An exact method for graph coloring

Corinne Lucet, Florence Mendes, Aziz Moukrim

An exact method for graph coloring*

C. Lucet, F. Mendes
LaRIA EA 2083, 5 rue du Moulin Neuf 80000 Amiens - France
(Corinne.Lucet, Florence.Mendes)@laria.u-picardie.fr

A. Moukrim
HeuDiaSyC UMR CNRS 6599 UTC, BP 20529 60205 Compiègne - France
Aziz.Moukrim@hds.utc.fr

November 15, 2004

Abstract

We are interested in the graph coloring problem. We propose an exact method based on a linear-decomposition of the graph. The complexity of this method is exponential according to the linearwidth of the entry graph, but linear according to its number of vertices. We present some experiments performed on literature instances, among which COLOR02 library instances. Our method is usefull to solve more quickly than other exact algorithms instances with small linearwidth, such as *mug* graphs. Moreover, our algorithms are the first to our knowledge to solve the COLOR02 instance *4-Inser_3* with an exact method.

Keywords: graph coloring, exact method, linearwidth, linear-decomposition.

1 Introduction

The notions of tree-decomposition and path-decomposition have been introduced by Robertson and Seymour [23]. The decomposition method we propose here is strongly related to these notions, which have been studied in particular by Bodlaender to solve some NP-hard problems [1].

Our approach is a method based on successive decompositions of the representative graph providing successive resolved subgraphs and their corresponding

*with the financial support of Conseil Régional de Picardie and FSE

separating sets named *boundary sets*. At each step, solutions of the resolved subgraph are represented by the different states of the boundary set vertices. The number of states to enumerate grows exponentially with the size of the boundary set. Its maximum size, for an optimal vertex numbering, corresponds to the linearwidth of the graph. The main advantage of this method is that the exponential factor of its complexity does not depend on the size of the graph but only on its linearwidth. This technique has been implemented efficiently by Carlier, Lucet and Manouvrier to solve various NP-hard problems such as network reliability or minimal Steiner tree computation [4, 18, 19].

We apply the decomposition method to one of the most studied problems of combinatorial optimization: the graph coloring problem. It constitutes a central problem in a lot of applications such as school timetabling, scheduling, or frequency assignment [5, 6]. The graph coloring problem consists in coloring the vertices of a graph with a minimum number of colors, ensuring that two adjacent vertices do not receive the same color. Various heuristic approaches have been proposed for this NP-hard problem [12]: greedy algorithms such as DSATUR [3], metaheuristics based on local search, tabu method, simulated annealing, hybrid algorithms, etc. (see for example [10, 11, 13, 16, 20, 22, 26]). To our knowledge, few exact methods are proposed to resolve this problem. One of the most well-known exact algorithms is the exact branch-and-bound algorithm implemented by Brelaz that uses DSATUR principles [3]. Implicit enumeration strategies are used in [17, 25, 27]. Mehrotra and Trick [21] studied a linear programming formulation which is solved by using column generation techniques. More recently, Mendez Diaz and Zabala presented a branch-and-cut algorithm [8, 9]. Herrmann and Hertz presented efficient algorithms used to find edge-critical and vertex-critical subgraphs that have same chromatic numbers as initial graphs but are easier to solve [15]. Desrosiers, Galinier and Hertz proposed different algorithms to detect these critical subgraphs [7]. They made experiments on random graphs and on different types of benchmark graphs. Their method is very efficient on several instances families.

Our paper is organized as follows. In section 2, we describe the decomposition method and introduce the necessary notions. In section 3, we develop the implementation of the method. We present an exact algorithm which enables us to solve efficiently large instances whose linearwidth is bounded. Computational results obtained on various instances are presented in section 4. They compare with two exact methods: a branch-and-cut algorithm [9] and an algorithm based on vertex-critical subgraphs detection [7]. Finally, we give some conclusions and discuss about the perspectives of this work.

2 Graph decompositions

To introduce the kind of decomposition that we use to solve the graph coloring problem, it is necessary to recall some graph theory definitions and the notions of tree-decomposition and path-decomposition.

2.1 Preliminary definitions

An *undirected graph* G is a pair, $G = (V, E)$, made up of a vertex set V and an edge set $E \subset V \times V$. A graph G is *connected* if for all vertices $w, v \in V (w \neq v)$, there exists a path from w to v. Without loss of generality, the graphs G we will consider in the following of this paper will be undirected and connected graphs. A *subgraph* of $G = (V, E)$, induced by $W \subseteq V$, is a graph $G(W) = (W, E_W)$ such that $E_W = E \cap (W \times W)$. A *tree* is a simple undirected graph, $T = (I, E_T)$, without cycle and with $|E_T| = |I| - 1$. A *rooted tree* is a tree directed from the root r to the leaves. If the edge (p, v) belongs to a rooted tree, p is the *father* of v, and v is one of the *sons* of p.

2.2 Tree-decomposition

A *tree-decomposition* of $G = (V, E)$ is a pair $(\{X_i / i \in I\}, T = (I, E_T))$ with $\{X_i / i \in I\}$ a family of subsets of V and T a tree such that:

– $\bigcup_{i \in I} X_i = V$,

– for all edges $(v, w) \in E$, there exists a subset $X_i, i \in I$, with $v \in X_i$ and $w \in X_i$,

– for all $i, j, k \in I$, if j is on the path from i to k in T then $X_i \cap X_k \subseteq X_j$.

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

The decomposition method is as follows. Given a tree-decomposition of the graph, partial solutions are built on the subsets X_i and then associated to solve the considered problem. The decomposition method computes the solutions from the leaves to the root of the tree T, by examinating all partial solutions on every subgraph $G(X_i)$. The number of partial solutions is exponential according to the size of the subgraphs X_i. These partial solutions are computed from the solutions of X_f, for all f belonging to the sons of i in T. Unlike a simple enumerative method, this method allows one to factorize partial solutions of the X_i sets into classes. This factorization provides an efficient method if the cardinality of the sets X_i is small, i.e. if the treewidth of the tree-decomposition is sufficiently small.

Whereas for some graph families, such as trees and serie-parallel graphs, one can compute the treewidth in linear time, computing the treewidth of any graph is a NP-complete problem [24]. Bodlaender [2] gives for a constant k an algorithm in $O(n)$ which for a graph G solves the problem "is the treewidth of G at most k ?". If so, it determines a tree-decomposition with treewidth at most k. This algorithm based on clique search and graph contraction has an exponential complexity with respect to k ($O(n * 2^{k^2})$). It cannot be used in practice, even for $k = 4$.

2.3 Path-decomposition and linear-decomposition

The decomposition method that we will use in the following is based on a special case of tree-decomposition. We will consider a tree with only one leaf, that is a path.

A *path-decomposition* $(X_1, \ldots, X_r)$ of a graph G is an ordered sequence of subsets of V such that:

– $\bigcup_{1 \leq i \leq r} X_i = V$,

– for all edges $(v, w) \in E$, there exists a subset $X_i, 1 \leq i \leq r$, with $v \in X_i$ and $w \in X_i$,

– for all $i, j, k \in \{1, \ldots, r\}$, if $i \leq j \leq k$ then $X_i \cap X_k \subseteq X_j$.

The *pathwidth* of a path-decomposition is $max_{1 \leq i \leq r}(|X_i| - 1)$. The pathwidth of a graph is the minimum pathwidth over all possible path-decompositions of G. A *vertex linear ordering* of G is a bijection $\mathcal{N} : V \rightarrow \{1, \ldots, |V|\}$. For more clarity, we denote k the vertex $\mathcal{N}^{-1}(k)$. Let $F_i = \{j \in V / \exists (j, l) \in E \; j \leq i \; < l\} \; \forall i \in \{1, \ldots, |V|\}$. The *linearwidth* of a vertex linear ordering $\mathcal{N}$ is $F_{max}(\mathcal{N}) = max_{i \in V}(|F_i|)$. The linearwidth of G, written $F_{max}(G)$, is the minimum linearwidth over all possible vertex linear orderings of G. The linearwidth of a graph equals its pathwidth [19].

Computing the pathwidth or the linearwidth of any graph is a NP-complete problem [24], similarly as computing the treewidth of any graph. The treewidth of a graph G is smaller or equal to its pathwidth, and as a consequence the exponential factor of a tree-decomposition is smaller than that of a path-decomposition. Nevertheless, implementing the decomposition method on a linear-decomposition is easier than using a tree-decomposition. First, from a technical point of view, several X_i partial solutions may have to be stored in memory when resolving a problem with a tree-decomposition. It involves some problems of memory storage and combination of the X_i when implementing the algorithm. Moreover, creating a linear-decomposition is easier than creating a

tree-decomposition. Thus, we use a vertex linear ordering of the graph to resolve the graph coloring problem with a linear-decomposition. The resolution method is then based on a sequential insertion of the vertices, using a vertex linear ordering previously determined. This will be developed in the following section.

3 Application to the graph coloring problem

In this section, we propose a method which uses linear-decomposition in order to solve the graph coloring problem.

3.1 Problem definition

A *coloring* of a graph $G = (V, E)$ is an assignment of a color $c(i) \in I$ to each vertex such that $c(i) \neq c(j)$ for all edges $(i, j) \in E$.
If the cardinality of I is k, the coloring of G is called a *k-coloring*. The minimum value of k for which a k-coloring is possible is called the *chromatic number* of G and is denoted $\chi(G)$. The graph coloring problem consists in finding the chromatic number of a graph.

3.2 Linear decomposition principle

Consider a graph $G = (V, E)$. Let $N = |V|$ and $M = |E|$. The vertices of G are numbered according to a linear ordering $\mathcal{N} : V \rightarrow \{1, \ldots, N\}$. Let V_i be subset of V, made of the vertices numbered from 1 to i. Let $H_i = (V_i, E_i)$ be the subgraph of G induced by V_i. F_i is the *boundary set* of H_i, i.e. the subset of V_i such that $v \in F_i$ if and only if $\exists (v, w) \in E$ and $v \leq i < w$ (see figure 1). Let $H'_i = (V'_i, E'_i)$ be the subgraph of G induced by $V'_i = (V \setminus V_i) \cup F_i$. Any kind of relation between the vertices of H_i and those of H'_i depends on the vertices of F_i.

The linear decomposition is a dynamic method. During the coloring we will consider N subgraphs $H_1, \ldots, H_N$ and the N corresponding boundary sets $F_1, \ldots, F_N$. Starting from a vertex linear ordering, we build at first iteration a subgraph H_1 which contains only the vertex 1, then at each step the next vertex and its corresponding edges are added, until H_N. Partial solutions of step i are built from partial solutions of step $i - 1$.

At each subgraph H_i corresponds a boundary set F_i containing the vertices of H_i which have at least one neighbor in H'_i. The boundary set F_i is built from F_{i-1} by adding the vertex i and removing the vertices that have no neighbor with

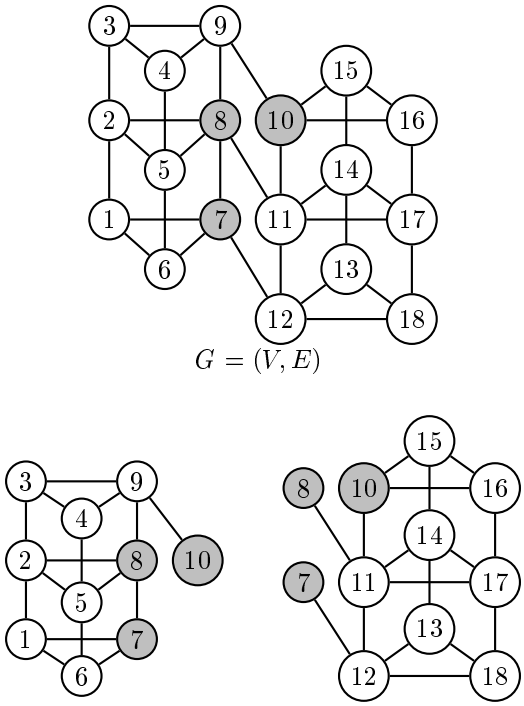


Figure 1: Subgraph H_{10} of G and its boundary set $F_{10} = \{7, 8, 10\}$

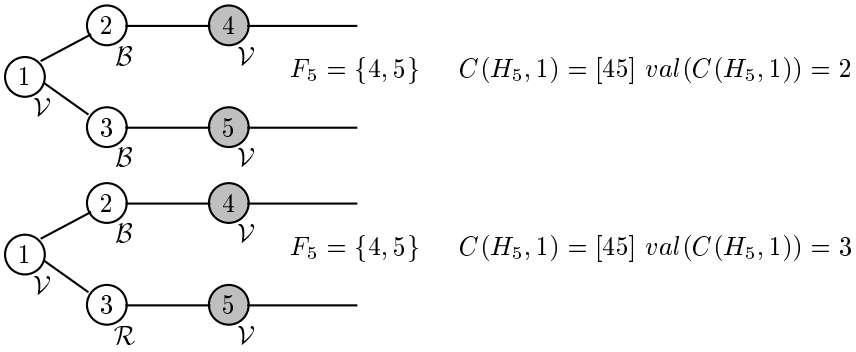


Figure 2: Different colorings of H_5 but same configuration of F_5

an ordering number greater than i. Several colorings of H_i may correspond to the same coloring of F_i (see figure 2). Moreover, the colors used by the vertices $V_i \setminus F_i$ do not interfere with the coloring of the vertices which have an ordering number greater than i, since no edge exists between them. So, only the partial solutions corresponding to different colorings of F_i have to be stored in memory. This way, several partial solutions on H_i may be summarized by a unique partial solution on F_i, called *configuration of* F_i.

The graph coloring problem is solved by evaluating at each step the configurations of the boundary set F_i. At step i, the subgraph H_{i-1} is solved. It means that to each configuration of F_{i-1} corresponds a value of the minimum number of colors necessary to color H_{i-1} for this fixed coloring of the boundary set vertices. Then, partial solutions are built using solutions of the precedent step. This point will be detailed in section 3.4.

3.3 Boundary set configurations

A configuration of the boundary set F_i is a given coloring of the vertices of F_i. This can be represented by a partition of F_i, denoted $B_1, \ldots, B_j$, such that two vertices u, v of F_i are in the same block B_c if and only if they have the same color. The number of configurations of F_i depends obviously on the number of edges between the vertices of F_i. The minimum number of configurations is 1. If the vertices of F_i form a clique, only one configuration is possible: $B_1, \ldots, B_{|F_i|}$, with exactly one vertex in each block. The maximal number of configurations of F_i equals the number of partitions of a set with $|F_i|$ elements. When no edge exists between the boundary set vertices, all the partitions are to be considered.

The number of partitions of a set composed by i elements and j blocks,

Table 1: Classification of the partitions of sets containing from 1 to 4 elements

$\mathcal{A}_{i,j}$	j=1	j=2	j=3	j=4
i=1	**1** [1]			
i=2	**1** [12]	**2** [1][2]		
i=3	**1** [123]	**2** [13][2] **3** [1][23] **4** [12][3]	**5** [1][2][3]	
i=4	**1** [1234]	**2** [134][2] **3** [13][24] **4** [14][23] **5** [1][234] **6** [124][3] **7** [12][34] **8** [123][4]	**9** [14][2][3] **10** [1][24][3] **11** [1][2][34] **12** [13][2][4] **13** [1][23][4] **14** [12][3][4]	**15** [1][2][3][4]

written $A_{i,j}$, is given by the recursive formula of Stirling numbers of the second kind:

$$A_{i,j} = j \cdot A_{i-1,j} + A_{i-1,j-1}$$

with $A_{1,1} = 1$ and $A_{i,j} = 0$ if $i < j$.

The number $T(F_i)$ of different partitions of the boundary set F_i equals the sum of the $A_{|F_i|,j}$ for j from 1 to $|F_i|$.

$$T(F_i) = \sum_{j=1 \ to \ |F_i|} A_{|F_i|,j}$$

To identify the configurations of the boundary set, we associate to each one an ordering number between 1 and $T(F_i)$. The partitions of sets with at most four elements and their ordering number are reported in table 1.
Let $C(H_i, x)$ be the x^{th} configuration of F_i for the subgraph H_i. Its value, denoted $val(C(H_i, x))$, equals the minimum number of colors necessary to color H_i for this configuration.

In figure 2, two different colorings of H_5 correspond to the same configuration of F_5. Only 2 colors are necessary to color H_5 with the configuration for which vertices 2 and 3 have a same color, whereas 3 colors are necessary when vertices 2 and 3 have different colors. The value of the configuration $C(H_5, 1)$, represented by the partition [45], is 2, because we keep only the best valuation.

3.4 Coloring algorithm

The details of the implementation of the decomposition method are reported in algorithm 1. Note that $H_1 = (\{1\}, \emptyset)$ and $F_1 = \{1\}$. So, there is only one configuration of F_1, $C(H_1, 1) = [1]$. The insertion of the vertex 2 of G in $C(H_1, 1)$ can provide one or two configurations of F_2 (only one configuration if the vertices 1 and 2 are neighbors, two configurations otherwise).

At step i, we do not examine all the possible configurations of the step $i-1$, but only those which have been created at precedent step, it means those for which there is no edge between two vertices of the same block. For each configuration of F_{i-1}, we introduce the vertex i in each block successively. Each time the introduction is possible without breaking the coloring rules, the corresponding configuration of F_i is generated. We generate also the configurations obtained by adding to each configuration of F_{i-1} a new block containing the vertex i.

For a given subgraph H_i, only the configurations that are different are represented. Their ordering number x, included between 1 and $T(F_i)$, is computed by an algorithm according to their number of blocks and their number of elements. When different colorings of H_i correspond to the same configuration of F_i, only the best valuation is kept in $val(C(H_i, x))$.

At step N, only one configuration $C(H_N, 1)$ is generated from configurations of step $N-1$. It represents all the optimal coloring solutions and its value equals $\chi(G)$.

Example of configuration computing.
Assume that we are searching for a coloring of the graph G of figure 3 and that we are at step i with $F_i = \{u, v, w, i\}$.
Suppose that at step $i-1$, we had $F_{i-1} = \{u, v, w\}$ and that the configurations of F_{i-1} were:

- $C(H_{i-1}, 2) = [uw][v]$ with value α.
- $C(H_{i-1}, 4) = [uv][w]$ with value β.
- $C(H_{i-1}, 5) = [u][v][w]$ with value γ.

The values of α and β are at least 2, since the corresponding configurations have 2 blocks. Remark that these values may be upper than 2, depending on the configurations of the preceeding steps. By the same way, γ is at least 3.
We want to generate the configurations of F_i from the configurations of F_{i-1}.

- it is impossible to insert i in the first block of $C(H_{i-1}, 2)$, since u and i are neighbors. It is possible to insert i in the second block of $C(H_{i-1}, 2)$. We obtain $C(H_i, 3) = [uw][vi]$ and $val(C(H_i, 3)) = \alpha$. It is also possible to

Algorithm 1 Coloring algorithm

Input: a graph G

Output: χ : the chromatic number of G

```
H_1 = ({1}, ∅)
F_1 = {1}
C(H_1, 1) = [1]
for i = 2 to N do
  Build H_i and F_i
  for each configuration C(H_{i-1}, x) of F_{i-1} do
    for j = 1 to number of blocks of C(H_{i-1}, x) do
      if i does not have any neighbor in the block j then
        part = C(H_{i-1}, x)
        insert i in the block j of part
        generate the configuration C(H_i, y) corresponding to part
        if C(H_i, y) already exists then
          val(C(H_i, y)) = min(val(C(H_i, y)), val(C(H_{i-1}, x)))
        else
          val(C(H_i, y)) = val(C(H_{i-1}, x))
        end if
      end if
    end for
    part = C(H_{i-1}, x)
    add to part a new block containing i
    val(part) = max(val(C(H_{i-1}, x)), number of blocks of part)
    generate the configuration C(H_i, y) corresponding to part
    if C(H_i, y) already exists then
      val(C(H_i, y)) = min(val(C(H_i, y)), val(part))
    else
      val(C(H_i, y)) = val(part)
    end if
  end for
end for
χ = val(C(H_N, 1))
```

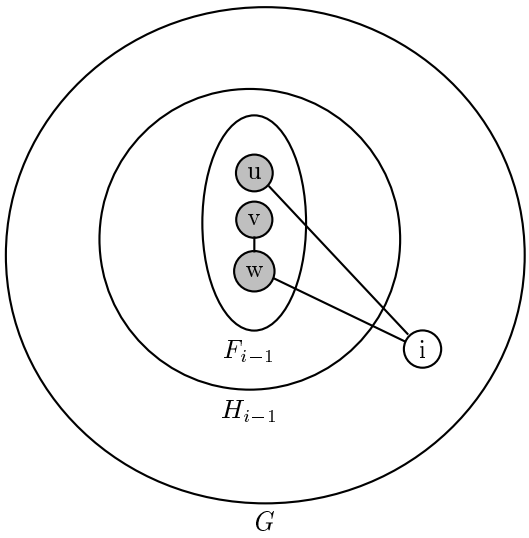


Figure 3: Construction of $H_i = (V_{i-1} \cup \{i\}, E_{i-1} \cup \{(u,i),(w,i)\})$

add a third block, it provides the configuration $C(H_i, 12) = [uw][v][i]$ with $valC(H_i, 12) = max(\alpha, 3)$, since for this configuration at least three colors are needed.

- we must add a block to introduce i in the configuration $C(H_{i-1}, 4)$. We obtain $C(H_i, 14) = [uv][w][i]$ with value $max(\beta, 3)$.

- i may be introduced in the second block of $C(H_{i-1}, 5)$. It provides the configuration $C(H_i, 10) = [u][vi][w]$ with value γ. By adding a new block, the configuration $C(H_i, 15)$ is created with $val(C(H_i, 15)) = max(\gamma, 4)$.

Thus five configurations are provided at step i, they are used to determine the configurations of the following step, and so on until the whole graph is colored.

3.5 Improved coloring algorithm

The number of configurations of a boundary set F_i is exponential according to $|F_i|$. Moreover, a coloring represented by a configuration of k blocks needs at least k colors. By applying a succession of k-colorings (see algorithm 2), we avoid examining the configurations of more than k blocks, which proves to be very interesting when F_{max} is larger than $\chi(G)$.

Take again the example of figure 3, and suppose that we are now searching for a 3-coloring of the graph G. All the configurations are made of at most three blocks, so their values are upper bounded by 3. This time, only four

Algorithm 2 k-coloring function

Input: a graph G and an integer k

Output: $Result : True$ if and only if G is k-colorable

```
H_1 = ({1}, ∅)
F_1 = {1}
C(H_1, 1) = [1]
i = 2
Result = True
while i ≤ N and Result do
  Result = False
  Build H_i and F_i
  for each configuration C(H_{i-1}, x) of F_{i-1} do
    for j = 1 to number of blocks of C(H_{i-1}, x) do
      if i does not have any neighbor in the block j then
        Result = True
        part = C(H_{i-1}, x)
        insert i in the block j of part
        generate the configuration C(H_i, y) corresponding to part
        if C(H_i, y) already exists then
          val(C(H_i, y)) = min(val(C(H_i, y)), val(C(H_{i-1}, x)))
        else
          val(C(H_i, y)) = val(C(H_{i-1}, x))
        end if
      end if
    end for
    if number of blocks of C(H_{i-1}, x) < k then
      Result = True
      part = C(H_{i-1}, x)
      add to part a new block containing i
      val(part) = max(val(C(H_{i-1}, x)), number of blocks of part)
      generate the configuration C(H_i, y) corresponding to part
      if C(H_i, y) already exists then
        val(C(H_i, y)) = min(val(C(H_i, y)), val(part))
      else
        val(C(H_i, y)) = val(part)
      end if
    end if
  end for
  i = i + 1
end while
```

Algorithm 3 Linear Decomposition Coloring Algorithm (LDC)

Input: a graph G

Output: k : the chromatic number of G

```
Determine a lower bound LB of k
Determine an upper bound UB of k
while (LB ≠ UB) do
  set the value of k between LB and UB (*)
  result=k-coloring(G)
  if result = false then
    LB = k + 1
  else
    UB = k
  end if
end while
```

(*) $Kdic : k = \lfloor (LB + UB)/2 \rfloor$, $Kseq : k = LB + 1$

configurations are provided at step i:
$C(H_i, 3) = [uw][vi]$ with value 2 or 3, $C(H_i, 12) = [uw][v][i]$ with value 3, $C(H_i, 14) = [uv][w][i]$ with value 3, and $C(H_i, 10) = [u][vi][w]$ with value 3.
We cannot add a new block to $C(H_{i-1}, 5)$, because it has already got three blocks.
The gain may obviously be more significant when the size of the boundary set increases. Indeed, let $T'(F_i)$ be the maximum number of partitions of F_i which have at most k blocks:

$$T'(F_i) = \sum_{j=1 \ to \ k} A_{|F_i|,j}$$

When no edge exists between the vertices of F_i, the gain $Gain(F_i)$ for this step equals $T(F_i)$ minus $T'(F_i)$:

$$Gain(F_i) = \sum_{j=k+1 \ to \ |F_i|} A_{|F_i|,j}$$

In the improved Linear Decomposition Coloring algorithm (LDC), we start by determining a lower bound LB of the chromatic number of G (see algorithm 3). For that, we apply on G some heuristics based on triangulated graphs. Indeed, it is easy to compute the chromatic number of a triangulated graph [14]. The used lower bound functions as follows: as long as the graph is not triangulated, we remove a vertex of smallest degree, and then we color the remaining triangulated graph by determining a perfect elimination order [14] on the vertices

of G. The upper bound UB is obtained by applying the heuristic DSATUR [3]. Then we apply a succession of k-colorings, k varying between LB and UB. We consider two different versions of LDC, depending on the way the value of k is set: by dichotomy, denoted $Kdic\text{-}LDC$, or sequentially growing from LB to UB, denoted $Kseq\text{-}LDC$. In theory, $Kdic\text{-}LDC$ has a better complexity than $Kseq\text{-}LDC$. Nevertheless, in practice $Kseq\text{-}LDC$ gives good lower bounds of the chromatic number of hard instances that $Kdic\text{-}LDC$ is unable to solve.

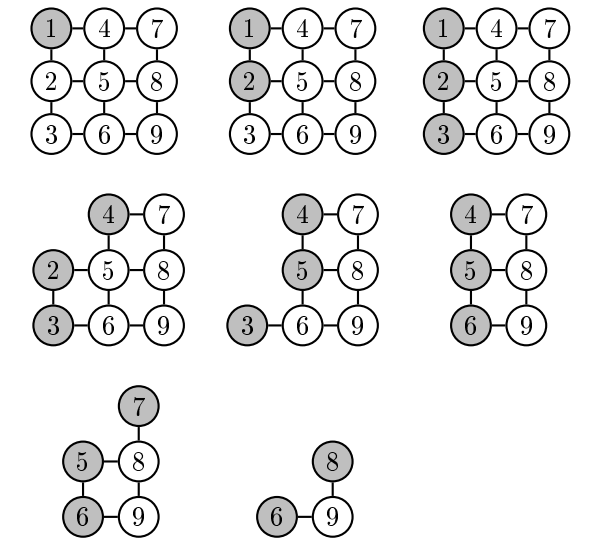
Algorithm 3 does not produce directly a coloring, nevertheless it can be obtained easily. All the configurations generated during the k-coloring have to be stored in memory. The color 1 is assigned to vertex N. When a k-coloring is found, a configuration of F_{N-1} corresponding to the configuration of F_N is chosen, a color is assigned to vertex $N-1$, and so on until vertex 1 is colored. Assign colors to the vertices needs a lot of memory space but does not increase the time complexity of the algorithm.

3.6 The vertex linear ordering

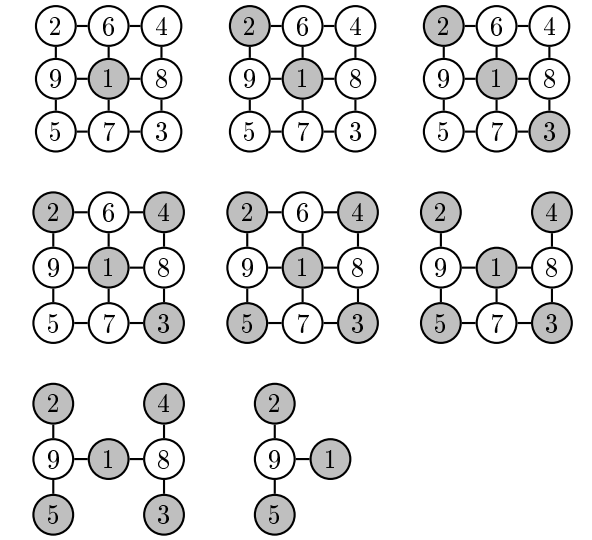
The maximum size of the boundary sets $F_{max}(\mathcal{N})$ depends on the vertices numbering chosen (see figure 4). It corresponds to the linearwidth of the vertex linear ordering used to build the different subgraphs H_i (cf section 2.3).

The complexity of the linear-decomposition is exponential with respect to $F_{max}(\mathcal{N})$, so it is necessary to make a good choice when numbering the vertices of the graph. Unfortunately, finding an optimal vertex linear ordering in order to obtain the smallest linearwidth is a NP-complete problem [1]. A first method to reduce the size of the boundary sets is to number the vertices by applying a double BFS [14] on the graph. It produces the numbering $\mathcal{N}_1$. An other method is to begin the numbering from a clique and then order the vertices by decreasing number of edges with already numbered vertices. It produces the numbering $\mathcal{N}_2$. The aim of this method is to give the smallest numbers as possible to the vertices of boundary sets of size $F_{max}(\mathcal{N}_2)$. To compare the linearwidths of these numberings with the linearwidth of the initial vertex numbering $\mathcal{N}_0$, we performed tests on random graphs $G_{N,d}$ (graphs with N vertices and with density d) and on COLOR02 computational symposium instances. Some representative results are reported in Table 2. For random graphs, results over average of 5 instances are given.

The numbering $\mathcal{N}_2$ gave smaller $F_{max}(\mathcal{N})$ for more instances than the other numberings, so we use this ordering in experiments of section 4. Despite this, the number of configurations increases sometimes very strongly when introducing a



$F_{max} = 3$



$F_{max} = 5$

Figure 4: The value of F_{max} is a consequence of the initial vertex numbering.

Table 2: Comparison of different vertex linear orderings

Problem	$F_{max}(\mathcal{N}_0)$	$F_{max}(\mathcal{N}_1)$	$F_{max}(\mathcal{N}_2)$	Problem	$F_{max}(\mathcal{N}_0)$	$F_{max}(\mathcal{N}_1)$	$F_{max}(\mathcal{N}_2)$
G(30,.1)	12.4	9.6	**8.4**	G(40,.5)	34.0	34.4	**33.6**
G(40,.1)	20.8	16.4	**15.0**	G(50,.5)	43.4	43.4	**41.8**
G(50,.1)	29.4	26.2	**22.0**	G(60,.5)	53.8	53.8	**52.4**
G(60,.1)	36.8	32.8	**28.0**	G(30,.9)	27.8	27.8	**27.2**
G(70,.1)	44.0	43.0	**34.2**	G(40,.9)	37.8	37.8	**36.4**
G(80,.1)	55.0	50.8	**42.8**	G(50,.9)	47.4	47.4	**46.8**
G(90,.1)	63.6	60.0	**51.8**	G(60,.9)	57.8	57.8	**57.0**
G(30,.5)	24.8	24.6	**22.6**	G(70,.9)	67.6	67.4	**67.0**
mug88_1	20	13	**8**	mug88_25	20	13	**8**
mug100_1	17	22	**7**	mug100_25	19	16	**8**
1-FullIns4	61	71	**46**	1-FullIns5	187	216	**132**
2-FullIns3	37	28	**22**	2-FullIns4	157	105	**93**
2-FullIns5	637	417	**359**	3-FullIns3	61	45	**36**
3-FullIns4	321	247	**200**	4-FullIns3	91	57	**49**
4-FullIns4	571	332	**302**	5-FullIns3	127	127	**64**
5-FullIns4	925	772	**447**	1-Inser_4	**31**	40	32
1-Inser_5	98	145	**91**	1-Inser_6	300	406	**276**
2-Inser_4	**48**	56	52	2-Inser_5	**197**	251	240
3-Inser_3	**13**	**13**	16	3-Inser_4	**69**	92	84
3-Inser_5	**350**	578	436	4-Inser_3	**15**	**15**	20
4-Inser_4	**94**	120	124	miles250	67	48	**16**
le450_5a	381	368	**339**	le450_5b	386	382	**343**
le450_5c	397	401	**385**	le450_5d	400	403	**385**
le450_15a	373	365	**339**	le450_15b	378	372	**336**
le450_15c	420	418	**396**	le450_15d	416	415	**401**
le450_25a	351	348	**293**	le450_25c	416	412	**394**
le450_25d	411	409	**384**				

vertex with too few neighbors in the boundary set. To avoid this case we add a reduction on the graph before each k-coloring.

3.7 Vertex reduction

Before each k-coloring, we delete some vertices of the graph by using the following property: for each vertex x, if the degree of x is strictly lower than k, x and its adjacent edges can be erased from the graph [13]. Indeed, assume x has $k-1$ neighbors. In the worst case, those neighbors must have different colors. Then the vertex x can take the k^{th} color. It does not interfere in the coloring of the remaining vertices because all its neighbors have already been colored. Therefore we can consider from the beginning that it will take a color unused by its neighbors and delete it from the graph before the coloring. We apply this principle recursively by examining the remaining vertices until having totally reduced the graph or being enable to delete any other vertex.

4 Experimental results

Our LDC algorithms have been implemented on a PC AMD Athlon Xp 2000+ in C language. We performed tests on random graphs and on benchmark instances used at the computational symposium COLOR02.

4.1 Random graphs

Random graphs $G_{N,d}$ have been created. For each graph, N is the number of vertices and d its density. We generated graphs with densities 0.1, 0.5 and 0.9. We give the results of our experiments over average of 5 instances in Table 3. For each type of graph, we give the number of edges M and the values LB and UB of our initial bounds. Algorithms $Kdic-LDC$ and $Kseq-LDC$ have been applied to each instance, using a CPU time limit of half an hour. For each algorithm, we give the average chromatic number and the CPU time when the program is able to compute. Otherwise, when less than 5 instances have been solved, we indicate the number of solved instances in brackets. Starting from $N = 30$, we increase the size of random graphs by step 10 until less than 3 instances are solved.

Our algorithms are not very efficient on random graphs in comparison with the results of other exact methods [21, 15, 7]. Indeed, Desrosiers, Galinier and Hertz are able to solve graphs with density 0.5 and with up to 100 vertices. This is mainly due to the fact that edges repartition in the graph is homogeneous,

Table 3: Results on random graphs

				LDC			
Problem	M	LB	UB	$Kdic$	T	$Kseq$	T
G(30,.1)	43.8	2.8	3.0	3.0	0.00	3.0	0.00
G(40,.1)	75.8	3.0	3.6	3.2	0.00	3.2	0.00
G(50,.1)	123.2	3.0	4.2	4.0	0.00	4.0	0.78
G(60,.1)	177.0	3.0	4.6	4.0[2]	0.00	4.0	2.54
G(70,.1)	237.8	3.0	5.0			4.0	2.66
G(80,.1)	320.4	3.0	5.0			4.2[4]	103.16
G(90,.1)	397.0	3.2	5.6			4.6[3]	104.80
G(100,.1)	477.2	3.0	5.6			5.0[4]	190.99
G(110,.1)	604.4	3.8	6.0			5.0[4]	91.90
G(120,.1)	711.8	3.2	6.4			[0]	*
G(30,.5)	211.2	5.2	8.2	7.0	11.18	7.0	2.92
G(40,.5)	380.0	6.2	9.8	8.2	6.19	8.2	6.16
G(50,.5)	608.6	6.2	12.0	6.0[3]	97.93	9.4[3]	83.12
G(60,.5)	881.8	6.6	12.8	[0]	*	[0]	*
G(30,.9)	392	15.2	16.2	15.8	0.02	15.8	0.01
G(40,.9)	699.8	18.2	22.6	18.8	7.11	18.8	0.44
G(50,.9)	1103.6	20.8	25.4	22.8	13.2	22.8	10.79
G(60,.9)	1606.6	24.0	30.6	[0]	*	26.2[4]	105.02
G(70,.9)	2172.6	25.0	33.4			28[1]	684.8

inducing a large linearwidth. For a graph with 50 vertices, we have (see Table 2) $F_{max}(\mathcal{N}_2) = 41$ when the graph density is 0.5 and $F_{max}(\mathcal{N}_2) = 46$ when the graph density is 0.9. Nevertheless, results of Table 3 reflect clearly the possibilities and the limits of our method. We are able to solve graphs with density 0.9 and 60 vertices. These graphs have a very large linearwidth but are also very constrained : at each step only few configurations are generated despite the big size of the boundary sets. The decomposition method is not efficient on random graphs with density 0.5. Indeed, the large size of the linearwidth induces the possibility to generate an exponential number of configurations and the 0.5 density does not limit enough the number of valid configurations. Our linear-decomposition method works better on graphs with small density. Indeed, the maximum size of the boundary set can be very reduced by the vertex numbering $\mathcal{N}_2$. For this kind of random graphs, we are able to solve instances with up to 100 vertices.

4.2 DIMACS and COLOR02 instances

We performed tests on benchmark instances used at the computational symposium COLOR02, among which well-known DIMACS instances (see description of the instances at http://mat.gsia.cmu.edu/COLOR02).

We first apply on each instance the lower bound and upper bound described in section 3.5. Results are reported in table 4. For each graph we give the number of vertices N, the number of edges M, and the values LB and UB of our initial bounds. At this stage, the lower bound equals the upper bound for some of the DIMACS instances, so we did not apply any k-coloring algorithm on these graphs.

We compare our LDC algorithms with the recent results of the exact Branch-and-Cut algorithm of Mendez Diaz and Zabala [9].We use a CPU time limit of half an hour. The results of our experiments are reported in table 5. The time column for B&C is only indicative since the results are from different machines. They used C++ code using ABACUS framework and CPLEX 6.0 LP solver on a Sun ULTRA workstation. Their CPU time limit was two hours. Comparing their time results for the same implementation of $Dsat$ that we use, our computer seems to run three times faster than their computer. We report also the results of Desrosiers, Galinier and Hertz [7] in VCS columns. Column *btk* contains the number of backtracks needed by their exact algorithm to solve an instance reduced to a vertex-critical subgraph. Values are enclosed in parentheses when their algorithm is not able to find the chromatic number

Table 4: Lower bound and Upper bound results

Problem	N	M	LB	UB	Problem	N	M	LB	UB
fpsol2.i.1	496	11654	**65**	**65**	1-FullIns3	30	100	3	5
fpsol2.i.2	451	8691	**30**	**30**	1-FullIns4	93	593	3	6
fpsol2.i.3	425	8688	**30**	**30**	1-FullIns5	282	3247	3	8
inithx.i.1	864	18707	**54**	**54**	2-FullIns3	52	201	4	5
inithx.i.2	645	13979	**31**	**31**	2-FullIns4	212	1621	4	6
inithx.i.3	621	13969	**31**	**31**	2-FullIns5	852	12201	4	7
le450-5a	450	5714	5	11	3-FullIns3	80	346	5	6
le450-5b	450	5734	5	11	3-FullIns4	405	3524	2	7
le450-5c	450	9803	4	13	4-FullIns3	114	541	6	7
le450-5d	450	9757	4	12	4-FullIns4	690	6650	2	8
le450-15a	450	8168	13	18	5-FullIns3	154	792	7	8
le450-15b	450	8169	15	17	5-FullIns4	1085	11395	2	9
le450-15c	450	16680	11	25	1-Inser_4	67	232	2	5
le450-15d	450	16750	11	26	1-Inser_5	202	1227	2	6
le450-25a	450	8260	20	25	1-Inser_6	607	6337	2	7
le450_25b	450	8263	**25**	**25**	2-Inser_3	37	72	2	4
le450-25c	450	17343	22	30	2-Inser_4	149	541	2	5
le450-25d	450	17425	19	29	2-Inser_5	597	3936	2	6
mulsol.i.1	197	3925	**49**	**49**	3-Inser_3	56	110	2	4
mulsol.i.2	188	3885	**31**	**31**	3-Inser_4	281	1046	2	5
mulsol.i.3	184	3916	**31**	**31**	3-Inser_5	1406	9695	2	6
mulsol.i.4	185	3946	**31**	**31**	4-Inser_3	79	156	2	4
mulsol.i.5	186	3973	**31**	**31**	4-Inser_4	475	1795	2	5
school1	385	19095	**14**	**14**	mug100-1	100	166	3	4
school1_nsh	352	14612	**14**	**14**	mug100-25	100	166	3	4
zeroin.i.1	211	4100	**49**	**49**	mug88-1	88	146	3	4
zeroin.i.2	211	3541	**30**	**30**	mug88-25	88	146	3	4
zeroin.i.3	206	3540	**30**	**30**	games120	120	638	**9**	**9**
anna	138	493	**11**	**11**	miles250	128	387	7	8
david	87	812	**11**	**11**	miles500	128	2340	**20**	**20**
homer	561	1629	**13**	**13**	miles750	128	4226	**31**	**31**
huck	74	602	**11**	**11**	miles1000	128	6432	**42**	**42**
jean	80	508	**10**	**10**	miles1500	128	10396	**73**	**73**

within the time limit.

The F_{max} column indicates for each graph the maximal value of the boundary set reached by our algorithm. We give the chromatic number of the graph and the CPU time when the program is able to compute. Otherwise, asterisks in the *time* column indicate that the program exceeded the time limit. In this case, we report the best lower and upper bounds of the chromatic number obtained by the algorithm.

The results of $Kseq\text{-}LDC$ are equivalent or better than those of $Kdic\text{-}LDC$ on the tested instances, that is why we report only $Kseq\text{-}LDC$ results. Our algorithm runs very fast on *mug* instances (0.0 or 0.1 second), because the vertex linear ordering chosen provides small linearwidths for these graphs (F_{max} column). For these instances, the results of $Kdic\text{-}LDC$ and $Kseq\text{-}LDC$ are the same, since $UB = LB + 1$. B&C and VCS found the chromatic number, but spent time to explore the Branch-and-Cut tree. Our algorithms are also able to solve some instances which have a large linearwidth, because all the partitions of the boundary set have not necessarily to be generated. We found the chromatic number of 13 instances, 12 of which solved in less than 30 seconds. The LDC algorithm is the first to our knowledge to find the chromatic number of the instance 4-$Inser\_3$ (bolded in table 5) with an exact method.

5 Conclusions

In this paper, we have presented an original method to solve the graph coloring problem by an exact way. This method has the advantage of solving easily large instances which have a small linearwidth, such as *mug* instances, and allows us to solve the difficult instance 4-$Inser\_3$. We consider using the linear-decomposition mixed with heuristics approach to deal with unbounded linearwidth instances.

References

[1] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[2] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[3] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, april 1979.

Table 5: Results on COLOR02 instances

Problem	LDC			B&C		VCS	
	F_{max}	k	T	k	T	k	btk
mug88_1	8	4	0.0	4	485	4	2204467
mug88_25	8	4	0.0	4	1690	4	942961
mug100_1	7	4	0.1	4	4029	4	1406570
mug100_25	8	4	0.0	4	5498	4	974170
1-FullIns4	27	5	7.3	5	703	5	6
1-FullIns5	26	4-6	*	4-6	*	6	271
2-FullIns3	9	5	0.0	5	3	5	1
2-FullIns4	19	5-6	*	5-6	*	6	8
2-FullIns5	20	5-7	*	5-7	*	7	715
3-FullIns3	11	6	0.0	6	1	6	0
3-FullIns4	20	3-7	*	6-7	*	7	10
4-FullIns3	11	7	0.0	7	3	7	1
4-FullIns4	17	4-8	*	7-8	*	8	12
5-FullIns3	13	8	0.0	8	3	8	1
5-FullIns4	17	3-9	*	7-9	*		
1-Inser_4	21	4-5	*	2-5	*	5	104296036
1-Inser_5	20	3-6	*	4-6	*	*	(133727661)
1-Inser_6	18	3-7	*	4-7	*	*	(50929137)
2-Inser_4	28	3-5	*	4-5	*	*	(154902785)
2-Inser_5	18	3-6	*	3-6	*	*	(48458541)
3-Inser_3	16	4	23.3	4	10	4	723616
3-Inser_4	28	3-5	*	3-5	*	*	(95076991)
3-Inser_5	17	3-6	*	3-6	*	*	(13784327)
4-Inser_3	**20**	**4**	**1774**	**3-4**	*****	*****	**(228367528)**
4-Inser_4	27	3-5	*	3-5	*	*	(70891706)
miles250	9	8	0.0			8	0
le450_5a	15	5-9	*	5-9	*	5	0
le450_5b	16	5-9	*	5-9	*	5	0
le450_5c	22	5-6	*			5	0
le450_5d	46	5-7	*	5-10	*	5	0
le450_15a	21	15-18	*	15-17	*	15	0
le450_15b	21	15-17	*	15-17	*	15	0
le450_15c	21	15-25	*	15-24	*	15	0
le450_15d	21	15-26	*	15-23	*	15	0
le450_25a	25	25	0.0			25	0
le450_25c	30	25-28	*	25-28	*	25	0
le450_25d	30	25-28	*	25-28	*	25	0

[4] J. Carlier and C. Lucet. A decomposition algorithm for network reliability evaluation. *Discrete Appl. Math.*, 65:141–156, 1996.

[5] D. de Werra. An introduction to timetabling. *European Journal of Operation Research*, 19:151–162, 1985.

[6] D. de Werra. On a multiconstrained model for chromatic scheduling. *Discrete Appl. Math.*, 94:171–180, 1999.

[7] C. Desrosiers, P. Galinier, and A. Hertz. Efficient algorithms for finding critical subgraphs. *Cahiers du GERAD*, G-2004-31, 2004.

[8] I. Mendez Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. In *Computational Symposium on Graph Coloring and its Generalizations (COLOR02)*, Ithaca, N-Y, september 2002.

[9] I. Mendez Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Optimization Online: Combinatorial Optimization*, september 2003.

[10] N. Funabiki and T. Higashino. A minimal-state processing search algorithm for graph coloring problems. *IEICE Transactions on Fundamentals*, E83-A(7):1420–1430, 2000.

[11] P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[13] F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In Trick and Johnson [28], pages 285–308.

[14] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[15] F. Herrmann and A. Hertz. Finding the chromatic number by means of critical graphs. *ACM Journal of Experimental Algorithmics*, 7(10):1–9, 2002.

[16] A. Hertz and D. De Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[17] M. Kubale and B. Jackowski. A generalized implicit enumeration algorithm for graph coloring. *Communications of the ACM*, 28(4):412–418, 1985.

[18] C. Lucet. *Méthode de décomposition pour l'évaluation de la fiabilité des réseaux.* PhD thesis, Université de Technologie de Compiègne, 1993.

[19] J.F. Manouvrier. *Méthode de décomposition pour résoudre des problèmes combinatoires sur les graphes.* PhD thesis, Université de Technologie de Compiègne, 1998.

[20] B. Manvel. Extremely greedy coloring algorithms. In F. Harary and J.S. Maybee, editors, *Graphs and applications: Proceedings of the First Colorado Symposium on Graph Theory*, pages 257–270, New York, 1985. John Wiley & Sons.

[21] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[22] C. A. Morgenstern. Distributed coloration neighborhood search. In Trick and Johnson [28], pages 335–357.

[23] N. Robertson and P. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[24] D. G. Corneil S. Arnborg and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[25] T. J. Sager and S. Lin. A pruning procedure for exact graph coloring. *ORSA Journal on Computing*, 3:226–230, 1991.

[26] S. Sen Sarma and S. K. Bandyopadhyay. Some sequential graph colouring algorithms. *International Journal of Electronic*, 67(2):187–199, 1989.

[27] E. Sewell. An improved algorithm for exact graph coloring. In Trick and Johnson [28], pages 359–373.

[28] Michael A. Trick and David S. Johnson, editors. *Cliques, Coloring, and Satisfiability: Proceedings of the Second DIMACS Implementation Challenge*. American Mathematical Society, 1993.