

4.1 Known Reductions

There are numerous well-known properties of the Tutte polynomial that can be exploited to prune the computation tree and, hence, improve performance. The first of these exploits the fact that the Tutte polynomial is multiplicative over the blocks (i.e., maximal 2-connected components) of a graph and that these blocks can be determined in linear time [Tutte 1954].

THEOREM 1. *Let $G = (V, E)$ be a graph with m blocks G_1, G_2, ..., G_m. Then $T(G; x, y) = \prod_{i=1}^{m} T(G_i; x, y)$.* □

At present, our system uses a standard algorithm for identifying biconnected components [Tarjan 1972], extracting the non-trivial biconnected components (that is, those with more than 2 vertices). The trivial biconnected components are edges and multi-edges whose underlying graph is a forest, and these are processed in a single step using the following lemma, which is immediate from the definitions.

LEMMA 1. *Let $G = (V, E)$ be a multi-graph whose underlying graph is a forest with s edges. Denote the multiplicity of each distinct edge in the graph by $d_1, \ldots, d_s$. Then,*

$$T(G; x, y) = \prod_{i=1}^{s} (x + y + y^2 + \ldots + y^{d_i - 1})$$

□

An *ear* in a graph is a path $v_1 \sim v_2 \sim \cdots \sim v_n \sim v_{n+1}$ where $d(v_1) > 2$, $d(v_{n+1}) > 2$ and $d(v_2) = d(v_3) = \cdots = d(v_n) = 2$. A cycle is viewed as a "special" ear where $v_1 = v_{n+1}$ and the restriction on the degree of this vertex is lifted. If a graph contains a multi-edge or an ear, then all the edges involved can be removed in a single operation. We denote an edge of multiplicity p by e^p and an ear with s edges by E_s. Deletion of a multi-edge or ear is defined naturally as meaning the deletion of all the edges. Contraction of a multi-edge means to delete all the edges and identify the end-vertices, while contraction of an ear means to delete all the edges and identify v_1 and v_{n+1}.

THEOREM 2. *Suppose that G is a biconnected graph that is either equal to a multi-edge e^p of multiplicity p or properly contains a multi-edge e^p. Then*

$$T(G; x, y) = \begin{cases} (x + y + \cdots + y^{p-1}), & \text{if } G = e^p \\ (1 + y + \cdots + y^{p-1}) T(G/e^p; x, y) + T(G - e^p; x, y), & \text{otherwise} \end{cases}$$

□

Ears are dual to multiple edges and so we have the dual result:

THEOREM 3. *Suppose that G is a biconnected graph that is either equal to an ear E_s (which is necessarily a cycle of length s) or properly contains an ear E_s. Then*

$$T(G;x,y) = \begin{cases} (y + x + \cdots + x^{s-1}), & \textit{if } G = E_s \\ (1 + x + \cdots + x^{s-1})T(G - E_s;x,y) + T(G/E_s;x,y), & \textit{otherwise} \end{cases}$$

□

In matroid terminology, these results say that an entire parallel class or series class can be processed at once.

These two results follow immediately from the rules for deletion/contraction and Figure 5 visually outlines the proof of Theorem 3. The value of these two theorems is that we can exploit them to further prune the computation tree; we find that they offer significant performance improvements in practice.

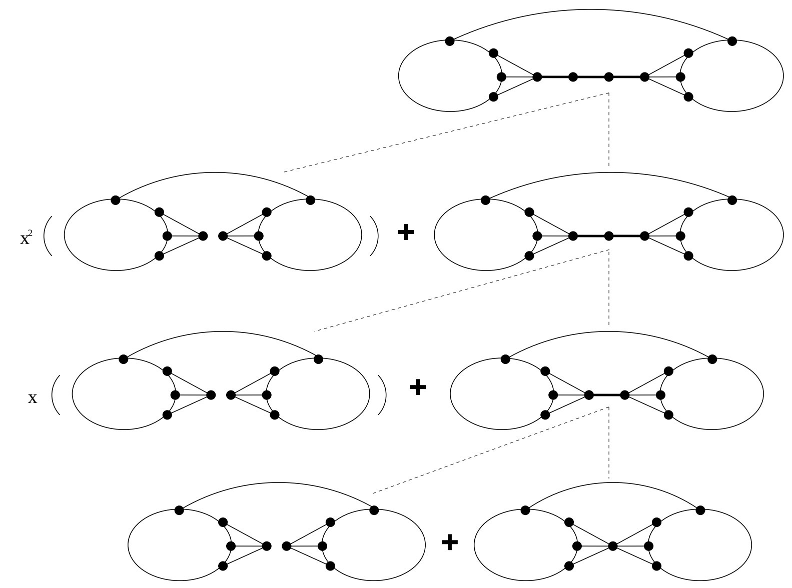


Fig. 5. Reduction of an ear. In the first delete/contract, we select an edge on the ear for removal; on the left branch, this leaves two single-edge biconnected components which immediately yield a factor of x^2.

There are more complex structures that can, in principle, be processed in a single step, such as when there is an ear in the underlying simple graph, but this ear contains multi-edges in the graph itself.

THEOREM 4. *Let $G = (V,E)$ be a multi-graph whose underlying graph is an n-cycle. Denote the multiplicity of each distinct edge in the cycle by $d_1, \ldots, d_n$, and let $y^{a\ldots b}$ be short-hand for $y^a + y^{a+1} \ldots + y^{b-1} + y^b$, Then,*

$$T(G;x,y) = \sum_{i=1}^{n} \left(\prod_{j=i+1}^{n} (x + y^{1\ldots d_j - 1}) \prod_{k=1}^{i-1} (y^{0\ldots d_k - 1}) \right) + (x + y^{d_n + d_{n-1} - 1}) \prod_{i=1}^{n-2} (y^{0\ldots d_i - 1})$$

PROOF. The proof is by induction. The first step is to use the Tutte recursion to reduce G into two smaller graphs. For the delete graph, we apply Lemma 1. For the contract graph we observe it is simply a $(k-1)$-multicycle to which the inductive hypothesis can be applied. Figure 6 outlines the proof for the special case $n = 6$. □

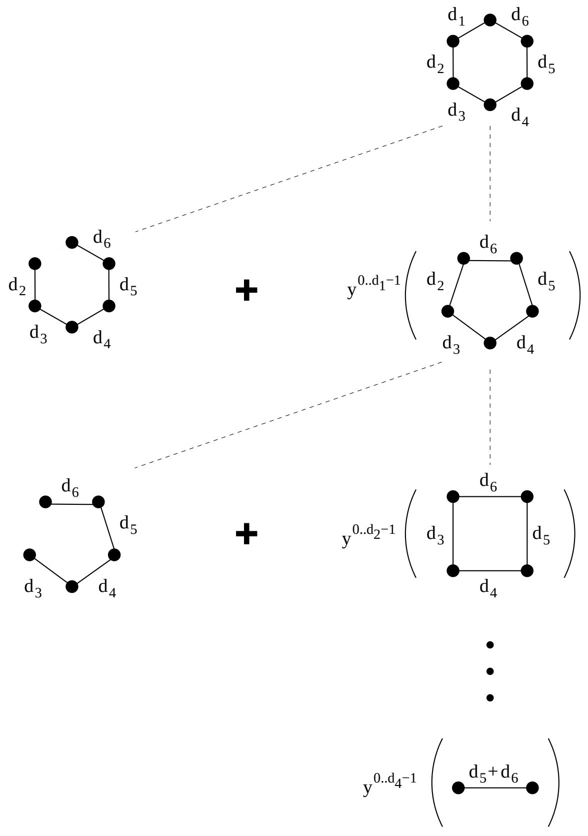


Fig. 6. An outline of the proof for reducing multi-cycles

5. ALGORITHM OVERVIEW

We now provide an overview of our algorithm to illustrate the main choices we have made. We also discuss some of the more practical, but nonetheless important issues which we faced when implementing our algorithm for computing Tutte polynomials.

As the algorithm operates, it essentially traverses the computation tree in a depth-first fashion (although the whole tree is never held in memory at once). That is, when a delete/contract operation is performed on G, the algorithm recursively evaluates $T(G-e; x, y)$ until its polynomial is determined, before evaluating $T(G/e; x, y)$. Other traversal strategies are possible and could offer some benefit, although we have yet to explore this. At each node in the computation tree, the algorithm maintains and/or generates a variety of information on the graph being processed — such as whether it is connected or biconnected — to help identify opportunities for pruning the tree. In particular, the following approaches are employed:

i) **Reductions.** Known properties of Tutte polynomials (see Section 4) are used to immediately reduce either the whole graph, or a subgraph, to a polynomial. This simplifies the computation tree and can speed up the various operations performed on graphs in the subtree (of course, if the whole graph is reduced there is no subtree!). In our system, trees, loops, cycles and multi-edges, multi-cycles, and multi-ears can be reduced immediately.

ii) **Biconnectedness.** Following Theorem 1, we break graphs which are not biconnected into their non-trivial biconnected components and the residual forest. The polynomials for the biconnected components are then computed independently, which is helpful as their computation trees may be significantly smaller.

iii) **Cache.** Computed polynomials for graphs encountered during the computation are stored in a cache. Thus, if a graph isomorphic to one already resolved is encountered, we simply recall its polynomial from the cache. This optimisation typically has a significant effect, since the whole subtree of the computation below the isomorph is pruned. To determine graph isomorphism, we employ McKay's *nauty* program [McKay 1990]. The size of the cache employed and the replacement strategy used when the cache fills require further study as both can have significant effects.

iv) **Edge Selection.** As indicated already, the choice of edge for deletion and contraction affects the likelihood of reaching a graph isomorphic to one already seen (see Section 2). Furthermore, it affects the chance of exposing structures (e.g. cycles and trees) which can be immediately reduced. We currently employ a simple edge selection strategy, that just uses an arbitrary ordering of the vertices. Starting from the first vertex in the ordering, it repeatedly selects edges from that vertex until none remain, before moving on to the next vertex in the ordering.

The coefficients computed for the Tutte polynomial of even a small graph are large and can easily go beyond the size of a machine's 32-bit or 64-bit word size. To address this, we have implemented a simple library for arbitrary sized integers.

5.1 Graph Isomorphism

To implement the cache for polynomials of graphs at nodes of the computation tree, we employ a simple hash map. This is keyed upon a canonical labeling of the graph obtained using *nauty* [McKay 1990]. Since *nauty* accepts only simple graphs, we transform multigraphs into simple graphs by inserting additional vertices as necessary. We refer to such graphs as being "constructed". To avoid a constructed graph from clashing with a normal simple graph having the same number of vertices and edges, we exploit the fact that *nauty* allows vertices to be coloured and will reflect the colour class of a vertex in the canonical form. Thus, vertices added to represent multi-edges are coloured differently from normal vertices. An interesting issue here is that, at each node in the computation tree, we must *recompute the canonical labelling from scratch* as the graph, by definition, is different from its parent. While we have not explored the ramifications of this as yet, there is potential for exploiting an incremental graph isomorphism algorithm which could more efficiently determine the canonical labelling of a graph given that of its parent.

An important problem we face is what to do when the cache fills up, which happens frequently for large graphs, even when large amounts (e.g. > 2GB) of memory are available. To resolve this, we employ techniques from garbage collection: items in the cache are displaced and, to avoid memory fragmentation, those left are compacted into a contiguous block (this is similar to mark-and-sweep garbage collection). To determine which graphs to displace, we employ a simple policy based on counting the number of times a graph in the cache has been "hit". When the cache is full, graphs with a low hit count are displaced before those with higher counts. A related problem is how much of the cache to displace. Clearly, displacing less means the cache will fill more frequently and, on average, contain more old items. Of course, the more items there are in the cache, the greater the potential for collisions when searching for an isomorph. In contrast, displacing more of the cache each time means that many graphs which may turn out to be useful later on will not survive. In our implementation, the default policy is to displace 30% of the cache in one go when it becomes full.

5.2 Correctness

The output of any complex computer program should be treated with caution, if not outright suspicion, and be carefully examined for internal consistency and cross-referenced against known values. Aside from manual testing on small graphs, we employ a number of "sanity checks" to increase our confidence in its correctness. The easiest check is to compute $T(G; 2, 2)$, which should give $2^{|E(G)|}$, and compare this with a direct computation of $2^{|E(G)|}$. Another check is to compute $T(G; 1, 1)$, which gives the number of spanning trees in the graph. Then, we can check that these evaluations are constant for a given graph, regardless of what parameters are chosen for a particular run of our algorithm (e.g. cache size, which reductions are applied, edge selection strategy, vertex ordering, etc).

As an illustration, we have sanity-checked the generalised Petersen graph $P(16, 6)$ discussed in Section 3 as follows:

—$T(2, 2) = 2^{48}$ as required.

—$T(1, 1) = 115184214544$ is the number of spanning trees of $P(16, 6)$ as determined by the matrix-tree theorem.

—$(-1)\ T(1 - x, 0)$ equals the chromatic polynomial of $P(16, 6)$ which was independently verified by two separate programs.

—$T(-1, -1) = -2$ equals the expected value $(-2)^d$ where $d = 1$ is the dimension of the bicycle space of $P(16, 6)$ [Rosenstiehl and Read 1978], which can be computed by elementary linear algebra.

—$T(0, -3) = -480$ is the number of 4-flows of $P(16, 6)$ which is equal to the number of edge-3-colourings of $P(16, 6)$ which can easily be verified by a direct search.

6. EXPERIMENTAL RESULTS

In this section, we report on some experimental results obtained using our system. The objective here is to give the reader an indication of the performance that can be expected in practice. We consider random connected graphs, random planar graphs and random regular graphs. The machine used for these experiments was an Intel Pentium IV 3GHz with 1GB of memory, running NetBSD v4.99.9.

6.1 Experimental Procedure

To generate *random connected graphs*, we employed the tool `genrang` (supplied with *nauty*) to construct random graphs with a given number of edges; from these, we selected connected graphs until there were 100 for each value of $|E|$ or $|V|$ (depending upon experiment). The `genrang` tool constructs a random graph by generating a random edge, adding it to the graph (if not already present), and then repeating this until enough edges have been added. We also used `genrang` to generate random simple regular graphs — this essentially works by generating a random regular multigraph and then throwing it out if it contains loops or multiple edges. Generating *random planar graphs* required a different approach since the number of randomly generated graphs that are planar is extremely small. Therefore, we employed a Markov-chain approach: here, a pair of vertices were selected at random; if the corresponding edge was not already present and the graph would remain planar, then it was added; otherwise, it was removed. This procedure was repeated for $3n^2$ steps (which, according to [Denise et al. 1996], is well beyond the equilibrium point).

6.2 Experimental Results

Figure 7 presents the data from our experiments on random connected graphs. Data is provided for timings with and without the cache enabled. From the figure, it is immediately obvious that the cache has a critical effect on the performance of the algorithm. As expected, performance deteriorates as graph density increases; however, the algorithm appears to perform surprisingly well on very dense graphs. This stems from the increased regularity present in dense graphs which gives rise to a greater number of isomorphic hits in the cache.

Figure 8 reports the data from our experiments on random planar, 3-regular and 4-regular graphs. From the graphs, it is clear that computing the Tutte polynomial for large graphs quickly becomes intractable. Nevertheless, using the isomorphism cache extends the size of graphs which can be computed. This is of significant value in practice, since it extends the range of graphs over which users of the tool can, for example, test a conjecture they are considering.

Finally, Figure 9 illustrates the performance of our algorithm with and without caching enabled on complete graphs. This is interesting as, compared with other approaches for computing Tutte polynomials, our algorithm is extremely efficient. For example, using Maple (and similarly for Mathematica), one cannot compute the Tutte polynomial of K_{10} in any feasible amount of time. This is perhaps not surprising, however, as these implementations make no effort to prune the computation tree [Pemmaraju and Skiena 2003].

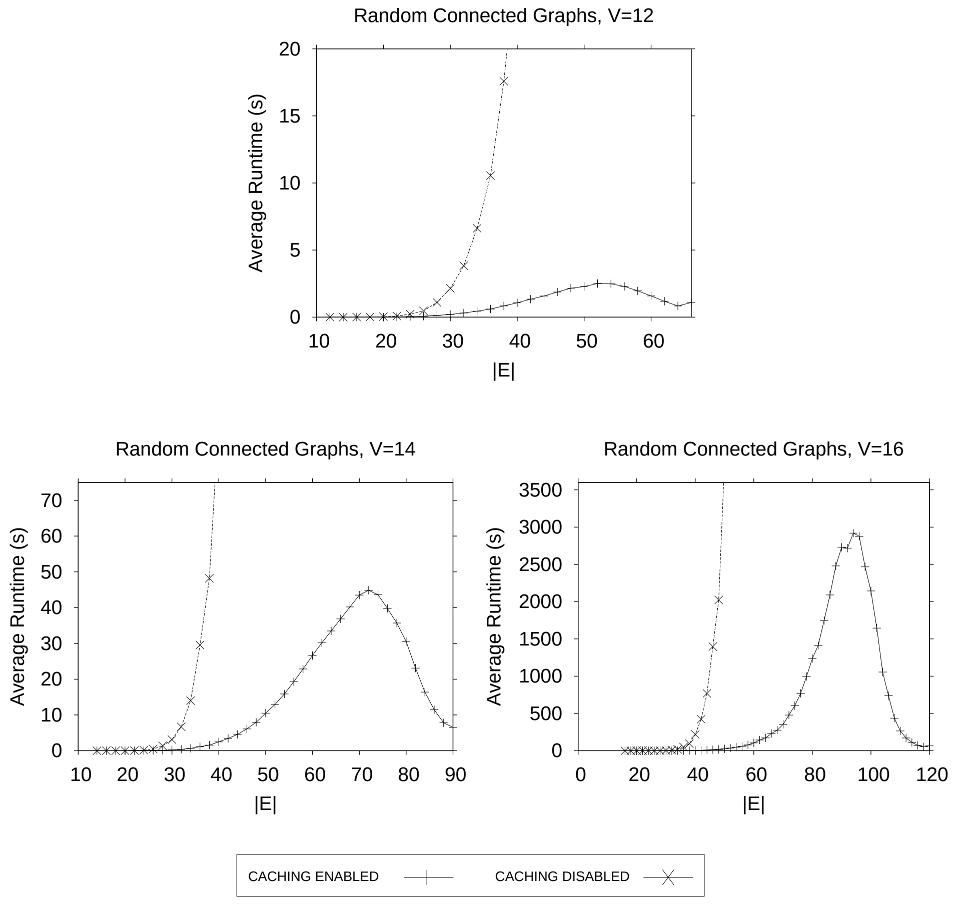


Fig. 7. Random Connected Graphs at $|V| = 12$, $|V| = 14$, $|V| = 16$ and varying $|E|$, where each data-point is averaged over 100 graphs. Data illustrating the time taken with and without the cache are shown. For each experiment, the cache size was fixed at 768MB.

7. CONCLUSION

Algorithms for computing Tutte polynomials have been, in general, rather simplistic. We have demonstrated a number of techniques which can greatly reduce the size of the computation tree. This, in turn, leads to an algorithm which can tackle significantly larger graphs than previously possible. While this task may seem futile (since the problem is #P-hard), it is important to remember that, in practice, the applications of this tool (e.g. for classifying DNA knots) have finite requirements; thus, we are moving towards a system which can handle sufficiently large graphs to be of use to practitioners.

A number of interesting questions remain for further research. One is the order in which to select edges for the delete/contract operations; can we identify heuristics which lead to good selection orders? Certainly, in our initial investigations (see [Pearce et al. 2009]), we have encountered heuristics which seem to perform

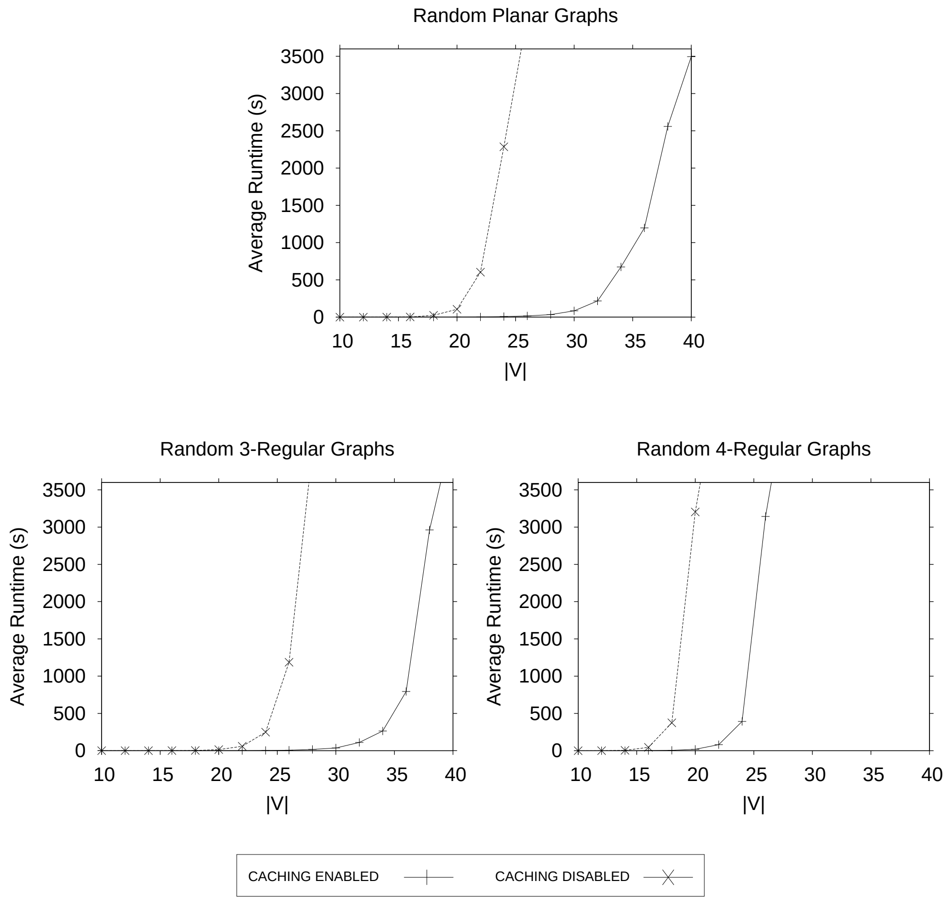


Fig. 8. Random Planar, 3-Regular and 4-Regular Graphs with varying $|V|$, where each data-point is averaged over 100 simple graphs. Data illustrating the time taken with and without the cache are shown. For each experiment, the cache size was fixed at 768MB.

consistently better than others, such as random selection. Also, at each node in the computation tree, we compute a canonical labelling of the corresponding graph so it can be stored in the cache (this enables later identification of isomorphs). But, should we do this at every node? For example, could we maintain the canonical labelling incrementally? Likewise, we currently compute the biconnected components at each node. Again, it's possible that further gains could be made by using an incremental algorithm instead (e.g. [Westbrook and Tarjan 1992; Rauch 1994]). Furthermore, our algorithm makes no particular effort to select edges whose deletion helps to create separating vertices; instead, it simply exploits them when, by chance, they occur. There is also a similar, though more complicated, algorithm for detecting separating pairs of vertices and decomposing the graph into *triconnected* components [Hopcroft and Tarjan 1973; Gutwenger and Mutzel 2001], though we have not yet experimented with this.

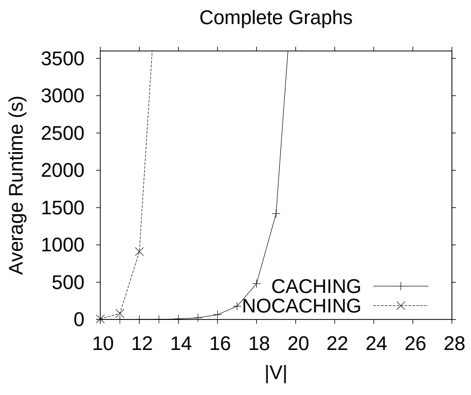


Fig. 9. Illustrating performance on complete graphs. Data illustrating the time taken with and without the cache are shown. For each experiment, the cache size was fixed at 768MB.

Nevertheless, even with this list of interesting unanswered questions, the implementation described gives researchers an effective way to experiment with Tutte polynomials, both to answer questions, and test conjectures for a wide range of sizes of graphs. The complete implementation of our algorithm can be obtained from `http://ecs.victoria.ac.nz/~djp/tutte`. This also supports practical computation of chromatic and flow polynomials, based on the same techniques presented in this paper.

Note Added in Proof

One of the earliest papers on chromatic polynomials [Hall et al. 1965] reported on the astonishing calculation of the chromatic polynomial of the truncated icosahedron (At that time, colourings were always viewed as *face*-colourings of planar maps, so in modern terminology this is actually the chromatic polynomial of the 32-vertex, 90-edge planar dual of the truncated icosahedron.) As a modern counterpart to this paper, we have now computed the full Tutte polynomial of this graph using a network of 150 computers. The polynomial can be obtained from `http://ecs.victoria.ac.nz/~djp/tutte` and further details will appear in a subsequent paper.

REFERENCES

Björklund, A., Husfeldt, T., Kaski, P., and Koivistor, M. 2008a. Computing the Tutte Polynomial in vertex-exponential time. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science (FOCS)*. 677–686.

Björklund, A., Husfeldt, T., Kaski, P., and Koivistor, M. 2008b. Computing the Tutte Polynomial in vertex-exponential time. Technical report, `arxiv.org/PS_cache/arxiv/pdf/0711/0711.2585v4.pdf`.

Bollobás, B. 1998. *Modern Graph Theory.* Number 184 in Graduate Texts in Mathematics. Springer, New York NY.

BRYLAWSKI, T. AND OXLEY, J. 1992. The Tutte Polynomial and its applications. In *Matroid applications*. Encyclopedia Math. Appl., vol. 40. Cambridge Univ. Press, Cambridge, 123–225.

DENISE, A., VASCONCELLOS, M., AND WELSH, D. J. A. 1996. The random planar graph. *Congressus Numerantium 113*, 61–79.

ELLIS-MONAGHAN, J. AND MERINO, C. 2009a. *Graph polynomials and their applications I: the Tutte Polynomial*. Invited chapter for Structural Analysis of Complex Networks. `http://arxiv.org/abs/0803.3079`.

ELLIS-MONAGHAN, J. AND MERINO, C. 2009b. *Graph polynomials and their applications II: interrelations and interpretations*. Invited chapter for Structural Analysis of Complex Networks. `http://arxiv.org/abs/0806.4699`.

GUTWENGER, C. AND MUTZEL, P. 2001. A linear time implementation of SPQR-trees. In *Proceedings of the 8th International Symposium on Graph Drawing (GD)*. Springer-Verlag, London, UK, 77–90.

HAGGARD, G. AND MATHIES, T. 1999. The computation of Chromatic Polynomials. *Discrete Math 199*, 227–231.

HAGGARD, G. AND READ, R. 1993. Chromatic Polynomials of large graphs. II. isomorphism abstract data type for small graphs. *J. Math. Comput 3*, 35–43.

HALL, D. W., SIRY, J. W., AND VANDERSLICE, B. R. 1965. The Chromatic Polynomial of the Truncated Icosahedron. *Proceedings of the American Mathematical Society 16,* 4, 620–628.

HOPCROFT, J. E. AND TARJAN, R. E. 1973. Dividing a graph into triconnected components. *SIAM Journal on Computing 2,* 3, 135–158.

JAEGER., F., VERTIGAN, D., AND WELSH, D. 1990. On the computational complexity of the Jones and Tutte Polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society 108,* 1, 35–53.

MCKAY, B. 1990. Nauty users guide (version 1.5). Technical report, Dept. Comp. Sci., Australian National University.

PEARCE, D. J., HAGGARD, G., AND ROYLE, G. 2009. Edge-selection heuristics for computing Tutte Polynomials. In *Proceedings of the Computing: The Australasian Theory Symposium (CATS)*. 153–162.

PEMMARAJU, S. AND SKIENA, S. 2003. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, Cambridge.

RAUCH, M. 1994. Improved data structures for fully dynamic biconnectivity. In *Proceedings of the ACM symposium on Theory of computing*. ACM, 686–695.

READ, R. 1987. An improved method for computing Chromatic Polynomials of sparse graphs. Tech. Rep. CORR 87-20, Dept. Comb. & Opt., University of Waterloo.

ROSENSTIEHL, P. AND READ, R. C. 1978. On the principal edge tripartition of a graph. *Ann. Discrete Math. 3*, 195–226. Advances in graph theory (Cambridge Combinatorial Conf., Trinity College, Cambridge, 1977).

ROYLE, G. F. 1988. Computing the Tutte Polynomial of sparse graphs. Tech. Rep. CORR 88-35, Dept. Comb. & Opt., University of Waterloo.

SEKINE, K., IMAI, H., AND TANI, S. 1995. Computing the Tutte Polynomial of a graph of moderate size. *Lecture Notes in Computer Science 1004*, 224–233.

SOKAL, A. D. 2005. The multivariate Tutte Polynomial (alias Potts model) for graphs and matroids. In *Surveys in combinatorics 2005*. London Math. Soc. Lecture Note Ser., vol. 327. Cambridge Univ. Press, Cambridge, 173–226.

TARJAN, R. E. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing 1,* 2, 146–160.

TUTTE, W. 1954. A contribution to the theory of Chromatic Polynomials. *Canadian Journal of Mathematics 6*, 80–81.

WESTBROOK, J. AND TARJAN, R. 1992. Maintaining bridge-connected and biconnected components on-line. *Algorithmica 7,* 1, 433–464.