SERIE B — INFORMATIK

A polynomial-time algorithm for computation of the Tutte polynomials of graphs of bounded treewidth

Artur Andrzejak*

B 95-16
December 1995

Abstract

For each fixed, positive integer k we give an algorithm which decides if a given graph G has treewidth at most k and if so, it computes the Tutte polynomial of G in time $O((2\,n)^{3+2\,\log_2(c_1)/\log_2(\frac{5}{4})})$, where c_1 is twice the number of partitions of a set with $3\,k+3$ elements. The decision if G has treewidth at most k can be obtained in linear time due to an algorithm of H. Bodlaender.

*artur@inf.fu-berlin.de, Freie Universität Berlin, Institut für Informatik, Takustr. 9, 14195 Berlin, Germany

1 Introduction

Triggered by the paper of Jaeger et al. [JVW90], the computational complexity of the Tutte polynomial has recently received a lot of attention. It was shown in [JVW90] that determining the Tutte polynomial $t(M;\ x, y)$ of a matroid for fixed values a, b of x and y, respectively, is $\#P$-hard unless the point (a, b) of the plane lies on a hyperbola $(x-1)(y-1) = 1$ or is one of 8 special points $(1, 1)$, $(-1, -1)$, $(0, -1)$, $(-1, 0)$, $(i, -i)$, $(-i, i)$, (j, j^2) and (j^2, j), where $i^2 = -1$ and $j = e^{2\pi i/3}$ ([Wel94] and [Wel93]). Later Vertigan [Ver92] showed that a similar result holds for the Tutte polynomial of a planar graph.

The evaluations of the Tutte polynomial range very wide and include such quantities as the chromatic and flow polynomials of a graph, the (all terminal) probability of a network, the partition functions of the Ising and Potts models of statistical physics and the Jones and Kauffman bracket polynomials of an alternating link ([BO92], [JVW90], [Wel93]). As a consequence, most of the quantities obtained by evaluation of the Tutte polynomial (especially all listed above) are $\#P$-hard for the class of planar graphs. In spite of this fact it seems to be interesting to investigate for which classes of graphs the calculation of the Tutte polynomial can be done in polynomial time. This question has been partially answered in the matroid case in [OW92]. For graphs it follows that one of these classes are the series-parallel networks.

In this paper we show that the Tutte polynomial of a graph of bounded treewidth can be computed in polynomial time. More precisely, we show that for each positive integer k there is an algorithm which decides in linear time if a given graph G has treewidth at most k and if so, the Tutte polynomial of G is computed in (remaining) time $O((2\,n)^{2+2\,\log_2(c_1)/\log_2(\frac{5}{4})})$, where c_1 is twice the number of partitions of a set with $3\,k+3$ elements. An explicit form of the algorithm is given, but because of large preprocessing time already for $k > 2$ its applicability is limited.

The class of graphs of treewidth bounded by an integer k (or, equivalently, the class of partial k-trees) is well studied and includes such graph classes as series-parallel networks ($k = 2$), chordal graphs with maximal clique size $k+1$ and interval graphs with maximal clique size $k+1$ (see [Bod93b] and [Lee90] for a survey). A huge number of problems being NP-hard in general turn out to be polynomial-time (or even linear-time) solvable for graphs of bounded treewidth. The problems include HAMILTONIAN CIRCUIT, CHROMATIC NUMBER, VERTEX COVER and many more (see [Bod94], [ALS91], [AP89], [AC93], and also [Bod93b] for more bibliography).

2 Definitions and Example

In this section we give some definitions, followed by an example of the application of the main algorithm.

Our definitions follow [Aig79], [Wel93] and [BT95].

For a graph G and $T \subseteq E(G)$ we denote as $G\backslash T$ or $G\backslash e$ if $T = \{e\}$ the deletion of T from G and G/T or G/e the contraction of T from G. In following, let $c(G)$ denote the number of components of G.

If G has an empty edge set then we set the *Tutte polynomial* $t(G;\ x, y)$ or $t(G)$ of G to be 1. Otherwise we have for any $e \in E(G)$

(R1) $t(G) = t(G \backslash e) + t(G/e)$ if e is not a loop or isthmus,

(R2) $t(G) = x\, t(G \backslash e)$ if e is a an isthmus,

(R3) $t(G) = y\, t(G \backslash e)$ if e is a loop.

It can be shown that the Tutte polynomial is well-defined. Obviously $t(G;\ x, y)$ is a 2-variable polynomial in x, y with nonnegative coefficients.

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

- $\bigcup_{i \in I} X_i = V$,
- for all $\{v, w\} \in E$, there exists an $i \in I$ such that $v \in X_i$ and $w \in X_i$,
- for all $i_1, i_2, i_3 \in I$, if i_2 is on the path from i_1 to i_3 in T, then $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$.

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G.

An equivalent notion is a *partial k-tree* (see [Arn85]). We have ([Lee90, p. 550])

Lemma 1 *Let $G = (V, E)$ be a graph with $|V| \geq k$. Then G is a partial k-tree if and only if the treewidth of G is not bigger than k.*

We consider mostly binary and rooted decomposition trees. We understand that a tree $T = (I, F)$ is *binary* if each node has at most two sons and it is rooted if there is a unique node in T called *root*. For such a tree we write $ls(i) \in I$ for the left son of $i \in I$ and $rs(i) \in I$ for the right son of i. We also assume that every node of T is either a leaf or has at least the left son.

Besides a rooted tree decomposition of G our algorithm needs the following partition of $E(G)$. We create it by assigning to each $i \in I$ a set E_i of edges of G "internal" to i in such a way that for different $i, j \in I$ the sets E_i, E_j are disjunct and if $\{v, w\} \in E_i$ then $v, w \in X_i$. By definition of $(T, \mathcal{U})$ such partition of $E(G)$ must exist. Because our algorithm works for any such partition of $E(G)$, we still designate a tree decomposition of G as $(T, \mathcal{U})$ but keep in mind that E_i are fixed for $i = 1, \ldots, |I|$.

Let $s(r)$ be the number of partitions of a set with r many elements. We write $P(Y)$ for a partition of the elements in a set Y and we denote by $P^1(Y)$ the partition of Y, where each element of Y is a singleton. For $i \in I$ and a vertex set Y, let

$G(i, P(Y))$ be a graph obtained in the following way. First we create a graph G', which has vertices in X_i and edges in E_i. Then $G(i, P(Y))$ is obtained from G' by identifying all vertices in each block of the partition P' of $Y \cap X_i$, being a restriction of $P(Y)$ to $Y \cap X_i$. For example, $G(i, P^1(Y)) = G'$ for any vertex set Y. A new vertex corresponding to a block of $P(X_i)$ keeps one or more names of the vertices from which it was created.

For two sets X and Y and their partitions $P(X)$ and $P(Y)$ we extend the notion of their supremum $P(X) \wedge P(Y)$ in the partition lattice as follows. Let C be $X \cup Y$, $P_X(C)$ a partition of C with all blocks of $P(X)$ and in which all elements in $C - X$ are singletons and $P_Y(C)$ a partition of C with all blocks of $P_Y(C)$ and in which all elements of $C - Y$ are singletons. Then $P(X) \wedge P(Y)$ is defined as the usual supremum $P_X(C) \wedge P_Y(C)$ in the lattice of partitions of the set C.

For a node i of T we define the graph $subG(i, P(Y))$ of G, where $P(Y)$ is a partition of a set $Y \subseteq V(G)$, as the following graph. Let G' be the graph union $\bigcup_j G(j, P^1(X_j))$, where j is i itself or a descendant of i. Then $subG(i, P(Y))$ is obtained from G' by identifying for each block of $P(Y)$ all vertices in $V(G') \cap Y$ being in the same block of $P(Y)$. In the way analogous to the definition of $G(i, P(Y))$ we keep all old names of vertices in $V(G') \cap Y$.

For example, for the decomposition tree in Figure 2 (where each dashed ellipse represents a node of T) the graph $subG(A, P^1(\{a, b, c\}))$ is the example graph G from the Figure 1 itself and $subG(B, P^1(\{a, c, d\}))$ is the graph with vertices a, c, d, e, f and edges $\{a, d\}$, $\{c, d\}$, $\{e, d\}$, $\{e, f\}$, $\{f, d\}$. The graph $subG(i, P(Y))$ is not necessary as a data structure for our main algorithm, but it facilitates the proofs and explanations.

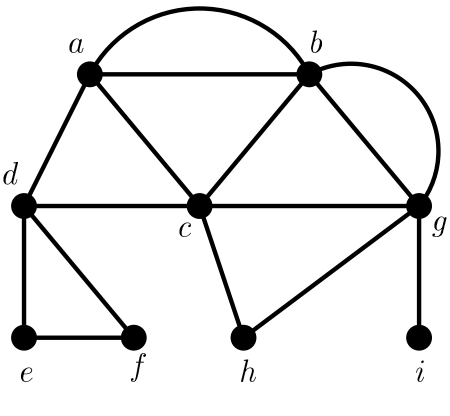

Figure 1: An example graph G

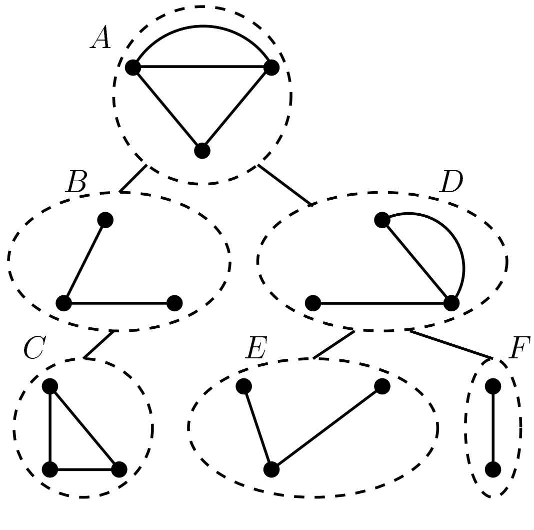

Figure 2: A tree decomposition of G

We would like to give an example of the way our algorithm works. In Figure 1 we see an example graph G on 9 vertices and in Figure 2 a tree decomposition

$(T, \mathcal{U})$ of G of width 2. The tree T must be binary, rooted and its depth must not exceed $2\lceil log_{\frac{5}{4}}(2n)\rceil$. If G has treewidth k, then its tree decomposition $(T, \mathcal{U})$ must have width at most $3k+2$. We will show in Section 4 that we can find such tree decomposition of width at most $r = 3k+2$ of G in time linear in n if G has treewidth at most k.

We obtain the Tutte polynomial of G by calling the recursive procedure $TP(A, P^1(X_A))$, where A is the root of the tree decomposition. The details of the algorithm and the analysis of its running time are described in Section 5. For now, we would like to give a rough idea why the running time is polynomial. We introduce a so called modified splitting tree $\bar{S}$ which corresponds to T. The tree $\bar{S}$ in our example is shown in Figure 3. Each node of this tree corresponds to one call of our procedure. For each node of $\bar{S}$, the corresponding running time of the algorithm is $O(n^2)$ plus a constant depending on $r = 3k+2$.

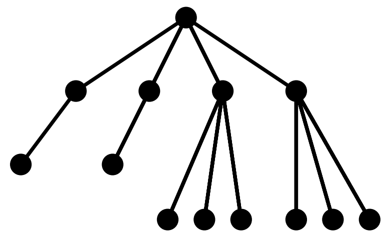

Figure 3: The modified splitting tree $\bar{S}$ corresponding to T

By calling $TP(i, P(Y))$, where $P(Y)$ is a partition of a set $Y \subseteq V(G)$, we cause a so called splitting formula to be applied on the the graphs $G' = subG(rs(i), P(Y))$ and on the complementary part $G'' = subG(ls(i), P(Y)) \cup G(i, P(Y))$. As a result we obtain the Tutte polynomial of $subG(i, P(Y))$. The notion of splitting formulas and a recipe how to obtain them are given in Section 3.

More detailed, a splitting formula requires as input Tutte polynomials of at most $s(r+1)$ graphs obtained from G' by small modifications. Thus, that many times the procedure $TP(rs(i), P'(Y'))$ will be called recursively, for each partition P' of a certain set $Y' \subseteq X_i$ once.

Furthermore, the same splitting formula requires as input the Tutte polynomials of at most $s(r+1)$ many graphs obtained from G'' by small modifications. The situation is slightly more complicated here since we have to regard $G(i, P(Y))$ in our computation and cannot call $TP(ls(i), P''(Y'')$ immediately (for each partition P'' of a certain set $Y'' \subseteq X_i$). The details are given in Algorithm 1. The only significant fact is that after some processing we have to call recursively $TP(ls(i), P''(Y''))$ at most $s(r+1)$ many times.

Thus, each node of $\bar{S}$ has at most $2\,s(r+1)$ many sons. The important point is that although the number of nodes of $\bar{S}$ increases exponentially fast with the depth of $\bar{S}$, the depth of $\bar{S}$ is at most logarithmic in n. Thus, the number of nodes in $\bar{S}$ is polynomial in n. We obtain an algorithm with running time polynomial in n, since its running time associated with each node of $\bar{S}$ is $O(n^2)$.

3 The splitting formulas

Let K and H be two graphs and let $G = K \cup H$ be their graph union with vertices $V(K) \cup V(H)$ and edges $E(K) \cup E(H)$, where $E(K) \cup E(H) = \emptyset$. We call the set $U = V(K) \cap V(H)$ the *connecting intersection* of K and H. For each fixed $r = |U|$, the problem how to calculate in polynomial time the Tutte polynomial $t(G;\ x, y)$ from the Tutte polynomials and from the number of components of some minors of K and of H has been solved by Seiya Negami in [Neg87]. The Tutte polynomial of G can be obtained from the mentioned input by simple matrix multiplication involving a matrix depending only on $r = |U|$.

Before we describe a general recipe on how to obtain these so called splitting formulas we give an example for $r = 3$. Let K and H be graphs with $U = V(K) \cap V(H)$ having three elements u_1, u_2, u_3. We designate by $\Gamma(U)$ the partition lattice of U ordered in such a way that 1 of the lattice $\Gamma(U)$ is the partition $(u_1;\ u_2;\ u_3)$ with three blocks. We name the elements of $\Gamma(U)$ as $p_1 = (u_1, u_2, u_3)$, $p_2 = (u_1;\ u_2, u_3)$, $p_3 = (u_2;\ u_1, u_3)$, $p_4 = (u_3;\ u_1, u_2)$, $p_5 = (u_1;\ u_2;\ u_3)$ and write $|p_i|$ for the number of blocks of a partition p_i. We also define the matrix T_3 with (i, j)-entry being $t^{|p_i \wedge p_j|}$, where t is a variable and $p_i \wedge p_j$ is the supremum of p_i and p_j in $\Gamma(U)$. Then T_3 turns out to be

$$T_3 = \begin{bmatrix} t & t & t & t & t \\ t & t^2 & t & t & t^2 \\ t & t & t^2 & t & t^2 \\ t & t & t & t^2 & t^2 \\ t & t^2 & t^2 & t^2 & t^3 \end{bmatrix}.$$

Furthermore, it has an inverse which is

$$T_3{}^{-1} = \frac{1}{t\,(t-1)\,(t-2)} \begin{bmatrix} t^2 & -t & -t & -t & 2 \\ -t & t-1 & 1 & 1 & -1 \\ -t & 1 & t-1 & 1 & -1 \\ -t & 1 & 1 & t-1 & -1 \\ 2 & -1 & -1 & -1 & 1 \end{bmatrix}.$$

To obtain the splitting formula to calculate $t(G;\ x, y)$ we need the following matrix obtained only from $T_3{}^{-1}$

$$C_3 = d \begin{bmatrix} (x-1)^2 & 1-x & 1-x & 1-x & 2 \\ 1-x & xy-x-y & 1 & 1 & 1-y \\ 1-x & 1 & xy-x-y & 1 & 1-y \\ 1-x & 1 & 1 & xy-x-y & 1-y \\ 2 & 1-y & 1-y & 1-y & (y-1)^2 \end{bmatrix}$$

where
$$d = \frac{1}{(x-1)(x\,y-x-y-1)(x\,y-x-y)}.$$
The (i,j)-entry of C_3 equals to $(y-1)^{|p_i|+|p_j|-3}\, B_{ij}$, where B_{ij} is the (i,j)-entry of ${T_3}^{-1}$ with t replaced by $(x-1)(y-1)$.

As next we obtain some minors of K and H. We write K/p_i for a graph obtained from K by identifying each subset of vertices in U being in the same block of a partition $p_i \in \Gamma(U)$. For example, in K/p_1 all three vertices are identified and all edges between them become loops. As another example, we have $K/p_5 = K$. We define H/p_i, $i = 1 \ldots 5$ in the same way.

As the input for the splitting formula we have ten Tutte polynomials $t(K/p_i;\ x, y)$, $t(H/p_i;\ x, y)$ for $i = 1 \ldots 5$ and the numbers $c(K/p_i)$, $c(H/p_i)$ for $i = 1 \ldots 5$ and $c(G)$, where $c(A)$ is the number of components of a graph A. We combine a part of this input to the vectors

$$\vec{k}_3 = \Big[(x-1)^{c(K/p_1)}\, t(K/p_1;\ x,y), \ldots, (x-1)^{c(K/p_5)}\, t(K/p_5;\ x,y)\Big]$$

and

$$\vec{h}_3 = \Big[(x-1)^{c(H/p_1)}\, t(H/p_1;\ x,y), \ldots, (x-1)^{c(H/p_5)}\, t(H/p_5;\ x,y)\Big].$$

Then the splitting formula is the equation

$$t(G;\ x,y) = (x-1)^{-c(G)}\, \vec{k}_3\, C_3 \vec{h}_3^{\,T}.$$

As we see, the only time-consuming operations needed to obtain the Tutte polynomial of G from the input are two matrix multiplications with entries being polynomials.

We obtain a splitting formula for an arbitrary, fixed $r \geq 2$ in the analogous way. Recall that $s(r)$ is the number of partitions of a set with r many elements. First, we compute all partitions of $U = V(K) \cap V(H)$ and order them in such a way that $p \leq p'$ if and only if the partition p' is a refinement of the partition p. We index these partitions in some way (for example, from $p_i \geq p_j$ it should follow $i \geq j$) and introduce the matrix T_r with (i,j)-entry being $t^{|p_i \wedge p_j|}$. According to [Neg87], the inverse T_r^{-1} of T_r exists, and so we define C_r as the $r(s) \times r(s)$ - matrix with (i,j)-entry being $(y-1)^{|p_i|+|p_j|-r}\, B_{ij}$, where B_{ij} is the (i,j)-entry of ${T_r}^{-1}$ with t replaced by $(x-1)(y-1)$. Thus, C_r depends only on r and on the numbering of the partitions in $\Gamma(U)$.

For a partition $p \in \Gamma(U)$, the graphs K/p and H/p are defined analogously as in the example. Again, we introduce the following vectors formed from the input of the formula

$$\vec{k}_r = \Big[(x-1)^{c(K/p_1)}\, t(K/p_1;\ x,y), \ldots, (x-1)^{c(K/p_{s(r)})}\, t(K/p_{s(r)};\ x,y)\Big]$$

and

$$\vec{h}_r = \left[(x-1)^{c(H/p_1)}\, t(H/p_1;\ x,y), \ldots, (x-1)^{c(H/p_{s(r)})}\, t(H/p_{s(r)};\ x,y)\right].$$

Then the general splitting formula is the equation

$$t(G;\ x,y) = (x-1)^{-c(G)}\, \vec{k}_r\, C_r \vec{h}_r^T. \tag{1}$$

If $|U| \leq 1$, then we obtain $t(G;\ x,y)$ simply by multiplying $t(K;\ x,y)$ with $t(H;\ x,y)$.

4 The tree decomposition algorithm

Our next goal is to show that we can obtain a rooted, binary tree decomposition of depth $O(\log n)$ of a graph G in linear time.

Lemma 2 *Let k be a constant. Given a tree decomposition of width k of a graph G on n vertices, we can compute a rooted, binary tree decomposition of G of depth at most $2\lceil \log_{\frac{5}{4}}(2n)\rceil$ and width at most $3\,k+2$ in time $O(n)$ (using a sequential algorithm).*

Proof: The algorithm for the problem is given in [Bod89, Theorem 4.1 and 4.2], but instead of using the tree contraction technique of Miller and Reif [MR85], we apply the tree contraction algorithm described in [KR90], [AD89] and in [KD88]. (This improvement has been suggested in [BT95]). The cited algorithm is shown to solve the problem with $O(n)$ operations in time $O(\log n)$ on an EREW PRAM. As the processor allocation is no problem (see [KR90]), we can apply the Brent's scheduling principle [Bre74]: a parallel algorithm requiring $w(n)$ operations and $t(n)$ time can be simulated using p processors in time $w(n)/p + t(n)$. Thus, a sequential algorithm for our problem will require $O(n)$ time. $\Box$

Although the problem of determining if a given graph G has treewidth at most k, where both G and k are input of the algorithm is NP-complete, we have the following result [Bod93a].

Theorem 3 *For all positive integers k there exists a linear-time algorithm that tests whether a given graph $G = (V, E)$ has treewidth at most k, and if so, outputs a tree decomposition of G with width at most k.*

A survey over the history of sequential and parallel algorithms for finding a tree decomposition of a graph is given in [BT95].

Combining Lemma 2 and Theorem 3 we obtain the following result.

Corollary 4 *For all positive integers k there is a linear time algorithm, which tests whether a given graph G on n vertices has treewidth at most k, and if so, outputs a binary, rooted tree decomposition of G of depth at most $2\lceil \log_{\frac{5}{4}}(2n)\rceil$ and width at most $3\,k+2$.*

5 The main algorithm

Let $(T,\mathcal{U})$ be a tree decomposition of a graph G, $V(G) = n$, where $T = (I, F)$ is a tree with nodes in I and edges in F and $\mathcal{U} = \{X_i | i \in I\}$ is a family of subsets of V. Let r be the width of $(T,\mathcal{U})$. We assume that T is a binary, rooted tree of depth at most $2\lceil \log_{\frac{5}{4}}(2n)\rceil$.

Before giving a detailed description of the main algorithm, we sketch the way it works. It consists of two recursive procedures TP and *TP-one-son-or-leaf*, both with input arguments i and $P(Y)$, where i is a node of T and $P(Y)$ a partition of a vertex set $Y \subseteq V(G)$. Both procedures return as the result the Tutte polynomial of the graph $subG(i, P(Y))$, but the procedure *TP-one-son-or-leaf* will be called only if it is sure that node i of T has no right son.

If i is a leaf of T, then we can calculate the Tutte polynomial of $G(i, P(Y))$ using rules R1, R2, and R3. This is done in the procedure *TP-one-son-or-leaf.* Otherwise we have the situation as in Figure 4 or in Figure 5. (This is the same representation as in Figure 2 but with a triangle representing the subtree of T).

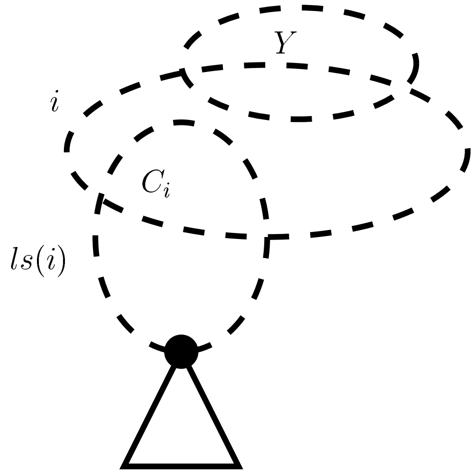

Figure 4: Node i has exactly one son

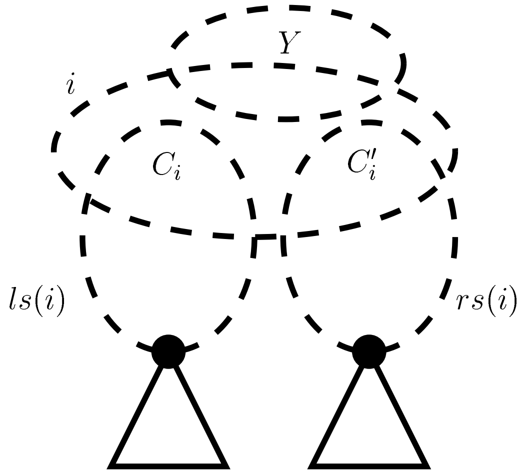

Figure 5: Node i has two sons

In the first case, *TP-one-son-or-leaf* is called in order to apply the splitting formula with the connecting intersection $C_i = X_i \cap X_{ls(i)}$. For each partition P of C_i the splitting formula requires the Tutte polynomial of a graph $G(i, P(Y) \wedge P)$ and the number of its components. The Tutte polynomials of each of these graphs are calculated using the rules R1, R2 and R3. We also need the Tutte polynomial of the graphs $subG(ls(i), P(Y) \wedge P)$ for each partition P of C_i. The Tutte polynomials of these at most $s(|C_i|)$ many graphs are calculated recursively calling $TP(ls(i), P(Y) \wedge P)$ in each case. Now only the number of components of each of these graphs and of $subG(i, P(Y))$ are missing in order to apply the splitting formula.

If the case of Figure 5 occurs, first we have to apply the splitting formula with the connecting intersection being $C'_i = (X_i \cup X_{ls(i)}) \cap X_{rs(i)} = X_i \cap X_{rs(i)}$. For each partition P' of C_i' we have to consider two graphs $subG(rs(i), P(Y) \wedge P')$ (corresponding to the graph K/P' of the splitting formula) and a graph $subG(ls(i), P(Y) \wedge P') \cup G(i, P(Y) \wedge P')$ (corresponding to the graph H/P' in the splitting formula). For each of the graphs $subG(rs(i), P(Y) \wedge P')$ we have to call recursively $TP(rs(i), P(Y) \wedge P')$ and, after removing the edge $\{i, rs(i)\}$ (i.e. cutting off in T the right son of i and its subtree), we have to apply recursively *TP-one-son-or-leaf*$(i, P(Y) \wedge P')$ in order to obtain the Tutte polynomial of $subG(ls(i), P(Y) \wedge P') \cup G(i, P(Y) \wedge P')$. Again, after obtaining the informations on number of components of the graphs required by the splitting formula we can apply the splitting formula, obtaining the Tutte polynomial of $subG(i, P(Y))$.

Algorithm 1

Input: A rooted, binary tree decomposition of a graph G of width at most r.
Output: The Tutte polynomial of G.
Actions: Call the recursive procedure TP by $TP(j, P^1(X_j))$, where j is the root of the decomposition tree T.

<u>**proc**</u> *TP-one-son-or-leaf*$(i, P(Y))$
  <u>**if**</u> $i =$ leaf of T
    <u>**then**</u> <u>**return**</u> the Tutte polynomial $t(G(i, P(Y));\ x, y)$, calculated
        using the rules R1, R2 and R3;
    <u>**else**</u>   (i has only the left son $ls(i)$).
        $C_i := X_i \cap X_{ls(i)}$ and $p := |C_i|$;
        calculate $c(subG(i, P(Y)))$;
        <u>**for**</u> each partition P of C_i <u>**do**</u>
          calculate $t(G(i, P(Y) \wedge P);\ x, y)$ using the rules R1, R2, R3;
          calculate $c(G(i, P(Y) \wedge P))$;
        <u>**od**</u>
        regard the result of the last loop as a vector $\vec{k}_p$ for the splitting
            formula (with graph $K = G(i, P(Y))$);
        <u>**for**</u> each partition P of C_i <u>**do**</u>
          <u>**call**</u> $TP(ls(i), P(Y) \wedge P);\ x, y)$;
          calculate $c(subG(ls(i), P(Y) \wedge P))$;
        <u>**od**</u>
        regard the result of the last loop as a vector $\vec{h}_p$ for the splitting
            formula (with graph $H = subG(ls(i), P(Y))$);
        <u>**return**</u> result of a splitting formula with input $\vec{k}_p$, $\vec{h}_p$
            and $c(subG(i, P(Y)))$;
  <u>**fi**</u>.

<u>**proc**</u> $TP(i, P(Y))$
  <u>**if**</u> right son of i does not exists

then result := **call** *TP-one-son-or-leaf*$(i, P(Y))$;
  return result;
else
  $C'_i := X_i \cap X_{rs(i)}$ and $p := |C'_i|$;
  calculate $c(subG(i, P(Y)))$;
  for each partition P' of C'_i **do**
    call $TP(rs(i), P(Y) \wedge P')$ in order to calculate
      $t(subG(rs(i), P(Y) \wedge P');\ x, y)$;
    calculate $c(subG(rs(i), P(Y) \wedge P'))$;
  od
  regard the result of the last loop as a vector $\vec{k}_p$ for the splitting
    formula (with graph $K = subG(rs(i), P(Y)))$;
  remove the edge $\{i, rs(i)\}$ from T;
    (Now i has only the left son).
  for each partition P' of C'_i **do**
    call *TP-one-son-or-leaf*$(i, P(Y) \wedge P')$ in order to calculate
      $t(subG(i, P(Y) \wedge P');\ x, y)$ (which equals
      $t(subG(ls(i), P(Y) \wedge P') \cup G(i, P(Y) \wedge P');\ x, y)$
      because $\{i, rs(i)\}$ is removed from T);
    calculate $c(subG(i, P(Y) \wedge P'))$;
  od
  regard the result of the last loop as a vector $\vec{h}_p$ for the splitting
    formula (with graph $H = subG(ls(i), P(Y)) \cup G(i, P(Y))$);
  return result of a splitting formula with input $\vec{k}_p$, $\vec{h}_p$
    and $c(subG(i, P(Y)))$;
fi.

The presented algorithm works for any fixed r being the maximal width of the tree decomposition. If r is increased, some preprocessing is necessary. Especially the "tables" of all partitions of a "generic" set of cardinality i for $i = 1, \ldots, r$ must be computed and for each $i = 1, \ldots, r$ the matrix C_i of a corresponding splitting formula must be obtained. Clearly this preprocessing has not a polynomial running time in r unless $P = \#P$, otherwise we would have a polynomial algorithm for the calculation of the Tutte polynomial of any graph G. The preprocessing is also a serious obstacle in practical applicability of the algorithm. This is due to the fact that already for small treewidth k of the input graph the numbers $s(r+1)$, where still $r = 3k + 2$ are very large (see Table 1). Thus, for example if we would like to treat graphs with treewidth at most 3 we have to calculate and store the matrices C_i of the splitting formulas for $i = 2, \ldots, 9$, where C_9 is already a matrix 8427194×8427194. (Remember that C_9 is an inverse of a matrix of the same size!)

To show that Algorithm 1 calculates the Tutte polynomial of G we need to show that for any $i \in I$ the procedure call $TP(i, P(Y))$ with $Y \subseteq V(G)$ terminates and gives the Tutte polynomial of $subG(i, P(Y))$, as $subG(j, P^1(X_j)) = G$, where j is the root of T.

We show this by induction on the height $h(i)$ of $i \in I$ in T, i.e. the depth of the subtree of T with root being i (by convention, the leaves of T should have height 0). We use the notations as in Algorithm 1. If $h(i) = 0$ then $G(i, P(Y)) = subG(i, P(Y))$ and thus $t(subG(i, P(Y));\ x, y)$ is calculated using R1, R2, and R3. For the induction step it is not hard to see from the proceeding example and the listing of Algorithm 1 that the splitting formula is correctly applied to graphs $K = G(i, P(Y))$ and $H = subG(ls(i), P(Y))$ (if i has no right son in T) or to graphs $K = subG(rs(i), P(Y))$ and $H = subG(ls(i), P(Y)) \cup G(i, P(Y))$ (if i has both sons). In the fist case we call for each partition P of C_i the procedure *TP* with the node of T being $ls(i)$, what gives the correct Tutte polynomial of the graph $subG(ls(i), P(Y) \wedge P)$ by induction assumption. In the latter case we call for each partition P' of C'_i the procedure $TP(rs(i), P(Y) \wedge P')$ and the procedure *TP-one-son-or-leaf*$(i, P(Y) \wedge P')$, which after some processing calls the procedure TP with the node of T being $ls(i)$. Thus, also in the second case the induction assumption can be applied. Concluding, we observe that the splitting formulas receive the correct input, as the correctness of the required number of components of the appropriate graphs if obvious. Therefore Algorithm 1 calculates the Tutte polynomial of G. We will analyze in the following how quickly this happens.

There is a rooted *splitting tree* $S = (M, Q)$ associated with the Algorithm 1. Its vertices $j \in M$ are (certain) pairs $(i, P(Y))$, where $i \in I$ and $P(Y)$ is a partition of some vertex set $Y \subseteq G$. S has an edge $e \in Q$ between vertices $(i_1, P_1(Y_1))$ and $(i_2, P_2(Y_2))$ if a procedure *TP* or *TP-one-son-or-leaf* called with arguments $(i_1, P_1(Y_1))$ has called recursively *TP* or *TP-one-son-or-leaf* with arguments $(i_2, P_2(Y_2))$, except if $(i_1, P_1(Y_1)) = (i_2, P_2(Y_2))$ i.e. if the called procedure *TP* noticed that i_1 has no right son so the procedure *TP-one-son-or-leaf* can be called immediately. The vertices of S are defined as the ends of the edges just described. Clearly S is a tree with the root $(j, P^1(X_j))$, where j is a root of T, the decomposition tree. We see that in at most every second step the chain of procedure calls descends one node down the decomposition tree T, therefore the depth of S is at most twice the depth of T.

In order to improve the upper bound on the running time of the Algorithm 1 we will consider a tree $\bar{S}$ being a modified tree S. If node i has both sons, we can assume that *TP*$(i, P(Y))$ has a double-nested loop, where for each partition P' (the outer loop index variable) of $C'_i = X_i \cap X_{rs(i)}$ the inner loop goes over all partitions P (the inner loop index variable) of $C_i = X_i \cap X_{ls(i)}$. That is, we embed the body of the procedure *TP-one-son-or-leaf* called in the loop of *TP* into the body of *TP*). Because all combinations of the partitions P and P' yield no more than all partitions of X_i, we can save some calls of the inner loop. We conclude that the such modified procedure *TP* makes no more than $2\, s(r+1)$ recursive calls of *TP* or *TP-one-son-or-leaf*. The tree $\bar{S}$ is defined for this modified procedure *TP* identically as S has been defined for the original procedure *TP*, i.e its edges corresponds to recursive calls. Thus, each node of $\bar{S}$ has at most $2\, s(r+1)$ sons. The advantage of $\bar{S}$ is the fact that it has depth of T and so this depth is at most $2\, \lceil \log_{\frac{5}{4}}(2\, n) \rceil$, while the depth of S was at most twice the depth of T.

(At this point we can see easily that our worst-case analysis of the running time of the Algorithm 1 (which depends on number of nodes of $\bar{S}$) is generous. First, in most cases the input of TP is $(i, P(Y))$ with a partition $P(Y)$ which reduces the number of different vertices in X_i of $G(i, P(Y))$. Secondly, the subsets C_i and C'_i of X_i will frequently not cover X_i completely reducing the size of the connecting intersections of the splitting formula additionally. It follows that in average the number of sons of a node of $\bar{S}$ will be much smaller than $2\,s(r+1)$).

We can easily bound the number of nodes of the tree $\bar{S}$ with the node set M from above. The depth of $\bar{S}$ is at most $2\,\lceil\log_{\frac{5}{4}}(2\,n)\rceil$ and so we have, putting $c_1 := 2\,s(r+1)$

$$\begin{aligned}
|M| &\leq 1 + c_1 + (c_1)^2 + \cdots + (c_1)^{2\,\lceil\log_{\frac{5}{4}}(2\,n)\rceil} = (c_1-1)^{-1}\,(c_1^{\,2\,\lceil\log_{\frac{5}{4}}(2\,n)\rceil+1} - 1)\\
&< (c_1-1)^{-1}\,c_1^{\,2\,\log_{\frac{5}{4}}(2\,n)+3} = (c_1-1)^{-1}\,c_1^3\,((\frac{5}{4})^{\log_{\frac{5}{4}}(c_1)})^{2\,\log_{\frac{5}{4}}(2\,n)}\\
&= (c_1-1)^{-1}\,c_1^3\,(\frac{5}{4})^{\log_{\frac{5}{4}}(c_1)\,2\,\log_{\frac{5}{4}}(2\,n)} = (c_1-1)^{-1}\,c_1^3\,(2\,n)^{2\,\log_{\frac{5}{4}}(c_1)}.
\end{aligned}$$

Now we have $\log_{\frac{5}{4}}(c_1) = \log_2(c_1)/\log_2(\frac{5}{4})$ and so we obtain

$$|M| < (c_1-1)^{-1}\,c_1^3\,(2\,n)^{2\,\log_2(c_1)/\log_2(\frac{5}{4})}. \tag{2}$$

We obtain the upper bound on the running time of the Algorithm 1 by multiplying the bound on $|M|$ with the maximal time the algorithm spends in a body of a single node of $\bar{S}$. For a node $(i, P(Y)) \in \bar{S}$ we have to execute one or more of the following actions:

(1) Calculate the Tutte polynomials of the graphs $G' = G(i, P(Y) \wedge P')$, where P' ranges over all partitions of a subset of X_i, using the rules R1, R2, and R3.

(2) Find the number of components of the graphs $subG(j, P' \wedge P(Y))$, where j is first the left son and then, if applicable, the right son of i and, if applicable, also $j = i$ and in each case P' ranges over all partitions of a subset of X_i. Furthermore we have to find the number of components of the graph $subG(i, P(Y))$ and of the graphs $G(i, P' \wedge P(Y))$ where again P' ranges of all partitions of a subset of X_i.

(3) Apply at most $s(r+1)+1$ many times splitting formulas and execute the remaining operations in the procedure body such as comparisons, loop initialization etc. This time is dominated by the matrix multiplications of the matrices C_m, $m \leq k$ of the splitting formulas and it is bounded by $c_6\,(s(r+1))^4$, where c_6 is a small constant.

To (1): For a fixed partition P' of a subset W of X_i we estimate in the following the time to calculate $t(G';\ x, y)$. We can treat in our calculation each set of parallel (non-loop) edges $e_1, \ldots, e_m$, $m \geq 2$ in E_i as a single edge because of the following generalization of the rule R1 which can be easily shown by induction:

$$\begin{aligned} t(G';\ x,y) =& t(G' \backslash \{e_2, \ldots, e_m\};\ x,y) + \\ & (y + \cdots + y^{m-1})\, t(G'/e_1 \backslash \{e_2, \ldots, e_m\};\ x,y). \end{aligned}$$

Now, if the rule R1 is applicable to e_1 in $t(G' \backslash \{e_2, \ldots, e_m\};\ x,y)$, we obtain

$$\begin{aligned} t(G';\ x,y) =& t(G' \backslash \{e_1, \ldots, e_m\};\ x,y) + \\ & (1 + y + \ldots, y^{m-1})\, t(G'/e_1 \backslash \{e_2, \ldots, e_m\};\ x,y). \end{aligned}$$

Otherwise, R2 or R3 can be applied to e_1 in $t(G' \backslash \{e_2, \ldots, e_m\};\ x,y)$.

Therefore the calculation time of $t(G';\ x,y)$ depends only on the number of edges in the underlying simple graph G'' of G'. The graph G'' has at most $\binom{|V(G')|}{2}$ edges. As $|V(G')| \leq r+1$, the calculation time of $t(G';\ x,y)$ is bounded from above by a constant c_2 depending only on r. Depending on the implementation, c_2 may vary. If a graph G'' has m edges, then applying R1 or generalized R1 to an appropriate edge we create two minors of G' with $m-1$ edges each. Thus, we have 2^m as a rough upper bound on the number of applications of the rules R1, R2 and R3. It follows that

$$c_2 \leq c_3\, 2^{\binom{r+1}{2}}$$

where each application of a rule R1, R2 or R3 needs a constant time c_3.

Now there are at most $s(r+1)$ partitions P' of a subset W of X_i, and so the algorithm spends at most the time $s(r+1)\, c_2$ applying the rules R1, R2 and R3.

To (2): For each vertex of $G(i, P(Y))$ we obtain the information to which component of $subG(i, P(Y))$ this vertex belongs by executing DFS on $subG(i, P(Y))$ once. The DFS has running time at most $O(|V(G)| + |E(G)|)$ (or $O((|V(G)|)^2)$ if $subG(i, P(Y))$ is not given as an adjacency list), where c_4 is some small constant depending on implementation. Ignoring parallel edges we see that DFS needs the time at most $c_4\, n^2$ for sufficiently large n. Now, for any partition P' of a subset of X_i the number of components $subG(j, P' \wedge P(Y))$, where $j \in \{ls(i), i, rs(i)\}$, can be found using the stored information about the vertices in X_i in time linear in r, with a small constant.

It is not hard to see that the time for finding the number of components of $G(i, P' \wedge P(Y))$ for a fixed P' is linear in r, if we know for each vertex in X_i in which component of $G(i, P(Y))$ it is. This information can be obtained executing once DFS on $G(i, P(Y))$ what takes time $O(r^2)$. Thus, the total time of calculating of the number of components of all graphs mentioned in (2) is bounded by

$$c_4\, n^2 + c_5\, s(r+1)$$

(for sufficiently large n), because we loop at most four times over at most $s(r+1)$ partitions of some subsets of X_i and because $r^2 \leq s(r+1)$ for any $r > 0$. The constants c_4 and c_5 are small.

Summing up the cost of the operations described in (1), (2) and (3) we conclude that the computing time of the Algorithm 1 associated with each node $(i, P(Y))$ is bounded by

$$c_4\, n^2 + (c_3\, 2^{\binom{r+1}{2}} + c_5)\, s(r+1) + c_6\, (s(r+1))^4$$

for sufficiently large n.

Now combining this result with Equation (2) we can bound the running time of the Algorithm 1 from above by

$$(c_1 - 1)^{-1}\, c_1^3\, (2\, n)^{2\, \log_2(c_1)/\log_2(\frac{5}{4})}\, [c_4\, n^2 + (c_3\, 2^{\binom{r+1}{2}} + c_5)\, s(r+1) + c_6\, (s(r+1))^4]$$

for sufficiently large n.

We have just proved the following proposition.

Proposition 5 *Let G be a graph on n vertices and $(T,\mathcal{U})$ a rooted, binary tree decomposition of depth at most $2\,\lceil \log_{\frac{5}{4}}(2\,n)\rceil$ with width at most r. Then the Algorithm 1 computes the Tutte polynomial of G in time $O((2\,n)^{2+2\,\log_2(c_1)/\log_2(\frac{5}{4})})$ (with the constant depending on r) , where c_1 is twice the number of partitions of a set with $r+1$ elements.*

Combining the last proposition with the Corollary 4 we obtain the main result of this paper.

Theorem 6 *For each positive integer k there is an algorithm which decides in linear time if a given graph G on n vertices has treewidth at most k and if so, it calculates the Tutte polynomial $t(G;\; x, y)$ of this graph in total time $O((2\,n)^{3+2\,\log_2(c_1)/\log_2(\frac{5}{4})})$ (with the constant depending on k), where c_1 is twice the number of partitions of a set with $3\,k+3$ elements.*

Table 1 gives the values of c_1 and the exponent $e(k)$ of the expression $(2\,n)^{3+2\,\log_2(c_1)/\log_2(\frac{5}{4})}$ for some small k's.

k	2	3	4	5	8
c_1	42294	8427194	276591709	$1.36415 \cdot 10^{12}$	$1.09143 \cdot 10^{21}$
$e(k)$	98.5	146	197.9	253.5	437.2

Table 1: Some parameters of the main algorithm for small values of k.

As a consequence of the Theorem 6 the great class of problems which are computable in polynomial time for graphs of bounded treewidth can be expanded by the following problems. The solution to each of them is given by direct evaluation of the Tutte polynomial (in some cases multiplied with an easily obtainable factor).

Corollary 7 *For any positive integer k and for each of the following problems there is an algorithm which solves the respective problem in time polynomial in n for a given graph G of treewidth at most k. The problems are to find for G the*

1. *chromatic polynomial of G ([Wel93]);*
2. *number of nowhere zero flows of G ([Wel93]);*
3. *(all terminal) reliability of G ([Wel93]);*
4. *partition function of the q-state Potts model of statistical mechanics (for $q = 2$ it is the partition function of the well-known Ising model) ([Wel93]);*
5. *partition function of the random cluster model introduced by Fortuin and Kasteleyn ([Wel93]);*
6. *number of acyclic orientations of G ([JVW90]);*
7. *number of acyclic suborientations of G ([GS93]);*
8. *number of initially connected acyclic suborientations of G ([GS93]);*
9. *number of connected subdigraphs of G ([GS93]);*
10. *number of different score vectors associated with an orientation of G ([JVW90]);*
11. *number of connected subdigraphs of G ([GS93]);*
12. *number $W(G, m)$ which denotes the number of pairs (A, f) such that A is an acyclic orientation of G and $f : V(G) \to \{1, \ldots, m\}$ is a function which holds $f(u) \geq f(v)$ for every edge of G directed from u to v (for $m = 1$ this is the number of acyclic orientations of G) ([JVW90]);*
13. *Jones polynomial of an oriented alternating link diagram, where G is its associated unsigned "blackface" graph ([Wel93]);*

Many of the listed problems are known to be $\#P$-hard already for any graph class containing all planar graphs ([Wel93]).

6 Acknowledgments

Dominic Welsh *et al.* have proved several years ago that the Tutte polynomial of a graph of bounded treewidth is polynomial-time computable but never published the proof. The author would like to thank him for the opportunity to prove this statement again.

The author also would like to thank Jan Arne Telle, University Bergen, for his very valuable suggestions on algorithms for graphs of bounded treewidth.

References

[AD89] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick and T. Przytycka *A simple parallel tree contraction algorithm*, J. Algorithms 10, No. 2, (1989) 287-302.

[Aig79] M., Aigner, *Combinatorial theory*, (Springer-Verlag New York, Inc. 1979).

[AC93] Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski and Detlef Seese, *An algebraic theory of graph reduction*, J. Assoc. Comput. Mach. 40, No. 5, (1993) 1134-1164.

[ALS91] Stefan Arnborg, Jens Lagergren and Detlef Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms 12, No. 2, (1991) 308-340.

[AP89] Stefan Arnborg and Andrzej Proskurowski, *Linear time algorithms for NP-hard problems restricted to partial k-trees*, Discrete Appl. Math. 23, No. 1, (1989) 11-24.

[Arn85] Stefan Arnborg, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey*, BIT 25, (1985) 2-23.

[BO92] Thomas Brylawski and James Oxley, *The Tutte polynomial and its applications*, Matroid applications, Encycl. Math. Appl. 40, (1992) 123-225.

[BT95] Hans L. Bodlaender and T., Hagerup, *Parallel algorithms with optimal speedup for bounded treewidth*, Extended abstract in proceedings ICALP. Technical Report UU-CS-1995-25. (Can be obtained at the URL http://www.cs.ruu.nl/people/hansb/).

[Bod89] Hans L. Bodlaender, *NC-Algorithms for graphs with small treewidth*, in: J. van Leeuwen, ed., Lecture Notes in Computer Science. v. 344. Conference: Graph-Theoretic Concepts in Computer Science, International Workshop WG '88, Amsterdam (Netherlands), (Springer-Verlag, Berlin 1989), 1-10.

[Bod93a] Hans L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, in: Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, (ACM Press, 1993), 226-234.

[Bod93b] Hans L. Bodlaender, *A tourist guide through treewidth*, Acta Cybern. 11, No. 1-2, (1993) 1-21.

[Bod94] Hans L. Bodlaender, *Improved self-reduction algorithms for graphs with bounded treewidth*, Discrete Appl. Math. 54, No. 2-3, (1994) 101-115.

[Bre74] Richard P. Brent, *The parallel evaluation of general arithmetic expressions*, J. Assoc. comput. Machin. 21, (1974) 201-206.

[GS93] Ira M. Gessel and Bruce E. Sagan, *The Tutte polynomial of a graph, depth-first search, and simplicial complex partitions*, to appear in the Electronic Journal of Combinatorics in an issue dedicated to Dominique Foata (The Journal can be viewed at the URL http://ejc.math.gatech.edu:8080/Journal/journalhome.html).

[JVW90] F. Jaeger, D. L. Vertigan and D. J. A. Welsh, *On the computational complexity of the Jones and Tutte polynomials*, Math. Proc. Camb. Philos. Soc. 108, No. 1, (1990) 35-53.

[KD88] S. Rao Kosaraju and Arthur Delcher, *Optimal parallel evaluation of tree-structured computations by raking*, in: VLSI algorithms and architectures, Proc. 3rd Aegean Workshop Comput., Corfu / Greece 1988, Lect. Notes Comput. Sci. 319, (1988) 101-110.

[KR90] Richard M. Karp and Vijaya Ramachandran, *Parallel algorithms for shared-memory machines*, in: J. Van Leeuwen, ed., Handbook of theoretical computer science, vol. A: Algorithms and Complexity. (MIT Press, Cambridge, MA, 1990), 869-941.

[Lee90] J. van Leeuwen, *Graph algorithms*, in: J. Van Leeuwen, ed., Handbook of theoretical computer science, vol. A: Algorithms and Complexity. (MIT Press, Cambridge, MA, 1990), 525-631.

[MR85] G. L. Miller and J. H. Reif, *Parallel tree contraction and its application*, in: Foundations of computer science. Papers of the 26th annual symposium, Portland, OR, Oct. 21-23, 1985, (IEEE Computer Society Press, Washington DC, 1985), 478-489.

[Neg87] Seiya Negami, *Polynomial invariants of graphs*, Trans. Am. Math. Soc. 299, (1987) 601-622.

[OW92] J. G. Oxley and D. J. A. Welsh, *Tutte polynomials computable in polynomial time*, Discrete Math. 109, No. 1-3, (1992) 185-192.

[Ver92] V. L. Vertigan, The computational complexity of Tutte invariants for planar graphs, to appear.

[Wel93] D. J. A. Welsh, *Complexity: Knots, colourings and counting*, London Mathematical Society Lecture Note Series. v. 186 (Cambridge University Press, Cambridge 1993).

[Wel94] D. J. A. Welsh, *The computational complexity of knot and matroid polynomials*, Discrete Math. 124, No. 1-3, (1994) 251-269.