ELSEVIER

Theoretical Computer Science 281 (2002) 291–309

Theoretical Computer Science

www.elsevier.com/locate/tcs

Counting H-colorings of partial k-trees ☆

Josep Díaz*, Maria Serna, Dimitrios M. Thilikos

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Campus Nord-Mòdul C6, c/Jordi Girona Salgado 1-3, 08034, Barcelona, Spain

Happy birthday Maurice!

Abstract

The problem of counting all H-colorings of a graph G with n vertices is considered. While the problem is, in general, #P-complete, we give linear time algorithms that solve the main variants of this problem when the input graph G is a k-tree or, in the case where G is directed, when the underlying graph of G is a k-tree. Our algorithms remain polynomial even in the case where $k = O(\log n)$ or in the case where the size of H is $O(n)$. Our results are easy to implement and imply the existence of polynomial time algorithms for a series of problems on partial k-trees such as core checking and chromatic polynomial computation.

Keywords: Graph homomorphism; Counting problems; Treewidth

1. Introduction

In this paper we consider a series of counting problems associated with various versions of homomorphisms from a graph G to a fixed graph H. Given a graph $H=(G(H),E(V))$, with loops but without multiple edges, for any graph $G=(V(G), E(G))$ an *homomorphism* of G to H is a map $\theta: V(G) \to V(H)$ with the property that $\{v,w\} \in E(G) \Rightarrow \{\theta(v),\theta(w)\} \in E(H)$. We will call such a homomorphism a *H-coloring* of G (for an example, see Fig. 1). For H fixed, the *H-coloring problem* asks if there is

☆ The work of all the authors was supported by the IST Program of the EU under contract number IST-99-14186 (ALCOM-FT) and by the Spanish CICYT TIC-2000-1970-CE. The work of the last author was partially supported by the Ministry of Education and Culture of Spain, Grant number MEC-DGES SB98 0K148809.

* Corresponding author.

E-mail addresses: diaz@lsi.upc.es (J. Diocute;az), mjserna@lsi.upc.es (M. Serna), sedthilk@lsi.upc.es (D. M. Thilikos).

0304-3975/02/$ - see front matter
PII: S0304-3975(02)00017-8

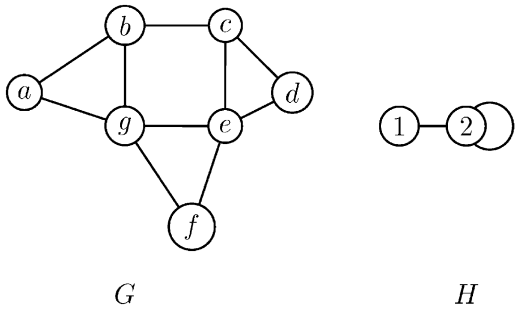

Fig. 1. An example of an H-coloring of G is the mapping $\theta(a)=\theta(d)=\theta(f)=1$ and $\theta(b)=\theta(c)=\theta(e)=2$.

an H-coloring of G. If H is bipartite or it has a loop, then the H-coloring problem can be trivially solved in polynomial time, but in the case that H is not-bipartite and loopless the problem is known to be NP-complete [24]. See also [22] for the complexity of the same problem for bounded degree graphs.

An application of the H-coloring problem is illustrated by the following example taken from [20]. Suppose that we have a graph whose vertices represent jobs to be processed and whose edges represent communication demands between two jobs. Let also H be a computer network where its vertices represent the processors and its edges correspond to communication links between them. We would like to assign the jobs to the processors so that all the communication demands between jobs are satisfied. This question can be modeled by the problem of asking whether there exists a homomorphism from G to H, the *H-coloring* problem.

An earlier version of the H-coloring problem is the *exact H-coloring problem*: given a graph G, decide if there is an H-coloring θ of G such that $\theta(V(G))=V(H)$ and $\theta(E(G))=E(H)$. This problem is known to be NP-complete, even for the case where H is a triangle [30]. We also consider the intermediate problem of asking whether there is an H-coloring θ of G such that $\theta(V(G))=V(H)$ and we call it *vertex exact H-coloring*.

Given a graph G, let $\mathscr{H}(G,H)$ denote the set of all H-colorings of G, and let $\mathscr{E}(G,H)$ denote the set of all exact H-colorings of G. Also, we denote as $\mathscr{V}(G,H)$ the set of all vertex H-colorings of G. For a fixed graph H, denote by *#H-coloring* the problem of, given G as input, computing $|\mathscr{H}(G,H)|$. Also, denote by *#EH-coloring* the problem of, given as input G, computing $|\mathscr{E}(G,H)|$ and by *#VEH-coloring* the problem of, given as input G, computing $|\mathscr{V}(G,H)|$. Dyer and Greehill [18] have proved that unless every connected component of H is an isolated vertex without a loop, a complete graph with all loops present or a complete unlooped bipartite graph, the #H-coloring problem is #P-complete, even when G has bounded degree. It is easy to verify that the same result hold for the #*EH*-coloring (#*VEH*-coloring) as the knowledge of $\mathscr{E}(G,H')$ ($\mathscr{V}(G,H')$) for any (induced) subgraph H' of H implies the knowledge of $\mathscr{H}(G,H)$. For further negative results on the approximability of the #H-coloring see [12] and [17]. For positive and negative complexity results concerning special cases of the #H-coloring problem, depending on restrictions on both H and G, see [19,16].

The notion of *treewidth* appears to play a central role in many areas of graph theory. Roughly, a graph has small treewidth if it can be constructed by assembling small graphs together in a tree structure, namely a tree decomposition with small width (see Section 2 for the formal definitions). It was first defined by Robertson and Seymour in [36] and served as one of the cornerstones of their lengthy proof of the Wagner conjecture, known now as the Graph Minors Theorem (for a survey see [37]). Treewidth appears to have interesting applications in algorithmic graph theory. In particular, a wide range of, otherwise intractable, combinatorial problems are polynomially, even linearly, solvable when restricted to graphs with bounded treewidth. Alternatively, graphs with bounded treewidth are called *partial k-trees*. Several general results have been developed for proving, constructively, the existence of such algorithms. Among them, we mention the work of Courcelle in [13] where the existence of polynomial algorithms is guaranted by the expressability of the corresponding problem by Monadic second order formulas (see also [14]). Similar results have been developed in [15] for solving counting problems for partial k-trees. As a consequence of these results, it is possible to contruct a polynomial time algorithm solving the $\#H$-coloring problem for partial k-trees, when k and the size of H are fixed constants. Unfortunately, the general results of Courcelle, do not provide implementable algorithms because of the big hidden constants in their complexity. Alternatively, the attention of many researchers was directed to the fast development and easy to implement "tailor made" algorithms for problems of specific interest. The standard methodology to get such solutions consists of a two step procedure: First to find a tree decomposition of the input graph, with treewidth bounded by a constant, although possibly not optimal, and second to use some kind of *dynamic programming* taking advantage of the bounded treewidth decomposition of the graph, to get a solution (for a survey, see [8,3]). In particular, such a dynamic programing technique, solving—among others—the H-coloring problem for partial k-trees, has been developed in [39]. For the first canonical step, Bodlaender in [7] presented, for any k, a linear time algorithm that, given a graph G, checks whether G is a partial k-tree and, if so, outputs a minimum width tree decomposition. As this algorithm appears extremely hard to be implemented (see also [9]), the first canonical step can be carried out by one of the previously developed algorithms that, given a graph with treewidth $\leqslant k$, are able to output a tree decomposition of G with width bounded by ck where c is a constant independent of k [34,28,31]. In particular, the deterministic algorithm by Reed [34] runs in $O(n \log n)$ time and outputs a tree decomposition with width $\leqslant 4k$. Moreover, its implementation is easier than the algorithm in [7]. Consequently, we will assume that any graph with bounded treewidth in this paper is always accompanied with a tree decomposition of bounded width.

In this paper we present a polynomial time algorithm for the counting problems $\#H$-coloring, $\#EH$-coloring, and $\#VEH$-coloring, for the case that the input graph G has small treewidth. Although our methodology follows the two cannonical steps, we shall remark that we present an easy to be implemented algorithm for the second step, assuming the aforementioned results on the existence of fast and implementable algorithms for obtaining a bounded treewidth decomposition for a partial k-tree G. Moreover, our algorithms remain polynomial even when there are no restrictions on

the size of H. Finally, we stress out that our results are the first positive results on counting H-colorings for the case where H is generic and only G is restricted.

All through the paper, we will consider as parameters the width of the given tree decomposition of G and the number of vertices and edges of H and we will denote them by k, h, and e, respectively. Moreover, we will consider the size of G as a variable and we will denote it as n. Finally, we will make explicit any dependence of variables and parameters. Therefore, in any notation $O(f(k,h,e,n))$ the hidden constant behind the "O"-notation will be independent of both variables and parameters.

In Section 2 we review the basic definitions and some preliminary results on treewidth. In Section 3 we present the algorithm solving the $\#H$-coloring problem. In Section 4 we present two variants of the algorithm of Section 3, for solving the $\#VEH$-coloring and the $\#EH$-coloring problem, respectively. In the final section, we present some extensions and consequences of our previous results for a series of problems on graphs with bounded treewidth.

2. Definitions and basic results

Robertson and Seymour [35,38] introduced the notion of treewidth.

Definition 2.1. A *tree decomposition* of a graph G is a pair (X,U) where U is a tree whose vertices we will call *nodes* and $X=(\{X_i \mid i\in V(U)\})$ is a collection of subsets of $V(G)$ such that
(1) $\bigcup_{i\in V(U)} X_i = V(G)$,
(2) for each edge $\{v,w\}\in E(G)$, there is an $i\in V(U)$ such that $v,w\in X_i$, and
(3) for each $v\in V(G)$ the set of nodes $\{i \mid v\in X_i\}$ forms a subtree of U.
The *width* of a tree decomposition $(\{X_i \mid i\in V(U)\},U)$ equals $\max_{i\in V(U)}\{|X_i|-1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

Computing treewidth is an NP-complete problem [4]. In Fig. 2, we present an optimal treewidth decomposition of the graph G in Fig. 1.

The following extensions of tree decomposition were introduced by Bodlaender and Kloks [7,9,27] (see also [3,7]).

Definition 2.2. A *rooted* tree decomposition is a triple $D=(X,U,r)$ in which U is a tree rooted at r and (X,U) is a tree decomposition.

Definition 2.3. Let $D=(X,U,r)$ be a rooted tree decomposition of a graph G where $X=\{X_i \mid i\in V(U)\}$. D is called a *nice* tree decomposition if the following are satisfied:
(1) Every node of U has at most two children,
(2) if a node i has two children j,h then $X_i=X_j=X_h$,
(3) if a node i has one child, then either $|X_i|=|X_j|+1$ and $X_j\subset X_i$ or $|X_i|=|X_j|-1$ and $X_i\subset X_j$.

The following lemma is taken from [9] and [27].

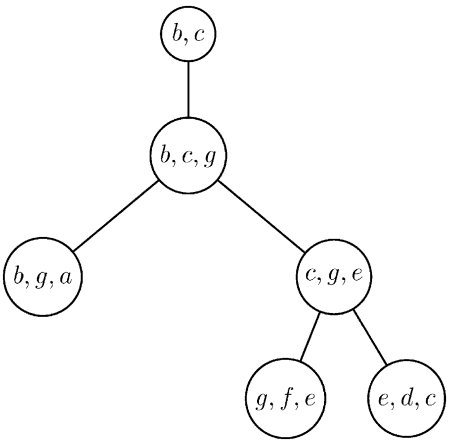

Fig. 2. A tree decomposition of G in Fig. 1.

Lemma 2.1. *For any constant $k \geqslant 1$, given a tree decomposition of a graph G of width $\leqslant k$ and $O(n)$ nodes, where n is the number of vertices in G, there exists an $O(n)$ algorithm that constructs a nice tree decomposition of G with width $\leqslant k$ and with at most $O(n)$ nodes.*

In Fig. 3 we present a nice tree decomposition for the graph G in Fig. 1. It has been constructed from the tree decomposition presented in Fig. 2.

We observe that a nice tree decomposition $D=(\{X_p \mid p \in V(U)\}, U, r)$ contains nodes of the following four possible types. A node $p \in V(U)$ is called:

Start, if p is a leaf,

Join, if p has two children q_i, $i=1,2$,

Forget, if p has only one child q and $|X_p| = |X_q| - 1$,

Introduce, if p has only one child q and $|X_p| = |X_q| + 1$.

Notice that for every start node we may assume that $|X_p| = 1$: the effect of *start* nodes with $|X_p| > 2$ can be obtained by using a *start* node with a set containing one vertex, and then $|X_p| - 1$ *introduce* nodes, which add all the other vertices. For the purposes of this paper we will assume that the root r is a *forget* node with $X_r = \emptyset$.

Taking into account Lemma 2.1, we will assume that any partial k-tree is given along with a nice tree decomposition of it.

Let $D=(X,U,r)$ be a nice tree decomposition of a graph G. For each node p of D, let U_p be the subtree of U, rooted at node p. We set $V_p = \bigcup_{v \in V(U_p)} X_v$ and, for any $p \in V(U)$, we define $G_p = G[V_p]$. Notice that $G_r = G$.

3. An algorithm for counting the number of H-colorings

Let H be a fixed graph. Given an input graph G together with a nice tree decomposition $D=(X,U,r)$ of it, we wish to compute $|\mathscr{H}(G,H)|$, in other words, to solve the $\#H$-coloring problem for G.

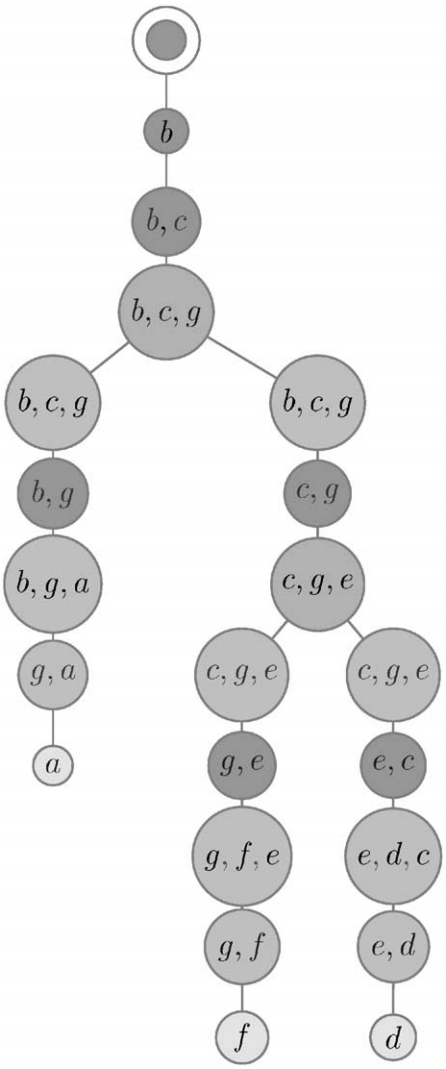

Fig. 3. A nice tree decomposition of G.

The strategy will be a dynamic programming approach. To optimize the space used in the counting part, we will preprocess the tree. The preprocessing will produce an ordering of the nodes in the nice tree decomposition.

Definition 3.1. Let T be a rooted binary tree T and b the number of vertices with degree two in T. An ordering $u_1,\ldots,u_s$ of $V(T)$ is *stingy* if the following conditions hold.

(1) u_1 is a leaf and u_n is the root.

(2) For any node u_i, its parent appears at a position $j>i$.
(3) For any j the number of nodes in the set $\{u_1,\ldots,u_j\}$ whose parent appears at a position $k>j$ is at most $\log b$.

Given a nice tree decomposition $D=(X,U,r)$ it is straightforward to compute a stingy ordering of the vertices of U.

Lemma 3.1. *Given a rooted binary tree T a proper ordering of the l nodes in $V(T)$ can be computed using $O(n\log n)$ space in $O(n)$ steps.*

The algorithm will do a *traversal* of D following a stingy sequence, computing and tabulating for each $p\in V(U)$ the appropriate information. Before describing it, we need some additional definitions.

Given a nice tree decomposition $D=(X,U,r)$ of a graph G, for each $p\in V(U)$, let $F_p=\{\varphi : X_p \to V(H)\}$. Notice that if D has width k, then for any $p\in V(U)$, $|F_p|\leqslant h^{k+1}$. Moreover, for the root we have $F_r=\{\emptyset\}$.

The table associated to a node $p\in V(U)$ will have an entry for each $\varphi\in F_p$, holding the value $I_p(\varphi)=|\{\theta\in\mathscr{H}(G_p,H)\mid \theta|_{X_p}=\varphi\}|$ (in general, if θ is a function, we denote $\theta|_S=\{(v,a)\in\theta\mid v\in S\}$). As we always have $\theta|_\emptyset=\emptyset$, we get $|\mathscr{H}(G,H)|=I_r(\emptyset)$.

The algorithm count-H, given in Fig. 4, takes as input G, D and a stingy sequence $S=(p_1,\ldots,p_s)$ of $V(U)$ and outputs the value $I_r(\emptyset)$.

Theorem 3.1. *Given a graph G with $n=|V(G)|$, a nice tree decomposition $D=(X,U,r)$ of G with width k, and a stingy ordering of the nodes in D, then, the algorithm in Fig. 4 computes the number of H-colorings of G in $O(nh^{k+1}\min\{k,h\})$ steps using $O(h^{k+1}\log n)$ additional space, where $h=|V(H)|$.*

Proof. Let us first prove the correctness of the proposed algorithm. The algorithm starts by computing the table for the first leaf, a *start* node, in the stingy sequence. Notice that by condition 2 in Definition 3.1, whenever we process a node $p\in V(U)$ its descendants have been already processed and the associated information stored. So, we have only to show that, for each type of node, our algorithm computes the corresponding values using the correct information stored in the tables of its descendants.

When X_p is a *start* node the computation is correct as G_p is formed by an unique vertex.

If p is an *introduce* node, with $\{v\}=X_p-X_q$, the graph G_p is obtained from G_q adding the new vertex v and the edges in the set $S_q=\{(u,v)\mid u\in S_q\}$, therefore only those extensions that preserve the H-coloring condition survive and therefore counted.

If p is a *forget* node, $X_p=X_q-\{v\}$, then $G_p=G_q$, and therefore, for each $\varphi\in F_p$, we only have to accumulate the number of colorings counted for any extension of φ that is a valid coloring of X_q.

When the computation arrives to a *join* node p, both of its children q_1, q_2 must have been already tabulated. Notice that the graph G_p is obtained as the union of the graphs G_{q_1} and G_{q_2}. Recall that G_{q_1} and G_{q_2} have no edges in common out of those in $X_p=X_{q_1}=X_{q_2}$. In such a case any H-coloring of G_p is the combination of an

Algorithm count-H(G,H,D,S)
$D=(X,U,r)$ is a nice tree decomposition of G with width k
$S=(p_1,\ldots,p_s)$ is a stingy ordering of $V(U)$

1. Set $i=1$.
2. Set $p=p_i$.
3. If p is a *start* node with $X_p=\{v\}$ then for all $a\in V(H)$ set $I_p((v,a))=1$.
4. If p is an *introduce* node then
- let q be its unique child and let v be the unique vertex in X_p-X_q
- set $S_q=\{u\in X_q \mid \{u,v\}\in E(G_p)\}$
- for all $\varphi\in F_q$ and $a\in V(H)$,
  if $\forall_{u\in S_q}\{\varphi(u),a\}\in E(H)$
  then set $I_p(\varphi\cup\{(v,a)\})=I_q(\varphi)$
  else set $I_p(\varphi\cup\{(v,a)\})=0$
- erase the information on node q.

5. If p is a *forget* node then
- let q be its unique child and v the unique vertex in X_q-X_p
- for all $\varphi\in F_p$, set $I_p(\varphi)=\sum_{a\in V(H)} I_q(\varphi\cup\{(v,a)\})$
- erase the information on node q.

6. If p is a *join* node with children q_1 and q_2 then
- for all $\varphi\in F_p$, set $I_p(\varphi)=I_{q_1}(\varphi)\cdot I_{q_2}(\varphi)$
- erase the information on nodes q_1 and q_2.

7. If $i<s$, set $i=i+1$ and goto step **2**.
8. Return $I_r(\emptyset)$.

Fig. 4. An algorithm for counting H-colorings.

H-coloring of G_{q_1} with an H-coloring of G_{q_2} with the same images on the vertices of X_p.

When the algorithm arrives to the last node, the root, the only value in its table will be the number of H-colorings of $G_r=G$.

Let us analyze now the algorithm's complexity. For any $p\in V(U)$ the number of elements in F_p is at most h^{k+1}. Moreover, $|V(U)|=\mathrm{O}(n)$. Therefore, the internal operations computing one entry for the cases of a *start*, *forget*, or *join* node, cost time $\mathrm{O}(h)$. In the case, of an *introduce* node, we can implement the computation either in $\mathrm{O}(h)$ or $\mathrm{O}(k)$ steps depending on whether we represent the set S_q by the set of vertices of G or the set of their images in H. Therefore, the time complexity follows.

To compute an upper bound on the required space, we have to take into account that the number of *join* nodes in a nice tree decomposition is at most n. Therefore, condition 3 in Definition 3.1 guarantees that the number of tables maintained at each step is at most $\mathrm{O}(\log n)$. As the size of a table is at most h^{k+1}, the space complexity follows. □

Notice that if h is constant, we can allow $k=O(\log n)$ and still have polynomial time. Moreover, for constant treewidth k, we can allow $h=O(n)$ and still have a polynomial time algorithm. As an example of this we have the following result, which is based on the fact that the outerplanar graphs have treewidth $\leqslant 2$.

Corollary 3.1. *There exists an algorithm that outputs the number of all the homomorphisms between any pair of outerplanar graphs in* $O(n^4)$ *steps and between any pair of forests in* $O(n^3)$ *steps.*

Taking in mind the algorithm of Bodlaender in [7] we have the following.

Corollary 3.2. *For any constant* k *there exists a function* f *and an algorithm that, given a graph* G *of treewidth bounded by* k, *computes* $|\mathscr{H}(G,H)|$ *in* $O((h^{k+1}\min\{k,h\}+f(k))n)$ *steps.*

Notice that, for constant k and h, our algorithm works in linear time.

4. Counting the number of (vertex) exact H-colorings

Let H be a fixed graph. Given a graph G together with a nice tree decomposition $D=(X,U,r)$ of G with width k and a stingy sequence of U, we wish to count $|\mathscr{V}(G,H)|$ and $|\mathscr{E}(G,H)|$, i.e. to solve the *#VEH*-coloring and the *#EH*-coloring problems.

The strategy will be similar as in the previous algorithm, a dynamic programming guided by a stingy sequence. The main modification will be the information stored in the tables.

4.1. The #VEH-coloring

The table associated to a node $p\in V(U)$ will have an entry for each pair (φ,R), where $\varphi\in F_p$ and $R\subseteq V(H)$, holding the value

$$I_p(\varphi,R)=|\{\theta\in\mathscr{V}(G_p,H)\mid\theta|_{X_p}=\varphi \text{ and } \theta(V(G_p))=R\}|.$$

Therefore, $|\mathscr{V}(G,H)|=I_r(\emptyset,V(H))$.

The algorithm count-exact-V-H, given in Fig. 5, takes as input G, D and a stingy sequence $S=(p_1,\ldots,p_s)$ of $V(U)$ and outputs the value $I_r(\emptyset,V(H))$.

Theorem 4.1. *Given a graph* G *with* $n=|V(G)|$, *a nice tree decomposition* $D=(X,U,r)$ *of* G *with width* k, *and a stingy ordering of the nodes in* D, *then, the algorithm in Fig.* 5 *computes the number of vertex exact* H*-colorings of* G *in* $O(nh^{k+1}2^h\max\{k,2^{2h}\})$ *time using* $O(h^{k+1}2^h\log n)$ *additional space, where* $h=|V(H)|$.

Proof. As in the proof of Theorem 3.1, we have to prove the correctness for each type of node. The arguments are similar, we only have to take into account not only

Algorithm count-exact-V-H (G,H,D,S)
$D=(X,U,r)$ is a nice tree decomposition of G with width k
$S=(p_1,\ldots,p_s)$ is a stingy ordering of $V(U)$

1. Set $i=1$.
2. Set $p=p_i$.
3. If p is a *start node*, with $X_p=\{v\}$ then
- for all $R\subseteq V(H)$ and $a\in V(H)$,
  if $R=\{a\}$
  then set $I_p((v,a),R)=1$,
  else $I_p((v,a),R)=0$.

4. If p is a *introduce* node then
- let q be its unique child and let v the unique vertex in X_p-X_q
- set $S_q=\{u\in X_q \mid \{u,v\}\in E(G_p)\}$
- for all $\varphi\in F_q$ and $R\subseteq V(H)$ and $a\in V(H)$,
  if $a\in R$ and $\forall u\in S_q[(\varphi(u),a)\in E(H)]$
  then set $I_p(\varphi\cup\{(v,a)\},R)=(I_q(\varphi,R)+I_q(\varphi,R\setminus\{a\}))$,
  else set $I_p(\varphi\cup\{(v,a)\},R)=0$
- erase the information on node q.

5. If p is a *forget* node then
- let q be its unique child and v be the unique vertex in X_q-X_p
- for all $\varphi\in F_q$ and $R\subseteq V(H)$, set $I_p(\varphi,R)=\sum_{a\in V(H)} I_q(\varphi\cup\{(v,a)\},R)$
- erase the information on node q.

6. If p is a *join* node with children q_1 and q_2 then
- for all $\varphi\in F_p$ and $R\subseteq V(H)$,
  - set $I_p(\varphi,R)=0$
    - for all R_1 and R_2 with $R_1\cup R_2=R$ set $I_p(\varphi,R)=I_p(\varphi,R)+I_{q_1}(\varphi,R_1)\cdot I_{q_2}(\varphi,R_2)$
- erase the information on nodes q_1 and q_2.

7. If $i<s$, set $i=i+1$ and goto step **2**.
8. return $I_r(\emptyset,V(H))$.

Fig. 5. An algorithm for counting vertex exact H-colorings.

the difference between G_p and the subgraphs associated to its descendants, but also the ways in which a given subset of vertices of H can be covered.

The easiest cases are the *start* and *forget* nodes. A *start* node has no descendants, and its unique associated vertex can only cover one vertex of H. In a *forget* node there is no additional information and, therefore, the set of covered vertex of H will remain.

If p is an *introduce* node, with $\{v\}=X_p-X_q$, the graph G_p is obtained from G_q by adding v and the corresponding edges. The only surviving mapping are those that preserve the H-coloring condition. Furthermore, when we determine the image of v

three possible cases arise: the color do not appear in the set R of covered colors, it is new and was not present in the set covered by node q, or it is old and appears in the set covered by node q. In the first case the total value is zero, otherwise we have to add all the colorings covering R from the two different possibilities.

When the computation arrives to a *join* node p, both of its children q_1, q_2 must have been already tabulated. The restriction to keep the coloring condition is the same; from the cover part we have to consider all the ways to split a set $R \subseteq V(H)$ into two partial covers one arising from each child. Note that, in the splitting, we can have $R_1 \cap R_2 \neq \emptyset$.

Let us analyze the complexity. For any $p \in V(U)$, the number of elements in F_p is at most h^{k+1}, and the number of possible subsets of $|V(H)|$ is 2^h. Moreover, $|V(U)| = O(n)$. The most costly cases are the *introduce* node, which needs $O(k)$ steps for each entry and a *join* node which needs $O(2^{2h})$ steps for each entry. Therefore, the time complexity follows. The additional space bound follows from the table size and the fact that the stingy sequence restricts the necessary space resources on each step to $O(\log n)$ tables. □

Corollary 4.1. *For any constant k there exists a function f and an algorithm that, given a graph G of treewidth bounded by k, computes $|\mathscr{V}(G,H)|$ in $O((h^{k+1}2^h \max\{k, 2^{2h}\} + f(k))n)$ time, where $h = |V(H)|$.*

4.2. The #EH-coloring

The table associated to a node $p \in V(U)$ will have an entry for each pair (φ, J), where $\varphi \in F_p$ and J is a subgraph of H, holding the value

$$I_p(\varphi, J) = |\{\theta \in \mathscr{E}(G_p, H) \mid \theta|_{X_p} = \varphi \text{ and } \theta(V(G_p)) = V(J) \text{ and } \theta(E(G_p)) = E(J)\}|.$$

Therefore, $|\mathscr{E}(G,H)| = I_r(\emptyset, H)$.

The algorithm count-exact-H, given in Fig. 6, takes as input G, D and a stingy sequence $S = (p_1, \ldots, p_s)$ of $V(U)$ and outputs the value $I_r(\emptyset, H)$. We will use $\mathscr{S}(H)$ to denote the set of subgraphs of H.

Theorem 4.2. *Given a graph G with $n = |V(G)|$, a nice tree decomposition $D = (X, U, r)$ of G with width k, and a stingy ordering of the nodes in D. Then the algorithm in Fig. 6 computes the number of H-colorings of G in $O(nh^{k+1}2^{h+e} \max\{k, 2^{2(h+e)}\})$ time using $O(h^{k+1}2^{h+e} \log n)$ additional space, where $h = |V(H)|$ and $e = |E(H)|$.*

Proof. The arguments are similar to those in the proof of Theorem 4.1. We have to further take into account the ways in which a given subgraph of H can be covered.

The easiest cases are the *start* and *forget* nodes. For the first, only one vertex of H can be covered, and the last will not cover any additional vertex or edge.

If p is an *introduce* node, with $\{v\} = X_p - X_q$, as before only those mapping extensions that preserve the H-coloring condition must survive. Furthermore, when we

Algorithm count-exact-H(G,H,D,S)
$D=(X,U,r)$ is a nice tree decomposition of G with width k
$S=(p_1,\ldots,p_s)$ is a stingy ordering of $V(U)$

1. Set $i=1$.
2. Set $p=p_i$.
3. If p is a *start node*, with $X_p=\{v\}$ then
- for all $J\in\mathscr{S}(H)$ and $a\in V(H)$,
  if $V(J)=\{a\}$
  then set $I_p((v,a),J)=1$
  else $I_p((v,a),J)=0$.

4. If p is an *introduce* node then
- let q be its unique child and let v be the unique vertex in X_p-X_q,
- let $S_q:=\{u\in X_q \mid \{u,v\}\in E(G_p)\}$
- for all $\varphi\in F_q$ and $J\in\mathscr{S}(H)$ and $a\in V(H)$,
  if $\{\{a,\varphi(u)\}\mid u\in S_q\}\subseteq E(J)$
  then
  – if $\forall\{a,\varphi(v)\}\in E(J)\exists u\in X_q$: $\varphi(u)=a$
    then set $I_p(\varphi,J)=I_q(\varphi|_{X_q},J-\{\varphi(v)\})$
    else set $I_p(\varphi,J)=0$
  – for all $A\subseteq\{\{a,\varphi(u)\}\mid u\in S_q\}$ with $\forall\{a,b\}\in E(J)-A\ \exists u\in X_q$: $\varphi(u)=b$,

$$\text{set } I_p(\varphi,J):=I_p(\varphi,J)+I_q(\varphi|_{X_q},(V(J),E(J)-A))$$

  else $I_p(\varphi,J):=0$
- erase the information on node q.

5. If p is a *forget* node then
- let q be its unique child and let v be the unique node in X_q-X_p
- for all $\varphi\in F_p$ and $J\in\mathscr{S}(H)$, set $I_p(\varphi,J)=\sum_{a\in V(J)}I_q(\varphi\cup\{(v,a)\},J)$
- erase the information on node q.

6. If p is a *join* node with children q_1 and q_2 then
- for all $\varphi\in F_p$ and $J\in\mathscr{S}(H)$,
  – set $I_p(\varphi,J):=0$
  – for all $J_1,J_2\in\mathscr{S}(H)$ with $J_1\cup J_2=J$ and $\varphi(X_p)\subseteq V(J_1)\cap V(J_2)$,
    set $I_p(\varphi,J)=I_p(\varphi,J)+I_{q_1}(\varphi,J_1)\cdot I_{q_2}(\varphi,J_2)$
- erase the information on nodes q_1 and q_2.

7. If $i<s$, set $i=i+1$ and goto step **2**.
8. Return $I_r(\emptyset,H)$.

Fig. 6. An algorithm for counting exact H-colorings.

determine the image of v, it may happen that either the colors of one of the images of the introduced edges are not present in the subgraph or that the colors of all the images are present. In the first case, the number of colorings is zero. We have two types of decomposition: from subgraphs without the image of v or from subgraphs with

the image of the same vertex set and less edges. In both cases we must insure that the edges not present are covered by the actual mapping.

When the computation arrives to a *join* node p, we have to consider all the ways to split a subgraph of H into two subgraphs, each covered by a child. Note that in the splitting we can have two subgraphs with non-empty intersection, but both must contain the common image of X_p.

Let us analyze the complexity. For any $p \in V(U)$, the number of elements in F_p is at most h^{k+1}, and the number of possible subgraphs of H, that is 2^{h+e}. Moreover, $|V(U)| = O(n)$. Again, the most costly cases are the *introduce* node, which needs $O(k + 2^e)$ steps and the *join* node which needs $2^{2(h+e)}$ time. The time complexity follows. The additional space bound follows from the table size and the fact that the stingy sequence restricts the necessary space resources on each step to $O(\log n)$ tables. □

Corollary 4.2. *For any constant k there exists an integer m and an algorithm that, given a graph G of treewidth bounded by k, computes $|\mathscr{E}(G,H)|$ in $O((h^{k+1}2^{h+e} \max\{k, 2^{2(h+e)}\} + f(k))n)$ time, where $h = |V(H)|$ and $e = |E(H)|$.*

Finally, we point out that in both algorithms of this section and for constant h, we can allow $k = O(\log n)$ and still have polynomial time (provided that we have a tree decomposition of G with width $O(\log n)$).

5. Extensions to other results

In this section, we present some consequences of the polynomial time algorithms given in the previous sections. For each problem, either we give a new polynomial-time result or we improve previously known results. Recall that if we are given a graph G, directed or undirected, all of our algorithms should be used in conjunction with one of the algorithms mentioned in the introduction, that in polynomial time, compute the treewidth decomposition of bounded width.

5.1. The directed case

Given two directed graphs $\vec{H}$ and $\vec{G}$, an *$\vec{H}$-coloring of $\vec{G}$* is any function $\theta : V(\vec{G}) \to V(\vec{H})$ with the property that $(v,w) \in E(\vec{G}) \Rightarrow (\theta(v), \theta(w)) \in E(\vec{H})$. If $\vec{H}$ is a fixed directed graph, the *$\vec{H}$-coloring problem* asks, given a directed graph $\vec{G}$ as input, whether there is a $\vec{H}$-coloring of $\vec{G}$. We define the *#$\vec{H}$-coloring problem* as the problem of, given a directed graph $\vec{G}$, counting all the $\vec{H}$-colorings of $\vec{G}$. The *#E$\vec{H}$-coloring problem* and the *#VE$\vec{H}$-coloring problem* are defined analogously.

It can be directly verified that all the algorithms and the proofs of this paper work also in the case where G and H are directed. In particular, the corresponding algorithms and proofs can be derived from the existing ones if we consider a nice tree decomposition of the underlying graph of $\vec{G}$ and replace any undirected edge $\{x, y\}$ of G or H by a directed edge (x, y). The times of the corresponding algorithms are

the same as those in Theorems 3.1, 4.2, and 4.1. In short, we state the following result.

Theorem 5.1. *The* #$\vec{H}$-coloring problem, *the* #$E\vec{H}$-coloring problem, *and the* #$VE\vec{H}$-coloring problem *are all fixed parameter tractable, with the parameters being the treewidth of the underlying graph of the input and the number of vertices of* $\vec{H}$.

5.2. Counting list H-colorings

We call any function $L: V(G) \to 2^{V(H)}$ mapping any vertex of G to a subset of $V(H)$ (H,G)-*list*. For any vertex $v \in V(G)$ we call the set $L(v) \subseteq V(H)$ *list* of v. The *list H-coloring* problem asks whether, given a graph G and a (H,G)-list L, there is an H-coloring χ of G so that for every $u \in V(G)$ we have $\chi(u) \in L(u)$. The complexity of the list H-coloring has been analyzed in [20,21]. The counting version of the list H-coloring is called *list #H-coloring*. The *list exact H-coloring* and the *list vertex exact H-coloring* are defined in the obvious way and we call their counting versions *list exact #H-coloring* and *list vertex exact #H-coloring*.

It is easy to adapt the algorithm count-H and obtain an algorithm for the list #H-coloring problem for graphs of bounded treewidth. The main changes are that $\mathscr{H}(G_p,H)$ is now replaced by $\mathscr{LH}(G_p,H)$ that contains all the list H-colorings of G_p and that ϕ is now a function from the set $F_p = \{\phi : X_p \to V(H) \text{ and } \forall_{x \in X_p} \phi(x) \in L(x)\}$. The only change that should be applied in algorithm count-H is on step **3** where the requirement $a \in V(H)$ should be restricted to $a \in L(v)$. Applying similar changes to the definitions of $\mathscr{V}(G_p,H)$ and $\mathscr{E}(G_p,H)$ and algorithm count-exact-V-H and count-exact-H, we can solve the *list vertex exact #H-coloring* and the *list exact #H-coloring*, respectively, for graphs of bounded treewidth. Summarizing, we have that Theorems 3.1, 4.1, and 4.2 can be rewritten in their "list" versions with the same complexity bounds.

5.3. Enumeration

Notice that for each of the algorithms described so far, if we retain the information of all the tables, it is possible to use it in order to enumerate all the homomorphisms. The storing of all the tables implies a burden of $O(n/\log n)$ to the space reported in Theorems 3.1, 4.1, and 4.2. In particular, setting up a suitable bookkeeping of the enumerated homomorphisms, a top–down traversal of the table information can pop-up each of them in $O(n)$ steps.

5.4. Coloring problems

Notice that a graph G is H-colorable if $|\mathscr{H}(G,H)| \geqslant 1$. This implies the following corollary of Theorems 3.1, 4.1, and 4.2 for the corresponding decision version of the problems examined.

Corollary 5.1. *For any graph H with h vertices and e edges, we can solve, for an n-vertex partial k-tree as input graph,*
- *the H-coloring problem in $O(nh^{k+1}\min\{k,h\})$ steps,*
- *the vertex exact H-coloring problem in $O(nh^{k+1}2^h\max\{k,2^h\})$ steps, and*
- *the exact H-coloring problem in $O(nh^{k+1}2^{h+e}\max\{k,2^{h+e}\})$ steps.*

Corollary 5.1 improves the time of the best known algorithm for the H-coloring of partial k-trees. This algorithm runs in $O(nh^{2(k+1)})$ steps and is due to Telle and Proskurowski in [39].

We denote as K_c the complete graph with c vertices. As the problem of counting the number of c-colorings of a graph G is equivalent to the $\#K_c$-coloring problem we can conclude the following.

Corollary 5.2. *An algorithm can be constructed to compute the number of c-colorings of a partial k-tree of n vertices in $O(nc^{k+1}\min\{k,c\})$ steps.*

We denote as $\beta_G(c)$ the function mapping c to the number of colorings of G that use exactly c colors, i.e. $\beta_G(c)=|\mathscr{V}(G,K_c)|$.

Lemma 5.1. *For any graph G, $\beta_G(c)=|\mathscr{H}(G,K_c)|-\sum_{1\leqslant r\leqslant c-1}\binom{c}{r}\beta_G(r)$.*

Proof. Notice that for any $r, 1\leqslant r\leqslant c$, the number of functions $\phi\in\mathscr{H}(G,K_c)$ where $|\phi(V(G))|\leqslant r$ is equal to $\binom{c}{r}\beta_G(r)$. The sum of these numbers for $r=1,\ldots,c$ gives $|\mathscr{H}(G,K_c)|$. Therefore, $|\mathscr{H}(G,K_c)|=\sum_{1\leqslant r\leqslant c}\binom{c}{r}\beta(r)$ and the lemma follows. □

The formula of Lemma 5.1 and Corollary 5.2 give a way to compute $\beta_G(c)$.

Corollary 5.3. *An algorithm can be constructed that, given partial k-tree G with n vertices and for any $c, 1\leqslant c\leqslant n$, computes $\beta_G(c)$ in $O(nc^{k+2}\min\{k,c\})$ steps.*

Corollary 5.3 can be rewritten as follows.

Corollary 5.4. *An algorithm can be constructed that computes the chromatic polynomial of a partial k-tree with n vertices in $O(kn^{k+3})$ steps.*

Up to now, the best algorithm for computing the chromatic polymonial for partial k-trees can be derived from the algorithm of Andrzejak [2]. This algorithm can compute in $O(n^{2+7\log_2 c})$ steps the Tutte polynomial of a partial k-tree of n vertices, where c is twice the number of partitions of a set with $3(k+1)$ elements. Several researchers have considered the problem of counting the number of proper c-colorings in a graph G. The problem is #P-hard for $c\geqslant 3$ and maximum degree $\Delta\geqslant 3$ [26]. In the same paper, it is proved that there exists a fully polynomial time Randomized Approximation Scheme (FPRAS) for the number of colorings in the case when $c\geqslant 2\Delta+1$. Recently, Bubley et al. in [11] proved that the problem is #P-hard for fixed Δ, but there is a FPRAS

for $c=5$ and $\Delta=3$. Edwards [19] proved that if $c\geqslant 3$ and the minimum degree $\delta\geqslant \alpha n$, $n=|V(G)|$, the counting problem is #P-complete if $\alpha<(c-2)/(c-1)$, but it is in P for $\alpha>(c-2)/(c-1)$.

5.5. Problems on cores

Given a graph G, a *core* of G is a subgraph C of G such that G is homomorphic to C, but G fails to be homomorphic to any proper subgraph of C. This notion of core is due to Hell and Nešetřil [25]. A graph G is *a core* if G is a core for itself or, equivalently, if $|\mathscr{E}(G,G)|=|\mathscr{H}(G,G)|$. For example, any graph C_{2k+1} is a core (C_{2k+1} is the connected 2-regular graph with $2k+1$ vertices). It is known that, in general, the problem of deciding whether G is a core is NP-complete [25] and in the same paper it is presented a polynomial time algorithm, to decide if G is a core, for the particular case when G has *independence number* $\alpha(G)\leqslant 2$. Finally, we mention that "almost all graphs" are cores [32,33].

We first prove the following.

Corollary 5.5. *An algorithm can be constructed that decides, in* $O(k^2n^{k+3})$ *steps, whether a partial k-tree with n vertices is a core.*

Proof. Notice that it is enough to check whether there exists a subgraph H of G where $|E(H)|=|E(G)|-1$ and such that $|\mathscr{H}(G,H)|\geqslant 1$. According to Theorem 3.1, this requires $O(|E(G)|kn^{k+2})=O(k^2n^{k+3})$ steps (take in mind that if G is a partial k-tree then $E(G)=O(k|V(G)|)$). □

We mention that the property of being a core is an EMS-property (i.e. involves counting or summing evaluations over sets definable on monadic second order logic) and therefore, the results in [15,5,1] imply the existence of a polynomial time algorithm for deciding it. So far, no explicit algorithm has been reported for this problem.

We consider the isomorphism problem on cores. Notice that graphs G and H are isomorphic iff $|\mathscr{E}(G,H)|\cdot|\mathscr{E}(H,G)|>1$. This check can be done in polynomial time when both G and H are cores and partial k-trees.

Corollary 5.6. *An algorithm can be constructed that checks, in* $O(kn^{k+2})$ *steps, whether two cores G and H of treewidth* $\leqslant k$ *are isomorphic, where* $n=\max\{|V(G)|, |V(H)|\}$.

In general, checking whether two partial k-trees are isomorphic can be done in $O(n^{k+4.5})$ steps [6]. Corollaries 5.5 and 5.6 improve this time for the case where some of the input graphs is a core.

Using the idea of Corollary 5.6, we can count automorphisms of cores with bounded treewidth.

Corollary 5.7. *An algorithm can be constructed that outputs, in* $O(kn^{k+2})$ *steps, the number of automorphisms of a core G of n vertices and treewidth* $\leqslant k$.

A graph G is called *rigid* when the identity is the only homomorphism in $\mathscr{H}(G,G)$ [25,32]. We conclude to the following.

Corollary 5.8. *An algorithm can be constructed that checks, in* $O(kn^{k+2})$ *steps, whether a n-vertex partial k-tree is rigid or not.*

5.6. Problems on counting independent sets

Notice that, in the case where H is the graph in Fig. 1, $|\mathscr{H}(G,H)|$ is the number of independent sets of G. Therefore, we have the following.

Corollary 5.9. *An algorithm can be constructed that outputs the number of independent sets of an n-vertex partial k-tree in* $O(n2^{k+1})$ *steps.*

Notice that the linear algorithm of Corollary 5.9 remains polynomial even if the treewidth of G is bounded by $O(\log n)$, provided that we have a tree decomposition of G with width $O(\log n)$.

Suppose that H is a star with a loop attached on its unique internal vertex (a star is a tree of diameter 2). Then, asking whether $|\mathscr{V}(G,H)| \geqslant 1$ is equivalent to asking whether G has an independent set of size $\geqslant k$. Finally, Theorem 4.1 yields the following.

Corollary 5.10. *An algorithm can be constructed that for any partial k-tree of n vertices and any r, outputs the number of the independent sets of G with at least r vertices in* $O(n(r+1)^{k+1}2^{r+1}\max\{k, 2^{2(r+1)}\})$ *steps.*

6. Remarks and open problems

Observe that, by specializing the structure of H, we can generate an arbitrary number of counting problems that have implication in statistical physics and are, in general, #P-complete [29,40]. As an example, we mention the problem of counting the q-particle Widom–Rowlinson configurations in bounded treewidth graphs [18].

The advantage of the three main algorithms of this paper is the "low" contribution of k in their time bounds. However, it is tempting to check whether reasonable (while still not better) bounds can emerge by directly applying the generic methodology of [15] to our problems. We base our assumption on the fact that, in [15], constants depend on the alternation depth of quantifiers in the corresponding logical expression. We find it an interesting question whether reasonable complexity bounds can be obtained for problems expressible by Monadic Second order logic formulas of few quantifier alternations.

An other direction of research is to consider that H is weighted. A weighting of H can be interpreted as a product measure on the set of H-colorings [10]. It will be of interest to know if it is possible to have a polynomial time algorithm for sampling H-colorings of graphs of bounded treewidth according to the probability distribution induced by the product measure.

Acknowledgements

We wish to thank Rafel Cases for his useful remarks on this research. Also, we are indebted to Bruno Courcelle; his well-aimed comments where substantially helpful for improving the presentation of this paper.

References

[1] K.R. Abrahamson, M.R. Fellows, Finite automata, bounded treewidth and well-quasiordering, in: N. Robertson, P. Seymour (Eds.), Proc. of the AMS Summer Workshop on Graph Minors, Graph Structure Theory, Contemporary Mathematics, Vol. 147, American Mathematical Society, Providense, RJ, 1993, pp. 539–564.
[2] A. Andrzejak, An algorithm for the Tutte polynomials of graphs of bounded treewidth, Discrete Math. 190 (1–3) (1998) 39–54.
[3] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey, BIT 25 (1985) 2–23.
[4] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Math. 8 (1993) 277–284.
[5] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree decomposable graphs, J. Algorithms 12 (1991) 308–340.
[6] H.L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, J. Algorithms 11 (1990) 631–643.
[7] H.L. Bodlaender, A linear time algorithm for finding tree decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.
[8] H.L. Bodlaender, Treewidth: algorithmic techniques and results, in: Mathematical Foundations of Computer Science 1997 (Bratislava), Springer, Berlin, 1997, pp. 19–36.
[9] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (1996) 358–402.
[10] G. Brightwell, P. Winkler, Graph homomorphism and phase transitions, J. Combin. Theory Ser. B 77 (1999) 415–435.
[11] R. Bubley, M. Dyer, C. Greenhill, M. Jerrum, On approximately counting colorings of small degree graphs, SIAM J. Comput. 29 (2) (1999) 387–400.
[12] C. Cooper, M. Dyer, A. Frieze, On Markov chains for randomly H-coloring a graph, J. Algorithms 39 (2001) 117–134.
[13] B. Courcelle, The monadic second-order logic of graphs III: treewidth, forbidden minors and complexity issues. Inform. Theory 26 (1992) 257–286.
[14] B. Courcelle, The expression of graph properties and graph transformations in monadic second-order logic, in: Grzegorz Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1, Chapter 5, World Scientific Publishing, River Edge, NJ, 1997.
[15] B. Courcelle, J.A. Makowski, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic, Discrete Appl. Math. 108 (2001) 23–52.
[16] J. Díaz, J. Nešetřil, M. Serna, H-coloring of large degree graphs, Technical Report No. 2000-465, KAM-DIMATIA Series, Charles University, 2000.
[17] M. Dyer, A. Frieze, M. Jerrum, On counting independent sets in sparse graphs, in: 40th Ann. Symp. on Foundations of Computer Science, 1999, pp. 210–217.
[18] M.E. Dyer, C.S. Greenhill, The complexity of counting graph homomorphisms, in: 11th ACM/SIAM Symp. on Discrete Algorithms, 2000, pp. 246–255.
[19] K. Edwards, The complexity of coloring problems on dense graphs, Theoret. Comput. Sci. 16 (1986) 337–343.
[20] T. Feder, P. Hell, List homomorphisms to reflexive graphs, J. Combin. Theory Ser. B 72 (2) (1998) 236–250.

[21] T. Feder, P. Hell, J. Huang, List homomorphisms and circular arc graphs, Combinatorics 19 (4) (1999) 487–505.
[22] A. Galluccio, P. Hell, J. Nešetřil, The complexity of H-colouring of bounded degree graphs, Discrete Math. 222 (1–3) (2000) 101–109.
[23] M.R. Garey, D.S. Johnson, Computers and Intractability : a Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[24] P. Hell, J. Nešetřil, On the complexity of H-coloring, J. Combin. Theory Ser. B 48 (1990) 92–110.
[25] P. Hell, J. Nešetřil, The core of a graph, Discrete Math. 109 (1992) 117–126.
[26] M. Jerrum, A very simple algorithm for estimating the number of k-colorings of a low degree graph, Random Structures Algorithms 7 (1995) 157–165.
[27] T. Kloks, Treewidth. Computations and Approximations, Lecture Notes in Computer Science, Vol. 842, Springer, Berlin, 1994.
[28] J. Lagergren, Efficient parallel algorithms for graph with bounded tree-width, J. Algorithms 20 (1996) 20–44.
[29] J.L. Lebowitz, G. Gallavotti, Phase transitions in binary lattice gases, J. Math. Phys. 12 (1971) 1129–1133.
[30] L. Levin, Universal sequential search problems, Problems Inform. Transmissions 9 (1973) 265–266.
[31] J. Matoušek, R. Thomas, Algorithms finding tree-decompositions of graphs, J. Algorithms 12 (1991) 1–22.
[32] J. Nešetřil, Aspects of structural combinatorics (graph homomorphisms and their use), Taiwanese J. Math. 3 (4) (1999) 381–423.
[33] J. Nešetřil, personal communication, 2000.
[34] B. Reed, Finding approximate separators and computing tree-width quickly, in: 24th ACM Symp. on Theory of Computing, 1992, pp. 221–228.
[35] N. Robertson, P.D. Seymour, Graph minors. I. Excluding a forest, J. Combin. Theory Ser. B 35 (1983) 39–61.
[36] N. Robertson, P.D. Seymour, Graph minors. III. Planar tree-width, J. Combin. Theory Ser. B 36 (1984) 49–64.
[37] N. Robertson, P.D. Seymour, Graph minors—a survey, in: I. Anderson (Ed.), Surveys in Combinatorics, Cambridge University Press, Cambridge, 1985.
[38] N. Robertson, P.D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms 7 (1986) 309–322.
[39] J.A. Telle, A. Proskurowski, Algorithms for vertex partitioning problems on partial k-trees, SIAM J. Discrete Math. 10 (1997) 529–550.
[40] B. Widom, J.S. Rowlinson, New model for the study of liquid–vapor plase transition, J. Stat. Phys. 52 (1970) 1670–1684.