Improved Exact Algorithms for Counting 3- and 4-Colorings

Fedor V. Fomin[1], Serge Gaspers[1] and Saket Saurabh[1,2]

[1] Department of Informatics, University of Bergen,
N-5020 Bergen, Norway.
{fomin|serge|saket}@ii.uib.no
[2] The Institute of Mathematical Sciences,
Chennai 600 113, India.
saket@imsc.res.in

Abstract. We introduce a generic algorithmic technique and apply it on decision and counting versions of graph coloring. Our approach is based on the following idea: either a graph has nice (from the algorithmic point of view) properties which allow a simple recursive procedure to find the solution fast, or the pathwidth of the graph is small, which in turn can be used to find the solution by dynamic programming. By making use of this technique we obtain the fastest known exact algorithms

- running in time $\mathcal{O}(1.7272^n)$ for deciding if a graph is 4-colorable and
- running in time $\mathcal{O}(1.6262^n)$ and $\mathcal{O}(1.9464^n)$ for counting the number of k-colorings for $k = 3$ and 4 respectively.

1 Introduction

The graph coloring problem is one of the oldest and most intensively studied problems in Combinatorics and Algorithms. The problem is to color the vertices of a graph such that no two adjacent vertices are assigned the same color. The smallest number of colors needed to color a graph G is called the *chromatic number*, $\chi(G)$, of G. The corresponding decision version of the coloring problem is k-Coloring, where for a given graph G and an integer k we are asked if $\chi(G) \leq k$. The k-Coloring problem is one of the classical NP-complete problems [12]. In fact it is known to be NP complete for every $k \geq 3$. A lot of effort was also put in designing efficient approximation algorithms for the optimization version of the problem, namely, given a k-colorable graph to try to color it with as few colors as possible. Unfortunately, it has been shown that if certain reasonable complexity conjectures hold then k-Coloring is hard to approximate within $n^{1-\epsilon}$ for any $\epsilon > 0$ [10, 13].

The history of exponential time algorithms for graph coloring is rich. Christofides obtained the first non-trivial algorithm computing the chromatic number of a graph on n vertices running in time $n!n^{O(1)}$ in 1971 [6]. In 1976, Lawler [15] devised an algorithm with running time $\mathcal{O}^*(2.4423^n)$ based on dynamic programming over subsets and enumeration of maximal independent sets. Eppstein [7] reduced the bound to $\mathcal{O}(2.4151^n)$ and Byskov [5] to

$\mathcal{O}(2.4023^n)$. In two breakthrough papers last year, Björklund & Husfeldt [3] and Koivisto [14] independently devised $2^n n^{\mathcal{O}(1)}$ algorithms based on a combination of inclusion-exclusion and dynamic programming.

Apart from the general chromatic number problem, the problem of k-Coloring for small values of k like 3, 4 has also attracted a lot of attention. The fastest algorithm deciding if a the chromatic number of a graph is at most 3 runs in time $\mathcal{O}(1.3289^n)$ and is due to Beigel & Eppstein [2]. For 4-Coloring Byskov [5] designed the fastest algorithm, running in time $\mathcal{O}(1.7504^n)$.

The counting version of the k-Coloring problem, #k-Coloring, is to count the number of all possible k-colorings of a given graph. #k-Coloring (and its generalization known as *Chromatic Polynomial*) are among the oldest counting problem. Recently Björklund & Husfeldt [3] and Koivisto [14] have shown that the chromatic polynomial of a graph can be computed in time $2^n n^{\mathcal{O}(1)}$.

For small k, #k-Coloring was also studied in the literature. Angelsmark et al. [1] provide an algorithm for #3-Coloring with running time $\mathcal{O}(1.788^n)$. Fürer and Kashiviswanathan [11] show how to solve #3-Coloring with running time $\mathcal{O}(1.770^n)$. No algorithm faster than $2^n n^{\mathcal{O}(1)}$ for #4-Coloring was known in the literature [1, 3, 11, 14].

Our results. In this paper we introduce a generic technique to obtain exact algorithms for coloring problems and its different variants. This technique can be seen as a generalization of the technique introduced in [8] for a different problem. The technique is based on the following combinatorial property which is proved algorithmically and which is interesting in its own: *Either* a graph G has a nice "algorithmic" property which (very sloppily) means that when we apply branching or a recursive procedure to solve a problem then the branching procedure on subproblems of a smaller size works efficiently, *or* (if branching is not efficient) the pathwidth of the graph is small. This type of technique can be used for a variety of problems (not only coloring and its variants) where sizes of the subproblems on which the algorithm is called recursively decrease significantly by branching on vertices of high degrees.

In this paper we use this technique to obtain exact algorithms for different coloring problems. We show that #3-Coloring and #4-Coloring can be solved in time $\mathcal{O}(1.6262^n)$ and $\mathcal{O}(1.9464^n)$ respectively. We also solve 4-Coloring in time $\mathcal{O}(1.7272^n)$. These improve the best known results for each of the problems.

2 Preliminaries

In this paper we consider simple undirected graphs. Let $G = (V, E)$ be a graph and let n denote the number of vertices and m the number of edges of G. We denote by $\Delta(G)$ the maximum vertex degree in G. For a subset $V' \subseteq V$, $G[V']$

is the graph induced by V', and $G - V' = G[V \setminus V']$. For a vertex $v \in V$ we denote the set of its neighbors by $N(v)$ and its *closed neighborhood* by $N[v] = N(v) \cup \{v\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. An *independent set* in G is a subset of pair-wise non-adjacent vertices. A subset of vertices $S \subseteq V$ is a *vertex cover* in G if for every edge e of G at least one endpoint of e is in S.

Major tools of our paper are tree and path decompositions of graphs. A *tree decomposition* of G is a pair $(\{X_i : i \in I\}, T)$ where each X_i, $i \in I$, is a subset of V, called a *bag* and T is a tree with elements of I as nodes such that we have the following properties :

1. $\cup_{i \in I} X_i = V$;
2. for all $\{u, v\} \in E$, there exists $i \in I$ such that $\{u, v\} \subseteq X_i$;
3. for all $i, j, k \in I$, if j is on the path from i to k in T then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition is equal to $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all its tree decompositions and it is denoted by $\mathbf{tw}(G)$. We speak of a *path decomposition* when the tree T in the definition of a tree decomposition is restricted to be a path. The *pathwidth* of G is defined similarly to its *treewidth* and is denoted by $\mathbf{pw}(G)$.

We need the following bound on the pathwidth of graphs with small vertex degrees.

Proposition 1 ([8]). *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices,*

$$\mathbf{pw}(G) \leq \frac{1}{6} n_3 + \frac{1}{3} n_4 + \frac{13}{30} n_5 + \frac{23}{45} n_6 + n_{\geq 7} + \varepsilon n,$$

where n_i is the number of vertices of degree i in G for any $i \in \{3, \ldots, 6, \geq 7\}$. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

In our algorithms we also use the following results.

Proposition 2 ([7]). *The number of maximal independent sets of size k in a graph on n vertices is at most $3^{4k-n} 4^{n-3k}$ and can be enumerated with polynomial time delay.*

Our $\mathcal{O}^*$ notation suppresses polynomial terms. Thus we write $\mathcal{O}^*(T(x))$ for a time complexity of the form $\mathcal{O}(T(x) \cdot |x|^{\mathcal{O}(1)})$ where $T(x)$ grows exponentially with $|x|$, the input size.

3 Framework for combining enumeration and pathwidth arguments

Let us assume that we have a graph problem for which

(a) we know how to solve it by enumerating independent sets, or maximal independent sets, of the input graph (for an example to check whether a graph G is 3-colorable, one can enumerate all independent sets I of G and for each independent set I can check whether $G - I$ is bipartite), and
(b) we also know how to solve the problem using dynamic programming over the path decomposition of the input graph.

For some instances, the first approach might be faster and for other instances, the path decomposition algorithm might be preferable. One method to get the best of both algorithms would be to compute a path decomposition of the graph using Proposition 1, and choose one of the two algorithms based on the width of this path decomposition. Unfortunately, this method is not very helpful in obtaining better worst case bounds on the running time of the algorithm.

Here in our technique we start by enumerating (maximal) independent sets and based on the knowledge we gain on the graph by this enumeration step, we prove that either the enumeration algorithm is fast, or the pathwidth of the graph is small. This means that either the input graph has a good algorithmic property, or it has a good graph-theoretic property.

To enumerate (maximal) independent sets of the input graph G, we use a very standard approach. Two sets I and C are constructed by a recursive procedure, where I is the set of vertices in the independent set and C the set of vertices not in the independent set. Let v be a vertex of maximum degree in $G - I - C$, the algorithm makes one recursive call where it adds v to I and all its neighbors to C and another recursive call where it adds v to C. This branching into two subproblems decreases the number of vertices in $G - I - C$ according to the following recurrence

$$T(n) \leq T(n - d(v) - 1) + T(n - 1).$$

From this recurrence, we see that the running time of the algorithm depends on how often it branches on a vertex of high degree. This algorithmic property is reflected by the size of C: frequent branchings on vertices of high degree imply that $|C|$ grows fast (in one branch).

On the other hand we can exploit a graph-theoretic property if C is small and there are no vertices of high degree in $G - I - C$. Based on the work of Monien and Preis on the bisection width of 3-regular graphs [16], small upper bounds on the pathwidth of the input graph G depending on their maximum degree have been obtained [8, 9] (also see Proposition 1). If a path

Input: A graph G, an independent set I of G and a set of vertices C such that $N(I) \subseteq C \subseteq V(G) - I$.
Output: An optimal solution which has the problem-dependent properties.

if *($\Delta(G - I - C) \geq \mathtt{a}$) or*
($\Delta(G - I - C) = \mathtt{a} - 1$ and $|C| > \alpha_{\mathtt{a}-1}|V(G)|$) or
($\Delta(G - I - C) = \mathtt{a} - 2$ and $|C| > \alpha_{\mathtt{a}-2}|V(G)|$) or
$\cdots$ or
($\Delta(G - I - C) = 3$ and $|C| > \alpha_3|V(G)|$)
then
  choose a vertex $v \in V(G) - I - C$ of maximum degree in $G - I - C$
  $S_1 \leftarrow \mathtt{enumISPw}(G, I \cup \{v\}, C \cup N(v))$ **R1**
  $S_2 \leftarrow \mathtt{enumISPw}(G, I, C \cup \{v\})$ **R2**
  return `combine`(S_1, S_2)
else if $\Delta(G - I - C) = 2$ *and* $|C| > \alpha_2|V(G)|$ **then**
  return `enumIS`(G, I, C)
else
  Stop this algorithm and **run** `Pw`(G, I, C) instead.

Fig. 1. Algorithm `enumISPw`(G, I, C)

decomposition of $G - I - C$ of size $\beta_d|V(G) - I - C|$ can be computed, then a path decomposition of G of size $\beta_d|V(G) - I - C| + |C|$ can be computed easily. Here β_d is a constant strictly less than 1 depending on the maximum degree of the graph. If it turns out that a path decomposition of small width can be computed, the algorithm enumerating (maximal) independent sets is completely stopped without any further backtracking and an algorithm based on this path decomposition is executed on the original input graph.

In the rest of this section, we give a general framework combining

- algorithms based on the enumeration of maximal independent sets, and
- algorithms based on path decompositions of small width,

and discuss the running time of the algorithms based on this framework. This framework is not problem-dependent and it relies on two black boxes that have to be replaced by appropriate procedures to solve a specific problem.

Algorithm `enumISPw`(G, I, C) in Figure 1 is invoked with the parameters $(G, \emptyset, \emptyset)$, where G is the input graph, and the algorithms `enumIS` and `Pw` are problem-dependent subroutines. The function `combine` takes polynomial time and is also problem-dependent. The values for $\mathtt{a}$, $\alpha_{\mathtt{a}}, \cdots$, and α_2 ($0 = \alpha_{\mathtt{a}} \leq \alpha_{\mathtt{a}-1} \leq \cdots \alpha_2 < 1$) are carefully chosen constants to balance the time complexities of enumeration and path decomposition based algorithms and to optimize the overall running time of the combined algorithm.

Algorithm `enumIS`(G, I, C) is problem-dependent and returns an optimal solution *respecting* the choice for I, the independent set, and C, the set of the vertices not belonging to the independent set (a vertex cover of $G[I \cup C]$). The

sets I and C are supposed to be completed into a (maximal) independent set and a (minimal) vertex cover for G by enumerating (maximal) independent sets of $G - I - C$, before the problem-specific treatment is done.

Algorithm $\texttt{Pw}(G, I, C)$ first computes a path decomposition based on G, I and C and the maximum degree of $G-I-C$. It then calls a problem-dependent algorithm based on this path decomposition of G.

Let n denote the number of vertices of G, $T(n)$ the running time of Algorithm `enumISPw` on G, $T_e(n, i, c)$ the running time of Algorithm `enumIS` and $T_p(n, i, c)$ the running time of Algorithm `Pw` with parameters G, I, C where $i = |I|$ and $c = |C|$. Also, suppose that for any graph with n vertices and maximum degree d, a path decomposition of width at most $\beta_d n$ can be computed. The following lemma is used by Algorithm `Pw` to compute a path decomposition of G of small width.

Lemma 1. *Suppose that given a graph H with $\Delta(H) \leq d$, a path decomposition of width at most $\beta_d|H|$ can be computed where $\beta_d < 1$ is a constant depending on d alone. Then a path decomposition of width at most $\beta_d|V(G) - I - C| + |C|$ can be computed for a graph G if I is an independent set in G, $N(I) \subseteq C \subseteq V(G)$ and $\Delta(G - I - C) \leq d$.*

Proof. As I is an independent set in G and C separates I from $G-I-C$, every vertex in I has degree 0 in $G - C$. Thus, a path decomposition of $G - C$ of size at most $\beta_d|V(G) - I - C|$ can be computed. Adding C to each bag of this path decomposition gives a path decomposition of width at most $\beta_d|V(G) - I - C| + |C|$ of G. □

Given the conditions under which `Pw` is executed, the following lemma upper bounds its running time.

Lemma 2. *If the considered problem can be solved on G in time $O^*(t_{pw}^{\ell})$, given a path decomposition of width ℓ of G, then*

$$T_p(n, i, c) = O^*\left(\max_{d\in\{2,3,\cdots,\mathtt{a}-1\}}\left(t_{pw}^{(\beta_d+(1-\beta_d)\alpha_d)n}\right)\right).$$

Proof. The lemma follows from Lemma 1 and the conditions on $|C|$ and $\Delta(G-I-C)$ under which Algorithm `Pw` is executed. □

To estimate the size of the search tree we assume that Algorithm `Pw` is not executed. Let t_n, t_i and t_c be constants such that $T_e(n, i, c) = O^*(t_n^n t_i^i t_c^c)$.

Lemma 3. *If Algorithm `Pw` is not executed, then*

$$T(n) = O^*\left(t_n^n t_c^{\alpha_2 n}\prod_{d=3}^{\mathtt{a}} t_d^{\Delta\alpha_d n}\right)$$

where $t_d = (1 + r_d)$ *and* r_d *is the minimum positive root of*

$$(1 + r)^{-(d-1)} \cdot r^{-1} \cdot t_i - 1.$$

Proof. We divide $T(n)$ into $T_d(n, i, c)$ for $d \in \{2, 3, \cdots, \mathsf{a}\}$ where d corresponds to the maximum degree of $G - I - C$ if $d < \mathsf{a}$ and $T_{\mathsf{a}}(n, 0, 0) = T(n)$ if Algorithm `Pw` is not executed. Clearly, $T_2(n, i, c) = T_e(n, i, c)$. Let us now express $T_d(n, i, c)$ in terms of $T_{d-1}(\cdot, \cdot, \cdot)$ for $d \in \{3, \cdots, \mathsf{a}\}$. Consider the part of the search tree with branchings on vertices of degree d (or at least d if $d = \mathsf{a}$). Observe that $|C|$ increases in the worst case by at most $(\alpha_{d-1} - \alpha_d)n = \Delta\alpha_d n$ in this part of the search tree. In each branch of the type **R1**, $|C|$ increases by d and in each branch of the type **R2**, $|C|$ increases by 1. Let $r \in [0, \Delta\alpha_d n/d]$ be the number of times the algorithm branches according to **R1**, then it branches $\Delta\alpha_d n - dr$ times according to **R2**. We get that

$$T_d(n, i, c) = O^*\left(\sum_{r=0}^{\Delta\alpha_d n/d} \binom{\Delta\alpha_d n - (d-1)r}{r} T_{d-1}(n, i + r, c + \Delta\alpha_d n)\right).$$

To prove the lemma, it is sufficient to expand $T_{\mathsf{a}}(n, 0, 0)$ and to prove that $\sum_{r=0}^{\Delta\alpha_d n/d} \binom{\Delta\alpha_d n-(d-1)r}{r} t_i^r \leq t_d^{\Delta\alpha_d n}$.

The sum over binomial coefficients $\sum_{r=0}^{\Delta\alpha_d n/d} \binom{\Delta\alpha_d n-(d-1)r}{r} t_i^r$ is bounded by $(\Delta\alpha_d n/d)B$ where B is the maximum term in this sum. Let us assume that $B = \binom{\Delta\alpha_d n-(d-1)j}{j} t_i^j$ for some $j \in \{0, 1, \ldots, \Delta\alpha_d n/d\}$.

$$B = \binom{\Delta\alpha_d n - (d-1)j}{j} t_i^j \leq \frac{(1 + r_i)^{\Delta\alpha_d n-(d-1)j}}{r_i^j} t_i^j.$$

Here we use the well known fact that for any $x > 0$ and $0 \leq k \leq n$,

$$\binom{n}{k} \leq \frac{(1 + x)^n}{x^k} \quad .$$

By choosing r_i to be the minimum positive root of $\frac{(1+r)^{-(d-1)}}{r} t_i - 1$, we arrive at $B < (1 + r_i)^{\Delta\alpha_d n} = t_d^{\Delta\alpha_d n}$. □

The following theorem combines Lemmas 2 and 3 to upper bound the overall running time of the algorithms resulting from this framework.

Theorem 1. *The running time of Algorithm* `enumISPw` *on a graph on* n *vertices is*

$$T(n) = O^*\left(t_n^n t_c^{\alpha_2 n} \prod_{d=3}^{\mathsf{a}} t_d^{\Delta\alpha_d n} + \max_{d \in \{2,3,\cdots,\mathsf{a}-1\}} \left(t_{pw}^{(\beta_d + (1-\beta_d)\alpha_d)n}\right)\right)$$

where $t_d = (1 + r_d)$ *and* r_d *is the minimum positive root of*

$$(1 + r)^{-(d-1)} \cdot r^{-1} \cdot t_i - 1.$$

The current best values for $\beta_d, d = \{2, \cdots, 6\}$ are obtained from Proposition 1.

4 Applications

In this section we use the framework of the previous section to derive improved algorithms for #3-Coloring, #4-Coloring and 4-Coloring.

4.1 Counting 3-Colorings

We first describe the problem-dependent subroutines we need to use in our Algorithm `enumISPw` to solve #3-Coloring in time $\mathcal{O}(1.6262^n)$.

Algorithm `enumISPw` returns here an integer, I corresponds to the color class C_1 and C to the remaining two color classes C_2 and C_3. Algorithm `enumIS` with parameters G, I, C enumerates all independent sets of $G - I - C$ and for each, adds this independent set to I, then checks if $G - I$ is bipartite. If $G - I$ is bipartite, then a counter counting the independent sets is incremented. This takes time $T_e(n, i, c) = 2^{n-i-c}$. Thus, $t_n = 2, t_i = 1/2$ and $t_c = 1/2$.

The function `combine` corresponds in this case to the plus-operation. The running time of Algorithm `Pw` is based on the following lemma.

Lemma 4. *Given a graph* $G = (V, E)$ *with a path decomposition of* G *of width* ℓ, #k-Coloring *can be solved in time* $\mathcal{O}(k^\ell n^{O(1)})$.

Now we use Theorem 1 and Proposition 1 to evaluate the overall complexity of our #3-Coloring algorithm.

Theorem 2. *The* #3-Coloring *problem can be solved in time* $\mathcal{O}(1.6262^n)$ *for a graph on* n *vertices.*

Proof. We use Theorem 1 and Lemma 1 with $\mathtt{a} = 5, \alpha_2 = 0.44258, \alpha_3 = 0.33093$ and $\alpha_4 = 0.16387$. The pathwidth part of the algorithm takes time

$$O^*\left(\max\left(3^{\alpha_2 n}, 3^{(1+5\alpha_3)n/6}, 3^{(1+2\alpha_4)n/3}\right)\right)$$

$$= \mathcal{O}(1.62617^n).$$

The branching part of the algorithm takes time

$$O^*\left(2^n \cdot (1/2)^{\alpha_2 n} \cdot 1.29716^{(\alpha_2-\alpha_3)n} \cdot 1.25373^{(\alpha_3-\alpha_4)n} \cdot 1.22329^{\alpha_4 n}\right)$$

$$= \mathcal{O}(1.62617^n).$$

□

4.2 Counting 4-Colorings

To solve #4-Coloring, Algorithm `enumIS` with parameters G, I, C enumerates all independent sets of $G - I - C$ and for each, adds this independent set to I, then counts the number of 3-colorings of $G - I$ using the previous algorithm. This takes time $T_e(n, i, c) = \sum_{\ell=0}^{n-i-c} 1.62617^{\ell+c}$. Thus, $t_n = 2.62617, t_i = 1/2.62617$ and $t_c = 1.62617/2.62617$. We evaluate the running time as previously.

Theorem 3. *The #4-Coloring problem can be solved in time $\mathcal{O}(1.9464^n)$ for a graph on n vertices.*

Proof. We use Theorem 1 and Lemma 1 with $\mathtt{a} = 6, \alpha_2 = 0.480402, \alpha_3 = 0.376482, \alpha_4 = 0.220602$ and $\alpha_5 = 0.083061$. The pathwidth part of the algorithm takes time

$$O^*\left(\max\left(4^{\alpha_2 n}, 4^{(1+5\alpha_3)n/6}, 4^{(1+2\alpha_4)n/3} 4^{(13+17\alpha_5)n/30}\right)\right)$$
$$= \mathcal{O}(1.9464^n).$$

The branching part of the algorithm takes time

$$O^*\big(\, 2.62617^n \cdot (1.62617/2.62617)^{\alpha_2 n} \cdot 1.24548^{(\alpha_2-\alpha_3)n} \cdot 1.21324^{(\alpha_3-\alpha_4)n} \cdot$$
$$1.18993^{(\alpha_4-\alpha_5)n} \cdot 1.17212^{\alpha_5 n}\big) = \mathcal{O}(1.9464^n).$$

□

4.3 4-Coloring

A well known technique [15] to check if a graph is k-colorable is to check for all maximal independent sets I of size at least $\lceil n/k \rceil$ whether $G - I$ is $(k-1)$-colorable. In the analysis, we use Proposition 2 to bound the number of maximal independent sets of a given size.

We also need the current best algorithm deciding 3-Coloring.

Theorem 4 ([2]). *The 3-Coloring problem can be solved in time $O(1.3289^n)$ for a graph on n vertices.*

In Algorithm `enumISPw`, which returns here a boolean, I corresponds to the color class C_1 and C to the remaining three color classes C_2, C_3 and C_4. Algorithm `enumIS` with parameters G, I, C enumerates all maximal independent sets of $G - I - C$ of size at least $\lceil n/k \rceil - |I|$ and for each, adds this independent set to I, then checks if $G - I$ is 3-colorable using Theorem 4. If yes, then G is 4-colorable. This takes time

$$T_e(n, i, c) = \sum_{\ell=\lceil n/4 \rceil - i}^{n-i-c} 3^{4\ell-n+c+i} 4^{n-c-i-3\ell} 1.3289^{n-i-\ell}.$$

As $\sum_{\ell=0}^{\lfloor 3n/4\rfloor - c} 3^{4\ell} 4^{-3\ell} 1.3289^{-\ell}$ is upper bounded by a constant, $t_n = 4^{1/4} 1.3289^{3/4}$, $t_i = 4^2/3^3$ and $t_c = 3/4$.

Theorem 5. *The* 4-Coloring *problem can be solved in time* $\mathcal{O}(1.7272^n)$ *for a graph on* n *vertices.*

Proof. We use Theorem 1 and Lemma 1 with $\mathsf{a} = 5, \alpha_2 = 0.39418, \alpha_3 = 0.27302$ and $\alpha_4 = 0.09127$, and the pathwidth algorithm of Lemma 4. □

References

1. O. Angelsmark and P. Jonsson. *Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems.* In the Proceedings of CP 2003. LNCS 2833: 81-95 (2003).
2. R. Beigel and D. Eppstein. *3-coloring in time* $O(1.3289^n)$. Journal of Algorithms 54 (2): 168-204 (2005).
3. A. Björklund and T. Husfeldt. *Inclusion–Exclusion Algorithms for Counting Set Partitions.* In the Proceedings of FOCS 2006: 575-582 (2006).
4. J. M. Byskov. *Exact Algorithms for Graph Colouring and Exact Satisfiability.* PhD Dissertation, (2004).
5. J. M. Byskov. *Enumerating Maximal Independent Sets with Applications to Graph Colouring.* Operations Research Letters 32(6): 547-556 (2004).
6. N. Christofides. An algorithm for the chromatic number of a graph. Computer J., 14: 38-39, 1971.
7. D. Eppstein. *Small Maximal Independent Sets and Faster Exact Graph Coloring.* J. Graph Algorithms Appl. 7(2): 131-140 (2003).
8. F. V. Fomin, S. Gaspers, S. Saurabh and A. A. Stepanov. *On Two Techniques of Combining Branching and Treewidth.* Report No 337, December 2006, Department of Informatics, University of Bergen, Norway.
9. F. V. Fomin and K. Høie. *Pathwidth of cubic graphs and exact algorithms.* Information Processing Letters 97(5): 191-196 (2006).
10. U. Feige and J. Kilian. *Zero Knowledge and the Chromatic Number.* Journal of Computer and System Sciences 57(2): 187-199 (1998).
11. M. Fürer and S. P. Kasiviswanathan. *Algorithms for counting 2-SAT solutions and colorings with applications.* In ECCC 33, 2005.
12. M. R. Garey and D. S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA., (1979).
13. S. Khot, A. K. Ponnuswami. *Better Inapproximability Results for MaxClique, Chromatic Number and Min-3Lin-Deletion.* In the proceedings of ICALP 2006. LNCS 4051: 226-237 (2006).
14. M. Koivisto. *An* $O(2^n)$ *algorithm for graph coloring and other partitioning problems via inclusion-exclusion.* In Proceedings of FOCS 2006: 583-590 (2006).
15. E. L. Lawler. *A Note on the Complexity of the Chromatic Number.* Information Processing Letters 5 (3): 66-67 (1976).
16. B. Monien and R. Preis. *Upper bounds on the bisection width of 3- and 4-regular graphs.* J. Discrete Algorithms 4(3): 475-498 (2006).