Combinatorics, Probability and Computing (1998) 7, 307–321. Printed in the United Kingdom
© 1998 Cambridge University Press

Evaluating the Tutte Polynomial for Graphs of Bounded Tree-Width

S. D. NOBLE†

Mathematical Institute, 24–29 St. Giles, Oxford OX1 3LB, England
(e-mail: noble@maths.ox.ac.uk)

Received 27 September 1996; revised 21 May 1997

It is known that evaluating the Tutte polynomial, $T(G;x,y)$, of a graph, G, is #P-hard at all but eight specific points and one specific curve of the (x,y)-plane. In contrast we show that if k is a fixed constant then for graphs of tree-width at most k there is an algorithm that will evaluate the polynomial at any point using only a linear number of multiplications and additions.

1. Introduction and notation

Our notation is fairly standard except that our graphs are allowed to have loops and parallel edges. $V(G)$ and $E(G)$, or just V and E, are used to denote the vertex set and edge set of a graph. We assume that V and E are both finite and let $n = |V|$ and $m = |E|$.

If $A \subseteq E$ then $G|A$ is the *restriction* of G to A formed by deleting all edges except those contained in A. For any $W \subseteq V$, $G : W$ is used to denote the graph with vertex set W and edge set consisting of all edges in E that have both end-points in W. If e is an edge of G then $G \setminus e$ is formed from G by deleting e, G/e is formed from G by contracting e, that is, deleting e and identifying its end-points. The number of connected components of G is denoted by $k(G)$. The *rank* of a set, A, of edges is denoted by $r(A)$ and is given by $r(A) = |V(G)| - k(G|A)$.

We denote the set of partitions of X by $\Pi(X)$ and let $\#\pi$ be the size of a partition, that is, the number of nonempty blocks that make up π. Throughout this paper, the partitions to which we refer are partitions of the vertex set of a graph and so, if π is a partition of X, we say that X is the vertex set of π and refer to elements of X as the vertices of π. We use I_X to denote the partition with vertex set X consisting entirely of singleton blocks. If V is the vertex set of π, then for $W \subseteq V$ the *restriction* to W, $\pi|W$, is formed

† Supported by EPSRC grant and by RAND-ESPRIT. Results first presented at the 15th British Combinatorial Conference, Stirling, July 3–7 1995.

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

by deleting all elements in the partition not contained in W and then deleting any empty blocks that are formed. If π_1 and π_2 have the same vertex set, X, then we define their *join* $\pi_1 \vee \pi_2$ to be the partition of X whose blocks are minimal sets such that, if u and v are in different blocks of $\pi_1 \vee \pi_2$, then u and v are in different blocks of π_1 and π_2. In other words, the operation $\vee$ corresponds to join in the partition lattice. More generally, if π_1 and π_2 have vertex sets X_1 and X_2, respectively, we form their join by first adding the vertices of $X_2 \setminus X_1$ to π_1 as singleton blocks, giving π_1', and similarly forming π_2' by adding the vertices of $X_1 \setminus X_2$ to π_2 as singleton blocks, and finally computing $\pi_1' \vee \pi_2'$. If $A \subseteq E$ and $X \subseteq V$, then $\Pi_X(A)$ denotes the restriction to X of the partition π of V, in which the blocks of π correspond to connected components of $G|A$.

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i | i \in I\}, T = (I, F))$, where $\{X_i | i \in I\}$ is a family of subsets of V, one for each vertex of T, and T is a tree such that

- $\bigcup_{i \in I} X_i = V$,
- for all edges $(v, w) \in E$, there exists $i \in I$ such that $v \in X_i$ and $w \in X_i$,
- for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *tree-width* of a tree-decomposition is $\max_{i \in I} |X_i| - 1$. The *tree-width* of a graph G is the minimum tree-width over all possible tree-decompositions of G. If we give T a root then we can define $Y_i = \{v \in X_j | j = i \text{ or } j \text{ is a descendant of } i\}$.

Many well-studied classes of graphs have bounded tree-width: for instance, series-parallel networks are the graphs with tree-width at most two. A large class of graph problems that are thought to be intractable can be solved when the input is restricted to graphs with tree-width at most a fixed constant k. For example, the NP-complete problems, 3-Colouring and Hamiltonian Circuit can be solved in linear time for graphs of bounded tree-width. See [4] for more information on tree-width.

In our algorithm we assume that for some fixed k we are given a graph, G, of tree-width $\leqslant k$. We first have to compute a tree-decomposition of width $\leqslant k$ such that $|I| \leqslant 2n$ and T is a binary tree. Let $f(k) = k^5 \cdot (2k+1)^{(2k+1)-2} \cdot ((2(2k+1)+3)^{2(2k+1)+3} \cdot (\frac{8}{3} \cdot 2^{2k+2})^{2(2k+1)+3})^{2(2k+1)-1}$. The algorithm given in [3] will, in time $O(f(k) \cdot n)$, produce a tree-decomposition $(\{X_i | i \in I'\}, T' = (I', F'))$ with $|I'| \leqslant n$, and from this it is easy to construct, in time $O(kn)$, a tree-decomposition with $|I| \leqslant 2n$ and T a rooted binary tree.

The *Tutte polynomial* of a graph G is given by

$$T(G; x, y) = \sum_{A \subseteq E} (x-1)^{r(E)-r(A)} (y-1)^{|A|-r(A)},$$

and contains a great deal of information about a graph. Here are some examples.

- At $(1, 1)$, T counts the number of maximal forests of G, or spanning trees if G is connected.
- At $(2, 1)$, T counts the number of forests of G.
- T is said to contain the chromatic polynomial, $P(G; \lambda)$, in that

$$P(G; \lambda) = \lambda^{k(G)} (-1)^{r(E)} T(G; 1-\lambda, 0).$$

- If G is connected then the all-terminal reliability $R(G; p)$ is given by

$$R(G; p) = (1-p)^{m-r(E)} p^{r(E)} T\left(G; 1, \frac{1}{1-p}\right).$$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

The family of hyperbolae H_α defined by

$$H_\alpha = \{(x, y) : (x-1)(y-1) = \alpha\}$$

seems to play a special role in the theory: for instance, the partition function of the Ising model is an evaluation along H_2, and along H_q, for any positive integer q, T specializes to the partition function of the q-state Potts model. A whole host of other specializations of T is contained in [5]. We are ready to state the main result of the paper.

Theorem 1.1. *For any fixed k, there exists an algorithm $\mathscr{A}$ that will input any graph G, with tree-width at most k, and rational numbers $x = p_x/q_x$ and $y = p_y/q_y$, where both p_x and q_x, and p_y and q_y are coprime, and evaluate the Tutte polynomial, $T(G; x, y)$, using at most $O(f(k) \cdot (n+M) \cdot (n+m) \cdot \log(n+m) \cdot \log\log(n+m) \cdot l \log l \cdot \log\log l)$ operations, where $l = \log(|p_x| + |q_x| + |p_y| + |q_y|)$ and M is the largest size of a parallel class of edges.*

Our result extends the work of Arnborg and Proskurowski [2], who gave a linear time algorithm to calculate the reliability of a graph, and also that of Oxley and Welsh [7], who gave a polynomial time algorithm for evaluating the Tutte polynomial for graphs of restricted width, a class that includes series-parallel networks, that is, those graphs with tree-width at most two. This contrasts with the situation for general graphs, where Jaeger, Vertigan and Welsh [6] have shown that the Tutte polynomial is $\#P$-hard to evaluate except at a few special points and along one special curve, a result that can be extended to bipartite planar graphs [9].

Suppose we write $T(G; x, y) = \sum_{i,j} t_{ij} x^i y^j$. A problem motivated by our result is to find an algorithm that will input any graph with tree-width at most k and output a list of the coefficients t_{ij}. We show that an algorithm that does this must have running time $\omega(n^3)$.

2. The algorithm

From here until the end of Section 2.3 we will assume that we are evaluating T at a point (x, y) with $x \neq 1$. Later we show how to compute T along the line $x = 1$; the method we give here does not work if $x = 1$ because it involves dividing by $x-1$. We first give an informal illustration of the idea behind the algorithm. Suppose we have a graph $G = (V_1 \cup V_2, E_1 \cup E_2)$ with $V_1 \cap V_2 = X$, $E_1 \cap E_2 = \emptyset$, and such that any edge in E_i has both end-points in V_i. We refer to a set X that occurs in this way as an *intersecting set*. Now suppose that, for any partition, π, of X and any i and j, we know the number of subsets A of E_1 (E_2) with rank i and cardinality j and satisfying $\Pi_X(A) = \pi$. We denote this number by $N_1(\pi, i, j)$ ($N_2(\pi, i, j)$). Using this information we can calculate $N(\pi, i, j)$, the number of subsets A of $E_1 \cup E_2$ with rank i, cardinality j and satisfying $\Pi_X(A) = \pi$. This is true because, if $A = A_1 \cup A_2$ where $A_1 \subseteq E_1$ and $A_2 \subseteq E_2$, the rank of A depends only on the rank of A_1 and A_2, and on $\Pi_X(A_1)$ and $\Pi_X(A_2)$ but not on the actual edges of A_1 and A_2. More precisely,

$$r(A) = r(A_1) + r(A_2) - |X| - \#\Pi_X(A) + \#\Pi_X(A_1) + \#\Pi_X(A_2).$$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

This means that if $\pi_1 \vee \pi_2 = \pi$ then the number of sets A contributing to $N(\pi, i, j)$ and satisfying $\Pi_X(A \cap E_1) = \pi_1$ and $\Pi_X(A \cap E_2) = \pi_2$ is

$$\sum_{i_1=0}^{i+I} \sum_{j_1=0}^{j} N_1(\pi_1, i_1, j_1) N_2(\pi_2, i + I - i_1, j - j_1),$$

where $I = |X| + \#\pi - \#\pi_1 - \#\pi_2$, and so

$$N(\pi, i, j) = \sum_{\substack{(\pi_1, \pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i_1=0}^{i+I} \sum_{j_1=0}^{j} N_1(\pi_1, i_1, j_1) N_2(\pi_2, i + I - i_1, j - j_1),$$

where again $I = |X| + \#\pi - \#\pi_1 - \#\pi_2$.

Let x and y be fixed with $x \neq 1$, and suppose that, rather than knowing $N_1(\pi, i, j)$ and $N_2(\pi, i, j)$ explicitly, we know the evaluation at the point (x, y) of certain polynomials similar to the Tutte polynomial; that is, we are given

$$\begin{aligned} t_1(\pi) &= \sum_{A \subseteq E_1 : \Pi_X(A) = \pi} (x-1)^{-r(A)} (y-1)^{|A| - r(A)} \\ &= \sum_{i,j} N_1(\pi, i, j)(x-1)^{-i}(y-1)^{j-i} \end{aligned}$$

and

$$\begin{aligned} t_2(\pi) &= \sum_{A \subseteq E_2 : \Pi_X(A) = \pi} (x-1)^{-r(A)} (y-1)^{|A| - r(A)} \\ &= \sum_{i,j} N_2(\pi, i, j)(x-1)^{-i}(y-1)^{j-i}, \end{aligned}$$

and we wish to compute for each $\pi \in \Pi(X)$

$$t(\pi) = \sum_{A \subseteq E_1 \cup E_2 : \Pi_X(A) = \pi} (x-1)^{-r(A)} (y-1)^{|A| - r(A)}.$$

Now, setting $I = I(\pi_1, \pi_2) = |X| + \#\pi - \#\pi_1 - \#\pi_2$, we have

$$\begin{aligned} t(\pi) &= \sum_{i,j} N(\pi, i, j)(x-1)^{-i}(y-1)^{j-i} \\ &= \sum_{i,j} N(\pi, i, j)\big((x-1)(y-1)\big)^{-i}(y-1)^{j} \\ &= \sum_{i,j} \sum_{\substack{(\pi_1, \pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i_1, j_1} \Big[N_1(\pi_1, i_1, j_1) N_2(\pi_2, i + I - i_1, j - j_1) \\ &\qquad \cdot \big((x-1)(y-1)\big)^{-i}(y-1)^{j} \Big] \\ &= \sum_{\substack{(\pi_1, \pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} \sum_{i, j, i_1, j_1} \Big[N_1(\pi_1, i_1, j_1)\big((x-1)(y-1)\big)^{-i_1}(y-1)^{j_1} \\ &\qquad \cdot N_2(\pi_2, i + I - i_1, j - j_1)\big((x-1)(y-1)\big)^{(-i-I+i_1)}(y-1)^{j-j_1} \\ &\qquad \cdot \big((x-1)(y-1)\big)^{I} \Big] \end{aligned}$$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

$$
\begin{aligned}
= \quad & \sum_{\substack{(\pi_1,\pi_2)\\ :\pi_1\vee\pi_2=\pi}} \sum_{i_1,j_1,i_2,j_2} \Big[N_1(\pi_1,i_1,j_1)\big((x-1)(y-1)\big)^{-i_1}(y-1)^{j_1} \\
& \cdot N_2(\pi_2,i_2,j_2)\big((x-1)(y-1)\big)^{-i_2}(y-1)^{j_2} \\
& \cdot \big((x-1)(y-1)\big)^{I} \Big] \\
= \quad & \sum_{\substack{(\pi_1,\pi_2)\\ :\pi_1\vee\pi_2=\pi}} t_1(\pi_1)t_2(\pi_2)\big((x-1)(y-1)\big)^{(|X|+\#\pi-\#\pi_1-\#\pi_2)}. \qquad (2.1)
\end{aligned}
$$

What this rather messy calculation means is that we can calculate t from t_1 and t_2.

2.1. Graphs without parallel edges

We now show how the algorithm works for graphs without parallel edges, although we allow up to one loop at each vertex.

Suppose that we are given G and a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of width k such that T is a binary tree with root r and $|I| \leqslant 2n$. We need to associate each edge of G with a particular node of T. To do this we construct for each $i \in I$ a set of edges D_i such that the sets D_i are pairwise disjoint, $\bigcup_{i\in I} D_i = E$ and $\{u,v\} \in D_i \Rightarrow \{u,v\} \subseteq X_i$. We say that D_i is the set of edges *inside* X_i; in fact D_i will be some subset of the edges induced by X_i. There are many ways of constructing the D_i and any of them will do. For any graph of tree-width at most k, we can obtain one such construction in time $O(m+n)$ just by assigning edges greedily. For each $i \in I$ we define $E_i = \{e \in D_j | j = i$ or j is a descendant of $i\}$ and say that E_i is the set of edges inside Y_i.

For each $i \in I$ and each $\pi \in \Pi(X_i)$, we define $T_i(\pi)$ by

$$T_i(\pi) = \sum (x-1)^{-r(A)}(y-1)^{|A|-r(A)},$$

where the sum is over all sets A satisfying $A \subseteq E_i$ and $\Pi_{X_i}(A) = \pi$, that is, all sets that consist of edges inside Y_i and that partition X_i into connected components given by π.

The algorithm computes all the set of values $\{T_i(\pi) : \pi \in \Pi(X_i)\}$ for each $i \in I$ working upwards from the leaves of T to the root, r. For each i the values $\{T_i(\pi) : \pi \in \Pi(X_i)\}$ are calculated from the values for the children of i using exactly the type of calculation outlined above.

Suppose that we want to calculate T_i where i is a node with two children j and k (this is the harder case). At this stage we will know $T_j(\pi)$ for all $\pi \in \Pi(X_j)$ and $T_k(\pi)$ for all $\pi \in \Pi(X_k)$. We now calculate, for each $\pi \in \Pi(X_i)$, $\mathrm{lift}_j(\pi)$ and $\mathrm{lift}_k(\pi)$, where these are given by

$$
\begin{aligned}
\mathrm{lift}_j(\pi) &= \sum_{\substack{A\subseteq E_j\\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}, \\
\mathrm{lift}_k(\pi) &= \sum_{\substack{A\subseteq E_k\\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}.
\end{aligned}
$$

This is easy because $\mathrm{lift}_j(\pi)$ will be zero if the vertices of $X_i \setminus X_j$ are not present as singleton blocks in π (there are no edges between vertices of $X_i \setminus X_j$ contained in E_j), and

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

otherwise

$$\mathrm{lift}_j(\pi) = \sum_{\pi_j} T_j(\pi_j),$$

where the summation is over all partitions of X_j satisfying $\pi_j|(X_j \cap X_i) = \pi|(X_j \cap X_i)$.

We now calculate, for each $\pi \in \Pi(X_i)$,

$$\mathrm{mix}_i(\pi) = \sum_{\substack{A \subseteq E_j \cup E_k \\ :\Pi_{X_i}(A) = \pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}.$$

This is done using the procedure outlined above and equation (2.1) using the functions lift_j and lift_k in the roles of t_1 and t_2 and X_i as the intersecting set, so that

$$\mathrm{mix}_i(\pi) = \sum_{(\pi_j,\pi_k)} \mathrm{lift}_j(\pi_j)\,\mathrm{lift}_k(\pi_k)\big((x-1)(y-1)\big)^{(|X_i|+\#\pi_i-\#\pi_j-\#\pi_k)}, \tag{2.2}$$

where the summation is over all pairs (π_j, π_k) such that $\pi_j, \pi_k \in \Pi(X_i)$ and $\pi_j \vee \pi_k = \pi$. To compute $T_i(\pi)$ we now just need to take account of the contribution from edges in D_i, that is, the edges inside X_i, so we set

$$\mathrm{con}_i(\pi) = \sum_{\substack{A \subseteq D_i \\ :\Pi_{X_i}(A) = \pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}. \tag{2.3}$$

Finally, we compute $T_i(\pi)$ for each partition π of X_i, using the same procedure as before but with X_i as our intersecting set and con_i and mix_i in place of t_1 and t_2, so that

$$T_i(\pi) = \sum_{(\pi',\pi'')} \mathrm{con}_i(\pi')\,\mathrm{mix}_i(\pi'')\big((x-1)(y-1)\big)^{(|X|+\#\pi-\#\pi'-\#\pi'')}, \tag{2.4}$$

where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi$.

We now present the algorithm that carries out the calculations discussed above.

EVAL-TUTTE

input G, where G has tree-width $\leqslant k$, rational numbers x and y with $x \neq 1$, a tree-decomposition $\{\{X_i | i \in I\}, T = (I, F)\}$ of G with width k, and such that T is a binary tree with specified root r, and also the partition $\{D_i | i \in I\}$

let $T^* = T$

while $T^* \neq \emptyset$ **do**

  let i be a leaf of T^*

  if i is a leaf of T

    then call LEAF(i)

    else if i has one child in T

      then call ONE-CHILD(i)

      else call TWO-CHILDREN(i)

  Delete i from T^*

let $T(G; x, y) = (x-1)^{r(E)} \sum_{\pi \in \Pi(X_r)} T_r(\pi)$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

proc LEAF(i)
while $\pi_i \in \Pi(X_i)$ **do**
  let $T_i(\pi_i) = \sum_A (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$
  where the summation is over all sets, A, of edges contained in D_i
  and satisfying $\Pi_{X_i}(A) = \pi_i$

proc ONE-CHILD(i)
let j be the child of i in T
while $\pi_i \in \Pi(X_i)$ **do**
  if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_j)) + |X_i \setminus X_j|$
    then let $\mathrm{lift}_j(\pi_i) = 0$
    else let $\mathrm{lift}_j(\pi_i) = \sum_{\pi_j} T_j(\pi_j)$
     where the summation is over all partitions π_j of X_j such that
    $\pi_j|(X_i \cap X_j) = \pi_i|(X_i \cap X_j)$
while $\pi_i \in \Pi(X_i)$ **do**
  let $\mathrm{con}_i(\pi_i) = \sum_A (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$
   where the summation is over all sets of edges, A, that are subsets of
  D_i and satisfy $\Pi_{X_i}(A) = \pi_i$
while $\pi_i \in \Pi(X_i)$ **do**
  let $T_i(\pi_i) = \sum_{(\pi',\pi'')} \mathrm{con}_i(\pi')\,\mathrm{lift}_j(\pi'')\big((x-1)(y-1)\big)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$
  where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$
  and $\pi' \vee \pi'' = \pi_i$

proc TWO-CHILDREN(i)
let j and k be the children of i in T
while $\pi_i \in \Pi(X_i)$ **do**
  while $l \in \{j, k\}$ **do**
    if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_l)) + |X_i \setminus X_l|$
      then let $\mathrm{lift}_l(\pi_i) = 0$
      else let $\mathrm{lift}_l(\pi_i) = \sum_{\pi_l} T_l(\pi_l)$
      where the summation is over all partitions π_l of X_l such that
      $\pi_l|(X_i \cap X_l) = \pi_i|(X_i \cap X_l)$
while $\pi_i \in \Pi(X_i)$ **do**
  let $\mathrm{mix}_i(\pi_i) = \sum_{(\pi_j,\pi_k)} \mathrm{lift}_j(\pi_j)\,\mathrm{lift}_k(\pi_k)$
    $\big((x-1)(y-1)\big)^{(|X_i|+\#\pi_i-\#\pi_j-\#\pi_k)}$
  where the summation is over all pairs (π_j, π_k) such that $\pi_j, \pi_k \in \Pi(X_i)$
  and $\pi_j \vee \pi_k = \pi_i$
while $\pi_i \in \Pi(X_i)$ **do**
  let $\mathrm{con}_i(\pi_i) = \sum_A (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$
  where the summation is over all sets of edges, A, that are subsets of
  D_i and satisfy $\Pi_{X_i}(A) = \pi_i$
while $\pi_i \in \Pi(X_i)$ **do**

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

let $T_i(\pi_i) = \sum_{(\pi',\pi'')} \mathrm{con}_i(\pi')\,\mathrm{mix}_i(\pi'')\big((x-1)(y-1)\big)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$
where the summation is over all pairs (π',π'') such that $\pi',\pi'' \in \Pi(X_i)$
and $\pi' \vee \pi'' = \pi_i$

We prove that the algorithm is correct by showing that for each i it calculates T_i correctly: we do this by induction on the height of i in the tree T. If i is a leaf then it is clear that the procedure LEAF sets $T_i(\pi)$ to the correct value. Otherwise, equations (2.2)–(2.4) in the discussion preceding the statement of the algorithm show that the procedure TWO-CHILDREN computes $T_i(\pi)$ correctly given T_j and T_k, where j and k are the two children of i. Given that TWO-CHILDREN is correct, it is easy to see that ONE-CHILD is correct because it is just the same as TWO-CHILDREN but omitting the calculation of mix. Once we know T_r, the algorithm correctly computes the answer by setting $T(G;x,y) = \sum_{\pi\in\Pi(X_r)}(x-1)^{r(E)}T_r(\pi)$.

2.2. Parallel edges

The algorithm given above will return the correct answer for graphs with parallel edges but may not be efficient. With the definition as above it is possible that, for some i, $|D_i|$ could be very large and not bounded by a function of k, and so the number of operations needed to compute a sum over all subsets of D_i may no longer be polynomially bounded in the size of the input graph. We extend the construction of the sets D_i to graphs with parallel edges by stipulating that $\bigcup_{i\in I} D_i$ contains precisely one edge from each parallel class, that is, a maximal set of edges all of which are in parallel with each other. The other conditions, namely that the sets D_i are pairwise disjoint and for all u and v $\{u,v\} \in D_i \Rightarrow \{u,v\} \subseteq X_i$, remain the same. We say a set A is *represented* in D_i if for each $e \in A$, either $e \in D_i$ or D_i contains an edge in parallel with e and denote this by $A \preceq D_i$. We let $m(e)$ be the size of the parallel class containing e.

The only problem comes when we compute con and leaf. Computing these two functions involves exactly the same calculation, so we only show how to modify con. Now,

$$\mathrm{con}_i(\pi) = \sum_{\substack{A\subseteq D_i \\ :\Pi_{X_i}(A)=\pi}} (x-1)^{-r(A)}(y-1)^{|A|-r(A)}$$

but, as we noted, this summation may be too large to compute efficiently. However,

$$\mathrm{con}_i(\pi) = \sum_{\substack{A\subseteq D_i \\ :\Pi_{X_i}(A)=\pi}} \sum_{(B_1,\ldots,B_{|A|})} (x-1)^{-r(A)}(y-1)^{|B_1|+\cdots+|B_{|A|}|-r(A)},$$

where if $A = \{e_1,\ldots e_l\}$ then the inner summation is over all l-tuples $(B_1,\ldots B_l)$ such that, for all j, B_j is nonempty and contained in the parallel class containing e_j. If $y \neq 1$ then

$$\mathrm{con}_i(\pi) = \sum_{\substack{A\subseteq D_i \\ :\Pi_{X_i}(A)=\pi}} \big((x-1)(y-1)\big)^{-r(A)} \prod_{e\in A}(y^{m(e)}-1),$$

and if $y = 1$ then

$$\mathrm{con}_i(\pi) = \sum_{\substack{A\subseteq D_i \\ :\Pi_{X_i}(A)=\pi,|A|=r(A)}} (x-1)^{-r(A)} \prod_{e\in A} m(e).$$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

To avoid having to calculate y^j repeatedly for the same value of j, we can compute the set $\{y, y^2, \ldots, y^M\}$, where $M = \max_{e \in E} m(e)$, before running the algorithm.

2.3. Complexity

Here we calculate an upper bound on the running time of our algorithm. We denote by $t(n, m, k, x, y, M)$ the maximum number of operations needed to evaluate $T(G; x, y)$ when G is a graph with n vertices, tree-width at most k, m edges and such that the maximum size of any parallel class is M. We let $\alpha(n, m, k, x, y, M)$ be the maximum time needed for one multiplication or addition during the evaluation of $T(G, x, y)$. There are four stages of preprocessing.

(1) Finding a tree-decomposition of width at most k which can be done in time $O(f(k) \cdot n)$ using the algorithm given in [3].
(2) Constructing a tree-decomposition where T is a binary tree that requires time $O(nk)$.
(3) Computing the partition $\{D_i : i \in I\}$ that requires time $O(m + n)$.
(4) Computing the numbers $\{y, y^2, \ldots, y^M\}$. The number of operations required for this stage is $O(M\alpha(n, m, k, x, y, M))$.

If π_1 is a partition of X_1 and π_2 is a partition of X_2 where $\max\{|X_1|, |X_2|\} = l$, then deciding whether $\pi_1 = \pi_2$ and computing $\pi_1 \vee \pi_2$ both require $O(l^2)$ operations. Computing $\Pi_X(A)$ where A is a set consisting of edges with both end-points in X and at most one member from each parallel class takes time $O(|X|^3)$. Recall that $B(k)$ denotes the kth Bell number, that is, the number of distinct partitions of a set of size k.

Because $|I| \leqslant 2n$, the number of operations needed for the main part of the algorithm, that is, omitting the preprocessing, is $O(nt'(n, m, N, k, x, y))$, where t' is the maximum time required for one call to the procedure TWO-CHILDREN.

The procedure TWO-CHILDREN consists of the following four stages.

(1) Calculation of lift_j and lift_k, for which we need time $O\big((B(k+1))^2(k^2+\alpha)\big)$.
(2) Evaluating mix takes time $O\big((B(k+1))^3(k^2+k\alpha)\big)$.
(3) The computation of con needs $O\big(B(k+1)2^{(k+1)^2}(k^3+k^2\alpha)\big)$.
(4) The final stage requires $O\big((B(k+1))^3(k^2+k\alpha)\big)$.

This gives a total time for a call to TWO-CHILDREN of $O\big((B(k+1))^3 \cdot 2^{(k+1)^2} \cdot \alpha\big)$.

Finally, we need to compute the maximum time for one arithmetical operation. To add, subtract, multiply or divide two l-bit integers takes $O(l \log l \log\log l)$ operations [1]. lift, con, mix, T_i are of the form $\sum_{A \in \mathscr{A}} (x-1)^{-r(A)}(y-1)^{(|A|-r(A))}$, where $\mathscr{A}$ is a subset of power-set of E and so the numbers involved in the calculations are either of this form or $\big((x-1)(y-1)\big)^j$, where $0 \leqslant j \leqslant 2k+2$, or y^j, where $0 \leqslant j \leqslant M$. Now suppose $x = p_x/q_x$ and $y = p_y/q_y$, where p_x, q_x, p_y and q_y are integers such that p_x and q_x are coprime, and p_y and q_y are coprime. Note that $p_x \neq q_x$ since $x \neq 1$. Now,

$$\sum_{A \in \mathscr{A}} (x-1)^{-r(A)}(y-1)^{(|A|-r(A))} = \sum_{A \in \mathscr{A}} \left(\frac{p_x - q_x}{q_x}\right)^{-r(A)} \left(\frac{p_y - q_y}{q_y}\right)^{(|A|-r(A))}$$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

$$= \sum_{A\in\mathscr{A}} \left(\frac{\left(\frac{(p_x-q_x)}{q_x}\right)^{(r(E)-r(A))} \left(\frac{(p_y-q_y)}{q_y}\right)^{(|A|-r(A))}}{\left(\frac{(p_x-q_x)}{q_x}\right)^{r(E)}} \right)$$

$$= \sum_{A\in\mathscr{A}} \left(\frac{(p_x-q_x)^{r(E)-r(A)} q_x^{r(A)} (p_y-q_y)^{(|A|-r(A))} q_y^{(m-|A|+r(A))}}{(p_x-q_x)^{r(E)} q_y^m} \right).$$

Considering the denominator, we have $(p_x-q_x)^{r(E)}q_y^m \leqslant (|p_x|+|q_x|)^n |q_y|^m$, and for the numerator, we have

$$\begin{aligned} &\sum_{A\in\mathscr{A}} (p_x-q_x)^{r(E)-r(A)} \cdot q_x^{r(A)} \cdot (p_y-q_y)^{(|A|-r(A))} \cdot q_y^{(m-|A|+r(A))} \\ &\quad\leqslant \sum_{A\subseteq E} |p_x-q_x|^{r(E)-r(A)} \cdot |q_x|^{r(A)} \cdot |p_y-q_y|^{(|A|-r(A))} \cdot |q_y|^m \\ &\quad\leqslant T(G; |p_x-q_x|+1, |p_y-q_y|+1) \cdot |q_x|^{r(E)} \cdot |q_y|^m \\ &\quad\leqslant (|p_x|+|q_x|+|p_y|+|q_y|+2)^m \cdot |q_x|^n \cdot |q_y|^m. \end{aligned}$$

The penultimate inequality follows because $T(G) = \sum_{A\subseteq E} (x-1)^{r(E)-r(A)} (y-1)^{|A|-r(A)}$, and the final one is obtained using equations (4.1)–(4.3) and induction on the number of edges of the graph. This means that an upper bound on the modulus of any of the numbers occurring in the denominator or numerator of numbers used in the calculations is

$$(|p_x|+|q_x|+|p_y|+|q_y|+2)^{(n+2m+2k+2)}.$$

The calculations within the main part of the algorithm all give quantities of the form $\sum_{A\in\mathscr{A}} (x-1)^{-r(A)} (y-1)^{|A|-r(A)}$, where $\mathscr{A}$ is a subset of the power-set of E, and so we can avoid problems arising due to the numerator and denominator having a large common factor and thus becoming large themselves, because we can reduce the fraction with 2 integer divisions so that the denominator is $|p_x-q_x|^{r(E)}|q_y|^m$. We have shown that $\alpha(n,m,k,x,y,M) \leqslant s \log s \log\log s$, where $s = (n+2m+2k+2)\log(|p_x|+|q_x|+|p_y|+|q_y|+2)$, and so the running time for the main part of the algorithm is $O\big(n(B(k+1))^3 \cdot 2^{(k+1)^2} \cdot (n+m+k) \cdot \log(|p_x|+|q_x|+|p_y|+|q_y|) \cdot \log s \cdot \log\log s\big)$, and the running time for the whole algorithm is

$$O\left(f(k) \cdot (n+M) \cdot (n+m) \cdot \log(n+m) \cdot \log\log(n+m) \cdot l \log l \cdot \log\log l\right),$$

where $l = \log(|p_x|+|q_x|+|p_y|+|q_y|)$.

Suppose the input graph has no parallel edges and G has tree-width k. By increasing the size of some of the X_i if necessary, it is possible to show that there exists a tree-decomposition for G such that, for each i, $|X_i| = k+1$. By adding extra vertices to T we can construct a tree-decomposition $(\{X_i | i \in I\},\ T = (I,F))$ for G such that T is rooted, for all $i \in I$, $|X_i| = k$ and, for any edge $\{i,j\}$ of T, $|X_i \cap X_j| = k$. If our tree-decomposition satisfies both these conditions and G has n vertices, then $I = n-k$. There are at most $k(k+1)/2 + (k+1)$ edges between vertices in the root of T and for any other node i of T there is precisely one vertex that does not appear in any ancestor of i in T, and hence between the vertices of X_i there are at most $k+1$ edges that have not been previously

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

counted for an ancestor of X_i. Therefore, the total number of edges is at most

$$(|I|-1)(k+1)+k(k+1)/2+(k+1)=\frac{(k+1)(2n-k)}{2},$$

so if the input graph has no parallel edges the total running time is at most

$$O\left(f(k)\cdot n^2k\cdot\log(nk)\cdot\log\log(nk)\cdot l\log l\cdot\log\log l\right).$$

3. Case $x = 1$

The algorithm given above will not work when $x = 1$ because it will try to divide by 0. To avoid this problem, we use much the same notation as before but assume without loss of generality that our input graph is connected and for the moment has no parallel edges, although as before we allow up to one loop at each vertex. We define $T'_i(\pi)$ by

$$T'_i(\pi)=\sum_A (y-1)^{|A|-r(A)},$$

where the summation is over sets A of edges contained in E_i, and such that each vertex of Y_i is connected to a vertex of X_i in the graph $G|A$. We note that an isolated vertex in X_i is connected to a vertex in X_i since it is connected to itself. The modified algorithm calculates $T'_i(\pi)$ for all π and for each i working upwards from the leaves to the root.

We first give the algorithm and then show that it is correct.

EVAL-TUTTE′
input G where G is connected and has tree-width $\leqslant k$, a rational number y and a tree-decomposition $\{\{X_i|i\in I\}, T=(I,F)\}$ of G with width k, and such that T is a binary tree with specified root r, and also the partition $\{D_i|i\in I\}$
let $T^*=T$
while $T^*\neq\emptyset$ **do**
    let i be a leaf of T^*
    if i is a leaf of T
        then call LEAF′(i)
        else if i has one child in T
            then call ONE-CHILD′(i)
            else call TWO-CHILDREN′(i)
    Delete i from T^*
let $T(G;1,y)=T_r(\pi)$ where $\pi\in\Pi(X_r)$ and $\#\pi=1$

proc LEAF′(i)
while $\pi_i\in\Pi(X_i)$ **do**
    let $T_i(\pi_i)=\sum_A (y-1)^{|A|-r(A)}$
    where the summation is over all sets, A, of edges contained in D_i and satisfying $\Pi_{X_i}(A)=\pi_i$

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

proc ONE-CHILD′(i)
let j be the child of i in T
while $\pi_i \in \Pi(X_i)$ **do**
  if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_j)) + |X_i \setminus X_j|$
    then let $\text{lift}'_j(\pi_i) = 0$
    else let $\text{lift}'_j(\pi_i) = \sum_{\pi_j} T'_i(\pi_j)$
    where the summation is over all partitions π_j of X_j such that $\pi_j|(X_i \cap X_j) = \pi_i|(X_i \cap X_j)$ and every block of π_j contains at least one vertex of X_i
while $\pi_i \in \Pi(X_i)$ **do**
  let $\text{con}'_i(\pi_i) = \sum_A (y-1)^{|A|-r(A)}$
  where the summation is over all sets of edges, A, that are subsets of D_i and satisfy $\Pi_{X_i}(A) = \pi_i$
while $\pi_i \in \Pi(X_i)$ **do**
  let $T'_i(\pi_i) = \sum_{(\pi',\pi'')} \text{con}'_i(\pi')\,\text{lift}'_j(\pi'')(y-1)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$
  where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi_i$

proc TWO-CHILDREN′(i)
let j and k be the children of i in T
while $\pi_i \in \Pi(X_i)$ **do**
  while $l \in \{j, k\}$ **do**
    if $\#\pi_i \neq \#(\pi_i|(X_i \cap X_l)) + |X_i \setminus X_l|$
      then let $\text{lift}'_l(\pi_i) = 0$
      else let $\text{lift}'_l(\pi_i) = \sum_{\pi_l} T'_i(\pi_l)$
      where the summation is over all partitions of X_l such that $\pi_l|(X_i \cap X_l) = \pi_i|(X_i \cap X_l)$ and each block of π_l contains at least one vertex of X_i
while $\pi_i \in \Pi(X_i)$ **do**
  let $\text{mix}'_i(\pi_i) = \sum_{(\pi_j,\pi_k)} \text{lift}'_j(\pi_j)\,\text{lift}'_k(\pi_k)(y-1)^{(|X_i|+\#\pi_i-\#\pi_j-\#\pi_k)}$
  where the summation is over all pairs (π_j, π_k) such that $\pi_j, \pi_k \in \Pi(X_i)$ and $\pi_j \vee \pi_k = \pi_i$
while $\pi_i \in \Pi(X_i)$ **do**
  let $\text{con}'_i(\pi_i) = \sum_A (y-1)^{|A|-r(A)}$
  where the summation is over all sets of edges, A, that are subsets of D_i and satisfy $\Pi_{X_i}(A) = \pi_i$
while $\pi_i \in \Pi(X_i)$ **do**
  let $T'_i(\pi_i) = \sum_{(\pi',\pi'')} \text{con}'_i(\pi')\,\text{mix}'_i(\pi'')(y-1)^{(|X|+\#\pi_i-\#\pi'-\#\pi'')}$
  where the summation is over all pairs (π', π'') such that $\pi', \pi'' \in \Pi(X_i)$ and $\pi' \vee \pi'' = \pi_i$

We consider the operation of the procedure TWO-CHILDREN′. Suppose we want to compute $T'_i(\pi)$, where i is a node with two children j and k, and we know T'_j and T'_k. We

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

first compute for all $\pi \in \Pi(X_i)$, $\text{lift}'_j(\pi)$ and $\text{lift}'_k(\pi)$, where lift'_j is given by

$$\text{lift}'_j(\pi) = \sum_A (y-1)^{|A|-r(A)},$$

where the summation is over all those sets A contained in E_j such that $\Pi_X(A) = \pi$, and each vertex of Y_j is connected to a vertex of X_i in $G|A$; lift'_k is defined analogously. If the vertices of $X_i \setminus X_j$ are not present as singleton blocks in π, then $\text{lift}'_j(\pi)$ will be zero and otherwise

$$\text{lift}'_j(\pi) = \sum_{\pi_j} T'_j(\pi_j),$$

where the summation is over those partitions of X_j satisfying $\pi_j|(X_i \cap X_j) = \pi|(X_i \cap X_j)$ and such that each block of π_j contains a vertex of X_i. This last restriction is needed to ensure that lift'_j is a sum over subsets, A, of E_j such that every vertex of Y_j is connected in $G|A$ to a vertex of X_i rather than just to a vertex of X_j. The procedure TWO-CHILDREN′ next calculates mix'_i, which is given by

$$\text{mix}'_i(\pi) = \sum_A (y-1)^{|A|-r(A)},$$

where the summation is over all subsets, A, of $E_j \cup E_k$ such that $\Pi_{X_i}(A) = \pi$, and every vertex of $Y_j \cup Y_k$ is connected to a vertex of X_i. To find an expression for mix' in terms of lift' we have to modify equation (2.1).

Suppose we have a graph $G = (V_1 \cup V_2, E_1 \cup E_2)$ with $V_1 \cap V_2 = X$, $E_1 \cap E_2 = \emptyset$, and such that any edge in E_i has both end-points in V_i. For each $\pi \in \Pi(X)$ and for each $i \in \{1, 2\}$, we define

$$t_i(\pi) = \sum_A (y-1)^{|A|-r(A)},$$

where the summation is over subsets A of E_i satisfying $\Pi_X(A) = \pi$, and such that every vertex of V_i is connected to a vertex of X. Now let $t(\pi)$ be given by

$$t(\pi) = \sum_A (y-1)^{|A|-r(A)},$$

where the summation is over all subsets A of $E_1 \cup E_2$ that satisfy $\Pi_X(A) = \pi$, and such that every vertex of $V_1 \cup V_2$ is connected to a vertex of X. By modifying the argument preceding equation (2.1) it is possible to show that

$$t(\pi) = \sum_{\substack{(\pi_1,\pi_2) \\ :\pi_1 \vee \pi_2 = \pi}} t_1(\pi_1) t_2(\pi_2) (y-1)^{(|X| + \#\pi - \#\pi_1 - \#\pi_2)}.$$

This means that we can calculate mix'_i and T'_i from lift'_j and lift'_k using this equation, just as in the main algorithm we could calculate mix_i and T_i using equation (2.1).

The procedure LEAF′ is the same as LEAF and the procedure ONE-CHILD′ is the same as TWO-CHILDREN′ omitting the calculation of mix' just as in the main algorithm. The final stage of the algorithm sets $T(G; 1, y) = T_r(\pi)$, where π is the partition of X_r containing all the vertices of X_r in one block. This is correct since $T(G; 1, y) = \sum_A (y-1)^{|A|-r(A)}$, where the summation is over all those subsets A of E such that $G|A$ is connected, and we assumed initially that G was connected.

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

Exactly the same modification as that for the main algorithm must be made to cope with non-simple graphs. The running time for this algorithm satisfies the bound given for the main algorithm.

4. Computing T

We can write the Tutte polynomial in the form

$$T(G;x,y)=\sum_{i,j} t_{ij}x^i y^j.$$

A natural problem to consider is that where we input a graph of tree-width at most k and output the list of coefficients t_{ij}. In contrast with the problem of evaluating T, this problem has complexity $\omega(n^3)$.

It is easy to construct a family of graphs $\{G_r\}$ with tree-width k such that there are $\omega(n^2)$ coefficients exceeding $2^{\omega(n)}$, and hence $\omega(n^3)$ time is required to list the coefficients. One way to do this is to take a complete graph on k vertices $v_1,\ldots,v_k$ and add vertices $v_{k+1},\ldots,v_{k+r}$ so that each one is connected by a single edge to each of $v_1,\ldots,v_k$ and now add vertices $v_{k+r+1},\ldots,v_{k+2r}$ so that, for each s with $k+r+1\leqslant s\leqslant k+2r$, v_s is connected to $v_2,\ldots,v_k$ and v_{s-r} by a single edge. T can be calculated using the following well-known recurrence relations. If e is an isthmus of G then

$$T(G;x,y)=xT(G/e;x,y), \tag{4.1}$$

if e is a loop then

$$T(G;x,y)=yT(G\setminus e;x,y), \tag{4.2}$$

and if e is neither a loop nor an isthmus then

$$T(G;x,y)=T(G\setminus e;x,y)+T(G/e;x,y). \tag{4.3}$$

We can examine the size of the coefficients of $T(G)$ by ordering the edges and considering the binary tree where each node is labelled with a graph obtained when (4.1)–(4.3) are applied recursively. The root is labelled with G, and if a node is labelled with H then, providing H has at least one edge, the children of H correspond to the graphs obtained from H by deleting and contracting the lexicographically first edge, e, remaining, providing e is neither a loop nor an isthmus. If e is a loop (isthmus) then H has one child corresponding to deleting (contracting) e. The leaves correspond to graphs with $k(G)$ vertices and no edges, and each leaf also corresponds to a term in the expansion of $T(G)$. If in obtaining a leaf L from G we contract i isthmuses and delete j loops, then L corresponds to a term $x^i y^j$.

Now consider G_r. We order the edges so that if $i<j$ those edges adjacent to v_{i+k} and v_{i+k+r} come before those adjacent to v_{j+k} and v_{j+k+r}, and the edges of the K_k come last. The edges adjacent to v_{i+k} and v_{i+k+r} are ordered so that

$$\{v_{i+k},v_{i+k+r}\}\prec\{v_k,v_{i+k+r}\}\prec\cdots\prec\{v_2,v_{i+k+r}\}\prec\{v_k,v_{i+k}\}\prec\cdots\prec\{v_1,v_{i+k}\},$$

where $e\prec f$ means that edge e precedes f. It is easy to see there are ways of deleting or contracting the edges adjacent to v_{k+1} and v_{k+r+1} in order that the rest of the graph

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press

is unchanged, and we have either deleted one loop, contracted one isthmus or neither. In other words there are nodes at depth $2k$ in the binary tree defined above which correspond to $G : (V(G) \backslash \{v_{r+k}, v_{2r+k}\})$ and are reached by deleting one loop or contracting one isthmus or neither. Similarly, for any i we can delete and contract the edges adjacent to v_{k+i} and v_{k+r+i} so that we delete one loop, contract one isthmus or neither and leave the K_k intact. We can delete and contract the K_k so that we contract one isthmus and delete no loops. This means that the coefficient of $x^{i+1}y^j$ is at least $r!/(i!j!(r-i-j)!)$ and so, if $\lfloor r/4 \rfloor \leqslant i, j \leqslant \lfloor r/2 \rfloor$, this coefficient is at least $4^{\lfloor r/4 \rfloor}$.

This means that the running time of an algorithm to list all the coefficients of T must be $\omega(n^3)$ because it takes this long to write them out, and hence, even when we restrict our input graphs to have tree-width at most k, more time is needed to list the coefficients of T than to evaluate it at a point. One approach to this problem is to evaluate T at several points using our algorithm and then use Lagrangian interpolation to find the coefficients.

5. Conclusion

As we mentioned in the introduction, evaluations of the Tutte polynomial correspond to a wide variety of counting problems. In the case where the input graph has tree-width at most k, algorithms for some of these problems already exist: see, for example, [2]. We have shown that, for any of these problems, if we restrict the input to graphs of tree-width at most k, for any fixed k, then there is a polynomial time algorithm. The methods we use can be extended to the Tutte polynomial on signed graphs and the polymatroid (E, f), where E is the edge set of a graph G, and for any subset A of E, $f(A)$ is the number of vertices incident with an edge from A. This polymatroid is described in [8].

References

[1] Aho, A. V., Hopcroft, J. and Ullman, J. D. (1974) *The Design and Analysis of Computer Algorithms*. Addison Wesley, Reading, MA, USA.
[2] Arnborg, S. and Proskurowski, A. (1989) Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.* **23** 11–24.
[3] Bodlaender, H. L. (1993) A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc. 25th Annual ACM Symposium on the Theory of Computing*. ACM Press, pp. 226–234.
[4] Bodlaender, H. L. (1993) A tourist guide through treewidth. *Acta Cybernet*. **11** 1–21.
[5] Brylawski, T. H. and Oxley, J. G. (1992) The Tutte polynomial and its applications. In *Matroid Applications* (N. White, ed.), Cambridge University Press, Cambridge, pp. 123–225.
[6] Jaeger, F., Vertigan, D. and Welsh, D. (1990) On the computational complexity of the Jones and Tutte polynomials. *Math. Proc. Cam. Phil. Soc.* **108** 35–53.
[7] Oxley, J. G. and Welsh, D. J. A. (1992) Tutte polynomials computable in polynomial time. *Discrete Math.* **109** 185–192.
[8] Oxley, J. G. and Whittle, G. P. (1993) Tutte invariants for 2-polymatroids. In *Graph Structure Theory* (N. Robertson and P. D. Seymour, eds), AMS, in *Contemporary Mathematics* **147** pp. 9–19.
[9] Vertigan, D. and Welsh, D. J. A. (1992) The computational complexity of the Tutte plane: the bipartite case. *Comb. Prob. Comp.* **1** 181–187.

https://doi.org/10.1017/S0963548398003551 Published online by Cambridge University Press