Computing the Tutte Polynomial on Graphs of Bounded Clique-Width

Omer Giménez[1*], Petr Hliněný[2**], and Marc Noy[1***]

[1] Department of Applied Mathematics
Technical University of Catalonia
Jordi Girona 1–3, 08034 Barcelona, Spain

e-mail: `[omer.gimenez, marc.noy]@upc.edu`

[2] Department of Computer Science, FEI,
Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava, Czech Republic

e-mail: `petr.hlineny@vsb.cz`

Abstract. The Tutte polynomial is a notoriously hard graph invariant, and efficient algorithms for it are known only for a few special graph classes, like for those of bounded tree-width. The notion of clique-width extends the definition of cograhs (graphs without induced P_4), and it is a more general notion than that of tree-width. We show a subexponential algorithm (running in time $\exp O(n^{2/3})$) for computing the Tutte polynomial on cographs. The algorithm can be extended to a subexponential algorithm computing the Tutte polynomial on on all graphs of bounded clique-width. In fact, our algorithm computes the more general U-polynomial.

Keywords: Tutte polynomial, cographs, clique-width, subexponential algorithm, U polynomial.
2000 Math Subjects Classification: **05C85, 68R10**

1 Introduction

The Tutte polynomial $T(G;x,y)$ of a graph G is a powerful invariant with many applications, not only in graph theory but also in other fields such as knot theory and statistical physics. One important feature of the Tutte polynomial is that by evaluating $T(G;x,y)$ at special points in the plane one obtains several parameters of G. For example, $T(G;1,1)$ is the number of spanning trees of G and $T(G;2,1)$ is the number of forests (that is, spanning acyclic subgraphs) of G.

A question that has received much attention is whether the evaluation of $T(G;x,y)$ at a particular point of the (x,y) plane can be done in polynomial

[*] Supported by Beca Fundació Crèdit Andorrà and Project BFM2001-2340.
[**] Supported by Czech research grant GAČR 201/05/050 and partly by the program "Information Society" of the Czech Academy of Sciences, project No. 1ET101940420.
[***] Supported by Project BFM2001-2340.

time. Jaeger, Vertigan and Welsh [8] showed that evaluating the Tutte polynomial of a graph is #P-hard at every point except those lying on the hyperbola $(x-1)(y-1)=1$ and eight special points, including at $(1,1)$ which gives the number of spanning trees. In each of the exceptional cases the evaluation can be done in polynomial time. On the other hand, the Tutte polynomial can be computed in polynomial time for graphs of bounded tree-width. This was obtained independently by Andrzejak [2] and Noble [11]. Recently Hliněný [7] has obtained the same result for matroids of bounded branch-width representable over a fixed finite field, which is a substantial generalization of the previous results. See [5] for additional references on this subject.

In this paper we study the problem of computing the Tutte polynomial for cographs and, more generally, for graphs of bounded clique-width. A graph has clique-width $\leq k$ if it can be constructed using k labels and the following four operations: 1) create a new vertex with label i; 2) take the disjoint union of several labeled graphs; 3) add all edges between vertices of label i and label j; and 4) relabel all vertices with label i to have label j. An expression defining a graph G built from the above four operations using k labels is a *k-expression* for G. A *cograph* is a graph of clique-width at most two; equivalently, it is a graph containing no induced path P_4 on four vertices.

Although a class of graphs with bounded tree-width has also bounded clique-width, the converse is not true. For instance, complete graphs have clique-width two. It is well-known that all problems expressible in monadic second order logic of incidence graphs become polynomial time solvable when restricted to graphs of bounded tree-width. For bounded clique-width less is true: all problems become polynomial time solvable if they are expressible in monadic second-order logic using quantifiers on vertices but not on edges (adjacency graphs) [3].

Our main results are as follows:

Theorem 1.1. *The Tutte polynomial of a cograph with n vertices can be computed in time* $\exp\big(O(n^{2/3})\big)$.

Theorem 1.2. *Let G be a graph with n vertices of clique-width k along with a k-expression for G as an input. Then the Tutte polynomial of G can be computed in time* $\exp\big(O(n^{1-1/(k+2)})\big)$.

Theorem 1.2 is not likely to hold for the class of all graphs, since it would imply the existence of a subexponential algorithm for 3-coloring, hence also for 3-SAT; that is considered highly unlike in the Computer Science community. Of course, the main open question is whether there exists a *polynomial time* algorithm for computing the Tutte polynomial of graphs of bounded clique-width. We discuss this issue in the last section.

In fact, our algorithms compute not only the Tutte polynomial, but the so-called U polynomial (see [12]), which is a stronger polynomial invariant. Moreover, we may skip the requirement of having a k-expression for G as an input in Theorem 1.2, if we do not care about an asymptotic behaviour in the exponent: Just to prove a subexponential upper bound we may use the approximation algorithm for clique-width by Oum and Seymour [13, 14].

Since our algorithms are quite complicated, for an illustration, we first present in Section 2 a simplified algorithm computing the number of forests in a cograph, that is, evaluating $T(G; 2, 1)$ for graphs of clique-width ≤ 2. (This is #P-hard on all graphs [8].) In Section 3 we extend the algorithm to the computation of the full Tutte polynomial on cographs. Finally, our main result, Theorem 1.2 is proved in details in the long version [6].

2 Forests in Cographs

The class of *cographs* is defined recursively as follows:

1. A single vertex is a cograph.
2. A disjoint union of two cographs is a cograph.
3. A complete union of two cographs is a cograph.

Here a *complete union* of two graphs $G \oplus H$ means the operation of taking a disjoint union $G \dot{\cup} H$, and adding all edges between $V(G)$ and $V(H)$. A cograph G can be represented by a tree, whose internal nodes correspond to operations 2) and 3) above, and whose leaves correspond to single vertices. We call such a tree an *expression* for G.

For example, all cliques are cographs, and the complement of a cograph is a cograph again. Cographs have long history of theoretical and algorithmic research. In particular, they are known to be exactly the graphs without induced paths on four vertices (P_4-free).

Let us call a *signature* a multiset of positive integers. The *size* $\|\boldsymbol{\alpha}\|$ of a signature $\boldsymbol{\alpha}$ is the sum of all elements in $\boldsymbol{\alpha}$, respecting repetition in the multiset. A signature $\boldsymbol{\alpha}$ of size n is represented by the *characteristic vector* $\boldsymbol{\alpha} = (a_1, a_2, \ldots, a_n)$, where there are $a_i \geq 0$ elements i in $\boldsymbol{\alpha}$, and $\sum_{i=1}^{n} i \cdot a_i = n$. (On the other hand, the *cardinality* of $\boldsymbol{\alpha}$ is $|\boldsymbol{\alpha}| = \sum_{i=1}^{n} a_i$, as usual.) An important fact we need is:

Recall that $\Theta(f)$ is a usual shortcut for all functions having the same asymptotic growth rate as f.

Lemma 2.1. *There are $2^{\Theta(\sqrt{n})}$ distinct signatures of size n.*

Proof. Each signature actually corresponds to a partition of n into an unordered sum of positive integers. It is well-known [10, Chapter 15] that there are $2^{\Theta(\sqrt{n})}$ of those. ∎

We call a *double-signature* a multiset of ordered pairs of non-negative integers, excluding the pair $(0, 0)$. The *size* $\|\boldsymbol{\beta}\|$ of a double-signature $\boldsymbol{\beta}$ is the sum of all $(x + y)$ for $(x, y) \in \boldsymbol{\beta}$, respecting repetition in the multiset. We, moreover, need to prove:

Lemma 2.2. *There are $\exp\left(\Theta(n^{2/3})\right)$ distinct double-signatures of size n.*

Lemma 2.2 is a particular case of Lemma 5.1, which is proved in [6].

Lemma 2.3. *A double-signature $\boldsymbol{\beta}$ of size n has at most $\exp\left(O(n^{2/3})\right)$ different submultisets (i.e. of different characteristic vectors).*

Proof. Just count all double-signatures of size $\leq n$. ∎

2.1 Forest Signature Table

Let us now consider a graph G and a forest $U \subset G$. The signature $\boldsymbol{\alpha}$ of U is the multiset of sizes of the connected components of U. (Obviously, $\boldsymbol{\alpha}$ has size $|V(G)|$ if U spans all the vertices.) We call a *(spanning) forest signature table* of the graph G a vector $\boldsymbol{T}$ (realized as an array $\boldsymbol{T}[\ldots]$); such that $\boldsymbol{T}$ records, for each signature $\boldsymbol{\alpha}$ of size $|V(G)|$, the number of spanning forests $U \subset G$ having signature $\boldsymbol{\alpha}$ (as $\boldsymbol{T}[\boldsymbol{\alpha}]$). For simplicity we usually skip the word "spanning" if it is clear from the context. We are going to compute the forest signature table of a cograph G recursively along the way G has been constructed. For that we describe two algorithms.

Let us denote by Σ_G the set of all signatures of size $|V(G)|$. It is important to keep in mind that signatures are considered as multisets, which concerns also set operations. For instance, a *multiset union* $\boldsymbol{\gamma} \uplus \boldsymbol{\delta}$ is obtained as the sum of the characteristic vectors of $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$, and a *multiset difference* $\boldsymbol{\gamma} \setminus \boldsymbol{\delta}$ is defined by the non-negative difference of those.

Algorithm 2.4. *Combining the spanning forest signature tables of graphs F and G into the one of the disjoint union $H = F \dot\cup G$.*

```
Input: Graphs $F, G$, and their forest signature tables $\boldsymbol{T}_F, \boldsymbol{T}_G$.
Output: The forest signature table $\boldsymbol{T}_H$ of $H = F \dot\cup G$.

create empty table $\boldsymbol{T}_H$ of forest signatures of size $|V(H)|$;
for all signatures $\boldsymbol{\alpha}_F \in \Sigma_F$, $\boldsymbol{\alpha}_G \in \Sigma_G$ do
   set $\boldsymbol{\alpha} = \boldsymbol{\alpha}_F \uplus \boldsymbol{\alpha}_G$ (a multiset union);
   add $\boldsymbol{T}_H[\boldsymbol{\alpha}] += \boldsymbol{T}_F[\boldsymbol{\alpha}_F] \cdot \boldsymbol{T}_G[\boldsymbol{\alpha}_G]$;
done.
```

The running time of this algorithm is proportional to the number of pairs of signatures (α_F, α_G), which is $\exp\big(O(n^{2/3})\big)$, where $n = |V(H)|$; this is due to Lemma 2.2 and the fact that we have the $O(\ )$ expression in the exponent.

The second algorithm is, on the other hand, much more complicated. It involves double-signatures in the following meaning: Consider a graph H with vertices partitioned into two parts $V(H) = V_1 \cup V_2$, and a forest $U \subset H$. The double-signature of U (wrt. V_1, V_2) is the multiset of pairs $\big(|V(C) \cap V_1|, |V(C) \cap V_2|\big)$ over all connected components C of U.

The idea behind the algorithm is to obtain the double-signatures (for $V_1 = V(F)$ and $V_2 = V(G)$) of the spanning forests in $H = F \oplus G$ from the signatures of the spanning forests in F and G. For every pair of forests $U_F \subset F$ and $U_G \subset G$, the algorithm iteratively counts the different ways in which each component of U_G can be joined to components of U_F. During the process, double signatures are needed to distinguish between former vertices of F and of G in already joined components. In fact, the algorithm works with pairs of signatures $\boldsymbol{\alpha}_F$ and $\boldsymbol{\alpha}_G$, that is, with whole classes of forests instead of particular forests. We also remark that a submultiset is considered among all possible selections of repeated elements, like if they were pairwise distinct.

Algorithm 2.5. *Combining the spanning forest signature tables of graphs F and G into the one of the complete union $H = F \oplus G$.*

```
Input: Graphs $F, G$, and their forest signature tables $\boldsymbol{T}_F, \boldsymbol{T}_G$.
Output: The forest signature table $\boldsymbol{T}_H$ of $H = F \oplus G$.

create empty table $\boldsymbol{T}_H$ of forest signatures of size $|V(H)|$;
for all signatures $\boldsymbol{\alpha}_F \in \Sigma_F$, $\boldsymbol{\alpha}_G \in \Sigma_G$ do
   set $z = |V(F)|$;
   create empty table $\boldsymbol{X}$ of forest double-signatures of size $z$;
      // Imagine particular forests $U_F \subset F$, $U_G \subset G$ of signature $\boldsymbol{\alpha}_F, \boldsymbol{\alpha}_G$,
      // and a selected component $C \subset U_G$ of size $c$.
   set $\boldsymbol{X}\big[$double-signature $\{(a,0) : a \in \boldsymbol{\alpha}_F\}\big] = 1$;
   for each $c \in \boldsymbol{\alpha}_G$ (with repetition) do
      create empty table $\boldsymbol{X}'$ of forest double-signatures of size $z + c$;
      for all double signatures $\boldsymbol{\beta}$ of size $z$ s.t. $\boldsymbol{X}[\boldsymbol{\beta}] > 0$ do
(†)        for all submultisets $\boldsymbol{\gamma} \subseteq \boldsymbol{\beta}$ (with repetition) do
            set $d_1 = \sum_{(x,y)\in\boldsymbol{\gamma}} x$, $d_2 = \sum_{(x,y)\in\boldsymbol{\gamma}} y$;
            set double-signature $\boldsymbol{\beta}' = (\boldsymbol{\beta} \setminus \boldsymbol{\gamma}) \uplus \{(d_1, d_2 + c)\}$;
(*)         add $\boldsymbol{X}'[\boldsymbol{\beta}'] += \boldsymbol{X}[\boldsymbol{\beta}] \cdot \prod_{(x,y)\in\boldsymbol{\gamma}} cx$;
         done
      done
      set $\boldsymbol{X} = \boldsymbol{X}'$, $z = z + c$;  dispose $\boldsymbol{X}'$;
   done
   for all double-signatures $\boldsymbol{\beta}$ of size $|V(H)|$ do
      set signature $\boldsymbol{\alpha}_0 = \{x + y : (x,y) \in \boldsymbol{\beta}\}$;
      add $\boldsymbol{T}_H[\boldsymbol{\alpha}_0] += \boldsymbol{X}[\boldsymbol{\beta}] \cdot \boldsymbol{T}_F[\boldsymbol{\alpha}_F] \cdot \boldsymbol{T}_G[\boldsymbol{\alpha}_G]$;
   done
done.
```

Proof of Algorithm 2.5. We now explain the algorithm, and show its correctness. It is better understandable if one imagines particular forests (representatives) $U_F \subset F$ and $U_G \subset G$ in the place of the signatures $\boldsymbol{\alpha}_F$ and $\boldsymbol{\alpha}_G$ chosen in the first `for` cycle. Then one may routinely verify that all subsequent computations depend only on the forest signatures $\boldsymbol{\alpha}_F$, $\boldsymbol{\alpha}_G$ (not on the particular forests), and hence it is correct to finally multiply the computed values in $\boldsymbol{X}$ by the numbers $\boldsymbol{T}_F[\boldsymbol{\alpha}_F] \cdot \boldsymbol{T}_G[\boldsymbol{\alpha}_G]$.

In the tables $\boldsymbol{X}, \boldsymbol{X}'$ we iteratively compute the numbers of all spanning forests in H that result by adding some edges between the forests U_F and U_G (stored by their double signatures). We consider an arbitrary order $C_1, C_2, \ldots, C_k$ on the connected components of U_G. For $i = 1, 2, \ldots, k$, we take the component C_i, and count all possible ways how to connect C_i by selected edges to a subset (†) of components of each of the previously constructed forests on $V(F \cup C_1 \cup \ldots \cup C_{i-1})$ which are recorded in the table $\boldsymbol{X}$. The other ends of those selected edges are considered only among vertices in $V(F)$. (Recall that the complete union $H = F \oplus G$ has added *all* edges between $V(F)$ and $V(C_i)$.) We then record (*) numbers of all the new forests on $V(F \cup C_1 \cup \ldots \cup C_i)$ in a new table $\boldsymbol{X}'$ that will play the role of $\boldsymbol{X}$ in the next iteration.

Saying precisely, after finishing iteration $i = 1, 2, \ldots, k$ described in the previous paragraph, each entry $\boldsymbol{X}'[\boldsymbol{\beta}]$ equals the number of all forests U' of signature $\boldsymbol{\beta}$ spanning $V(F \cup C_1 \cup \ldots \cup C_i)$ such that $U' \restriction V(F) = U_F$ and $U' \restriction V(G) = U_G \restriction C_1 \cup \ldots \cup C_i$. That follows easily by an induction from the previous arguments. At the end we count each spanning forest $U \subseteq H$ such that $U \restriction V(F) = U_F$ and $U \restriction V(G) = U_G$ exactly once. Finally, the double-signatures in the table $\boldsymbol{X}$ partition the vertices into $V(F)$ and $V(G)$, but that is no longer needed. So we "simplify" them – we record the resulting numbers only by the (single) forest signatures in the resulting table $\boldsymbol{T}_H$. ∎

2.2 Time Analysis

Lemma 2.6. *A modified implementation of Algorithm 2.5 runs in time* $\exp\big(O(n^{2/3})\big)$ *where* $n = |V(H)|$.

Proof. Since we have $O(\ )$ in the exponent, it is enough to verify that each of the `for` cycles in Algorithm 2.5 is iterated at most $\exp\big(O(n^{2/3})\big)$ times. That follows from Lemma 2.1 for the first cycle, and it is clear for the second cycle. For the third nested cycle it follows from Lemma 2.2.

A problem may occur in the fourth nested cycle '`for` all submultisets $\boldsymbol{\gamma} \subseteq \boldsymbol{\beta}$' if $\boldsymbol{\beta}$ consists, say, of $n/2$ copies of the element 2. Then there are up to $\exp\big(\Theta(n)\big)$ submultisets γ to consider. Fortunately, the results of the subsequent computation depend only on the characteristic vector of $\boldsymbol{\gamma}$. Hence it is enough to consider (much less of) pairwise different submultisets $\boldsymbol{\gamma} \subseteq \boldsymbol{\beta}$ (cf. Lemma 2.3), and then multiply the resulting number by all possible choices (combinations) of repeated elements of $\boldsymbol{\gamma}$ from $\boldsymbol{\beta}$. Formally, the program line (†) now reads

$$\texttt{for}\ \textit{all different submultisets}\ \boldsymbol{\gamma} \subseteq \boldsymbol{\beta}\ \texttt{do}\,,$$

and the line (*) reads

$$\texttt{add}\ \boldsymbol{X}'[\boldsymbol{\beta}'] \mathrel{+}= \boldsymbol{X}[\boldsymbol{\beta}] \cdot \prod_{(x,y)\in\boldsymbol{\gamma}} cx \cdot \prod_{(x,y)\in\langle\boldsymbol{\beta}\rangle} \binom{\mu_{\boldsymbol{\beta}}(x,y)}{\mu_{\boldsymbol{\gamma}}(x,y)}\,,$$

where $\langle\boldsymbol{\alpha}\rangle$ denotes the ordinary set formed by elements of a multiset $\boldsymbol{\alpha}$, and $\mu_{\boldsymbol{\alpha}} z$ is the repetition of an element z in $\boldsymbol{\alpha}$. The statement is proved. ∎

We remark that the improvement discussed in the proof of previous Lemma 2.6 have been fully incorporated in the subsequent algorithms.

Theorem 2.7. *The number of spanning forests in an n-vertex cograph can be computed in time* $\exp\big(O(n^{2/3})\big)$.

Proof. Consider a cograph G and a tree expression defining it. The forest signature table of a single vertex is trivial, and by Algorithms 2.4 and 2.5 (Lemma 2.6), the forest signature tables of a union or a complete union of two cographs can be computed in time claimed. Finally, knowing the forest signature table $\boldsymbol{T}$ of G, the number of all spanning forests of G is computed by adding up the entries of $\boldsymbol{T}$. ∎

3 The Tutte Polynomial of a Cograph

The Tutte polynomial can be defined in a number of equivalent ways. For our purposes, given a graph $G = (V, E)$ we define the Tutte polynomial as

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

where $r(F) = |V| - k(F)$ and $k(F)$ is the number of connected components of the spanning subgraph induced by the edge-subset F. It is clear that knowing $T(G; x, y)$ is the same as knowing, for every i and j, how many spanning subgraphs with the edge set F in G are there with $|F| = i$ and $k(F) = j$.

Consider a spanning subgraph $W \subset G$ determined on $V(W) = V(G)$ by an arbitrary subset $F \subset E(G)$, $F = E(W)$. The sizes of the connected components of W define a signature of size $|V(G)|$. In the *(spanning) subgraph signature table* $\boldsymbol{S}$ of G, for each signature $\boldsymbol{\alpha}$ of size $|V(G)|$ and each number of edges $f \in \{0, 1, 2, \ldots, |E(G)|\}$, we record the number $\boldsymbol{S}[\boldsymbol{\alpha}, f]$ of all spanning subgraphs of G having f edges and having component sizes according to the signature $\boldsymbol{\alpha}$. We shortly denote by $\boldsymbol{\gamma} \restriction_i$ the multiset formed by all the i-th coordinates (repetitions accounted for) of the elements of a double-signature $\boldsymbol{\gamma}$.

In order to prove Theorem 1.1 we need analogues of Algorithms 2.4 and 2.5 for computing subgraph signature tables. The algorithm for disjoint unions is again straightforward and we omit it; the one for complete unions comes next.

Algorithm 3.1. *A modification of Algorithm 2.5 for computing the (spanning) subgraph signature table of the complete union $H = F \oplus G$.*
Besides adding edge number as the second index to the signature tables, the only other major difference of this algorithm from Algorithm 2.5 is that the single line (*) is replaced with another `for` cycle calling a procedure `CellSel` of further Algorithm 3.2.

`Input:` Graphs F, G, and their subgraph signature tables $\boldsymbol{S}_F, \boldsymbol{S}_G$.
`Output:` The subgraph signature table $\boldsymbol{S}_H$ of $H = F \oplus G$.

```
create empty table $\boldsymbol{S}_H$ of subgraph signatures of size $|V(H)|$;
for all $\boldsymbol{\alpha}_F \in \Sigma_F$, and $e_F = 0, 1, \ldots, |E(F)|$ s.t. $\boldsymbol{S}_F[\boldsymbol{\alpha}_F, e_F] > 0$ do
    for all $\boldsymbol{\alpha}_G \in \Sigma_G$, and $e_G = 0, \ldots, |E(G)|$ s.t. $\boldsymbol{S}_G[\boldsymbol{\alpha}_G, e_G] > 0$ do
        set $z = |V(F)|$;
        create empty table $\boldsymbol{Y}$ of subgraph double-signatures of size $z$;
        set $\boldsymbol{Y}\big[$double-signature $\{(a, 0) : a \in \boldsymbol{\alpha}_F\}$, $e_F\big] = 1$;
        for each $c \in \boldsymbol{\alpha}_G$ (with repetition) do
```

```
            create empty table $\boldsymbol{Y}'$ of subgraph double-sign. of size $z+c$;
            for all $\boldsymbol{\beta}$ of size $z$, and $e$ s.t. $\boldsymbol{Y}[\boldsymbol{\beta}, e] > 0$ do
               for all different submultisets $\boldsymbol{\gamma} \subseteq \boldsymbol{\beta}$ do
                  set $r = \prod_{(x,y)\in\langle\boldsymbol{\beta}\rangle} \binom{\mu_{\boldsymbol{\beta}}(x,y)}{\mu_{\boldsymbol{\gamma}}(x,y)}$;
                  set $d_1 = \|\boldsymbol{\gamma}\restriction_1\| = \sum_{(x,y)\in\boldsymbol{\gamma}} x$, $d_2 = \|\boldsymbol{\gamma}\restriction_2\| = \sum_{(x,y)\in\boldsymbol{\gamma}} y$;
                  set double-signature $\boldsymbol{\beta}' = (\boldsymbol{\beta} \setminus \boldsymbol{\gamma}) \uplus \{(d_1, d_2 + c)\}$;
                  for $f = |\boldsymbol{\gamma}|, |\boldsymbol{\gamma}|+1, \ldots, c \cdot d_1$ do
                     set multiset $D = c \cdot (\boldsymbol{\gamma}\restriction_1) = \{cx : (x,y) \in \boldsymbol{\gamma}\}$;
                     call Algorithm 3.2: $p$ = CellSel($D$, $f$);
                     add $\boldsymbol{Y}'[\boldsymbol{\beta}', e+f]$ += $\boldsymbol{Y}[\boldsymbol{\beta}, e] \cdot r \cdot p$;
                  done
               done
            done
            set $\boldsymbol{Y} = \boldsymbol{Y}'$, $z = z + c$; dispose $\boldsymbol{Y}'$;
         done
         for all double-sign. $\boldsymbol{\beta}$ of size $|V(H)|$, and $f$, s.t. $\boldsymbol{Y}[\boldsymbol{\beta}, f] > 0$ do
            set signature $\boldsymbol{\alpha}_0 = \{x + y : (x,y) \in \boldsymbol{\beta}\}$;
            add $\boldsymbol{S}_H[\boldsymbol{\alpha}_0, f + e_G]$ += $\boldsymbol{Y}[\boldsymbol{\beta}, f] \cdot \boldsymbol{S}_F[\boldsymbol{\alpha}_F, e_F] \cdot \boldsymbol{S}_G[\boldsymbol{\alpha}_G, e_G]$;
         done
      done
   done.
```

Proof of Algorithm 3.1. This algorithm is similar to the improved version of Algorithm 2.5 (cf. Lemma 2.6), and so we only sketch the proof here. The main new difficulty lies in counting the different ways in which a connected component of c vertices in $\boldsymbol{\alpha}_G$ can be connected with f edges to the selected components of signatures $(x,y) \in \boldsymbol{\gamma}$. Recall that when counting forests we had no such difficulty, since we joined the component of $\boldsymbol{\alpha}_G$ to each component of $\boldsymbol{\gamma}$ with exactly one edge; thus we used exactly $f = |\boldsymbol{\gamma}|$ edges chosen in $\prod_{(x,y)\in\boldsymbol{\gamma}} cx$ different ways. The procedure 'CellSel(D, f)' counts this for spanning subgraphs, and we defer the explanation to Algorithm 3.2.

Finally, notice that the edge numbers in tables $\boldsymbol{Y}$, $\boldsymbol{Y}'$ do not account for the edges from $E(G)$, since we do not know how many edges has each one of the components of $\boldsymbol{\alpha}_G$. Those edges are summed up at the end, when obtaining the signatures for H from the double-signatures stored in $\boldsymbol{Y}$. ∎

Algorithm 3.2. *Computing the number of cellular selections: We are selecting ℓ elements from the union $C_1 \cup C_2 \cup \ldots \cup C_k$, where C_i for $i = 1, 2, \ldots, k$ are pairwise disjoint cells of sizes $d_i = |C_i|$, and we require that some element is selected from every cell.*

`Input:` A multiset $D = \{d_1, d_2, \ldots, d_k\}$ of cell sizes, and a number ℓ.

```
Output: The number CellSel$(D, \ell)$ of all such possible selections.
create table $\boldsymbol{u}[1..k][1..\ell]$, filled with $0$;
for $j = 1, 2, \dots, d_1$ do  set $\boldsymbol{u}[1][j] = \binom{d_1}{j}$;
set $z = d_1$;
for $i = 2, 3, \dots, k$ do
    add $z \mathrel{+}= d_i$;
    for $j = i, i+1, \dots, \min(\ell, z)$ do
        for $s = 1, 2, \dots, \min\bigl(j - (i-1),\, d_i\bigr)$ do
            add $\boldsymbol{u}[i][j] \mathrel{+}= \boldsymbol{u}[i-1][j-s] \cdot \binom{d_i}{s}$;
        done
    done
done
return $\boldsymbol{u}[k][\ell]$.
```

Proof of Algorithm 3.2. Let $u_{i,j} = \boldsymbol{u}[i][j]$ be the number of cellular selections of j elements chosen among the first i cells. These numbers satisfy the recurrence relation

$$u_{i,j} = \sum_{s=1}^{r} u_{i-1,j-s} \cdot \binom{d_i}{s}$$

where r is the maximum number of elements than can be selected from the i-th cell to obtain a total of j elements. Since the i-th cell has d_i elements available, and the $i-1$ previous cells contributed at least one element each to the resulting j elements, it follows that $r = \min\{j - (i-1), d_i\}$.

Algorithm 3.2 just applies the previous recurrence in a correct order, and avoids useless computations like with values of j too small or too large. It runs in $O(k\ell^2)$ steps. ∎

Proof of Theorem 1.1. As in Theorem 2.7, the subgraph signature table $\boldsymbol{S}$ of a cograph can be computed in time proportional to the number of all possible double-signatures of size n, i.e. in $\exp\bigl(O(n^{2/3})\bigr)$. Then, summing the entries of $\boldsymbol{S}$, we compute the numbers of spanning subgraphs with a given number of edges and a number of components. As we have remarked previously, these numbers give (efficiently) the Tutte polynomial. ∎

The U polynomial of an n-vertex graph G is defined in [12] as

$$U(G; \mathbf{x}, y) = \sum_{F \subseteq E} x_{n_1} \cdots x_{n_k} (y-1)^{|F| - r(F)},$$

where $n_1, \dots, n_k$ are the vertex sizes of the components of the spanning subgraph (V, F). If we let $x_1 = \dots = x_n = x - 1$ in the expression above, we recover the Tutte polynomial $T(G; x, y)$ up to a power of $x - 1$. It is clear that the subgraph signature table of a graph is precisely equivalent to the U polynomial, hence in the statement of Theorem 1.1 we can replace "U polynomial" for "Tutte polynomial".

4 Concluding remarks

We have shown that the Tutte and U polynomials can be computed in subexponential time for cographs, and more generally for graphs with bounded clique-width [6]. Such a result is very unlikely to hold for all graphs. Of course, the important question of whether the Tutte polynomial can be computed in polynomial time, or the problem is #P–hard even for graphs of bounded clique-width, remains open. (The U polynomial is obviously not computable in polynomial time due to its size.)

On the other hand, the *chromatic* polynomial for graphs of bounded clique-width can be computed in polynomial time (although not FPT). This follows by adapting the algorithm in [9] for computing the chromatic number, keeping track also of the number of r-colorings for $r = 1, \ldots, n$, where n is the number of vertices. To our knowledge, that is possibly the only currently known natural example of graph classes other than chordal graphs, where the chromatic polynomial can be computed in polynomial time, but the complexity of computing the Tutte polynomial is undecided.

References

1. G.E. Andrews, *The theory of partitions*, Cambridge U. Press, Cambridge, 1984.
2. A. Andrzejak, *An Algorithm for the Tutte Polynomials of Graphs of Bounded Treewidth*, Discrete Math. 190 (1998), 39–54.
3. B. Courcelle, J.A. Makowsky, U. Rotics, *Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width*, Theory Comput. Systems 33 (2000), 125–150.
4. B. Courcelle, S. Olariu, *Upper bounds to the clique width of graphs*, Discrete Appl. Math. 101 (2000), 77–114.
5. O. Giménez, M. Noy, *On the complexity of computing the Tutte polynomial of bicircular matroids*, Combin. Probab. Computing, to appear.
6. O. Giménez, P. Hliněný, M. Noy, *Computing the Tutte Polynomial on graphs of Bounded Clique-Width*, manuscript, 2005.
7. P. Hliněný, *The Tutte Polynomial for Matroids of Bounded Branch-Width*, Combin. Probab. Computing, to appear (2005).
8. F. Jaeger, D.L. Vertigan, D.J.A. Welsh, *On the Computational Complexity of the Jones and Tutte Polynomials*, Math. Proc. Camb. Phil. Soc. 108 (1990), 35–53.
9. D. Kobler, U. Rotics, *Edge dominating set and colorings on graphs with fixed clique-width*, Discrete Applied Math. 126 (2003), 197–221.
10. J.H. van Lint, R.M. Wilson, *A Course in Combinatorics*, Cambridge University Press, Cambridge, 1992.
11. S.D. Noble, *Evaluating the Tutte Polynomial for Graphs of Bounded Tree-Width*, Combin. Probab. Computing 7 (1998), 307–321.
12. S.D. Noble, D.J.A. Welsh, *A weighted graph polynomial from chromatic invariants of knots*, Ann. Inst. Fourier (Grenoble) 49 (1999), 1057–1087.
13. Sang-Il Oum, P.D. Seymour, *Approximating Clique-width and Branch-width*, submitted, 2004.
14. Sang-Il Oum, *Approximating Rank-width and Clique-width Quickly*, In: WG 2005, Proccedings, Lecture Notes in Computer Science, to appear (2005).