Pre-processings and Linear-Decomposition Algorithm to Solve the k-Colorability Problem*

C. Lucet[1], F. Mendes[1], A. Moukrim[2]

[1] LaRIA EA 2083, 5 rue du Moulin Neuf 80000 Amiens, France
[2] HeuDiaSyC UMR CNRS 6599 UTC, BP 20529 60205 Compiègne, France
(Corinne.Lucet, Florence.Mendes)@laria.u-picardie.fr
Aziz.Moukrim@hds.utc.fr

Abstract. We are interested in the graph coloring problem. We studied the effectiveness of some pre-processings that are specific to the k-colorability problem and that promise to reduce the size or the difficulty of the instances. We propose to apply on the reduced graph an exact method based on a linear-decomposition of the graph. We present some experiments performed on literature instances, among which DIMACS library instances.

1 Introduction

The Graph Coloring Problem constitutes a central problem in a lot of applications such as school timetabling, scheduling, or frequency assignment [5, 6]. This problem belongs to the class of NP-hard problems [10]. Various heuristics approaches have been proposed to solve it (see for instance [2, 8, 9, 11, 13, 17, 19, 21]). Efficient exact methods are less numerous: implicit enumeration strategies [14, 20, 22], column generation and linear programming [18], branch-and-bound [3], branch-and-cut [7], without forgetting the well-known exact version of Brelaz's DSATUR [2].

A *coloring* of a graph $G = (V, E)$ is an assignment of a color $c(x)$ to each vertex such that $c(x) \neq c(y)$ for all edges $(x, y) \in E$. If the number of colors used 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. Our approach to solve this problem is to solve for different values of k the *k-colorability problem*: "does there exist a k-coloring of G ?".

We propose to experiment the effectiveness of some pre-processings that are directly related to the k-colorability problem. The aim of these processings is to reduce the size of the graph by deleting vertices and to constrain it by adding edges. Then we apply a linear-decomposition algorithm on the reduced graph in order to solve the graph coloring problem. This method is strongly related to notions of tree-decomposition and path-decomposition, well studied by Bodlaender [1]. Linear-decomposition has been implemented efficiently by Carlier,

* with the support of Conseil Régional de Picardie and FSE

Lucet and Manouvrier to solve various NP-hard problems [4, 15, 16] and has for main advantage that the exponential factor of its complexity depends on the linearwidth of the graph but not on its size.

Our paper is organized as follows. We present in Sect. 2 some pre-processings related to the k-colorability problem and test their effectiveness on various benchmark instances. In Sect. 3, we describe our linear-decomposition algorithm. We report the results of our experiments in Sect. 4. Finally, we conclude and discuss about the perspectives of this work.

2 Pre-processings

In this section, we present several pre-processings to reduce the difficulty of a k-colorability problem. These pre-processings are iterated until the graph remains unchanged or the whole graph is reduced.

2.1 Definitions

An *undirected graph* G is a pair (V, E) made up of a vertex set V and an edge set $E \subset V \times V$. Let $N = |V|$ and $M = |E|$. 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 we will consider in the following of this paper will be only undirected and connected graphs. Given a graph $G = (V, E)$ and a vertex $x \in V$, let $\vartheta(x) = \{y \in V / (x, y) \in E\}$. $\vartheta(x)$ represents the neighborhood of x in G. The *subgraph* of $G = (V, E)$ induced by $I \subseteq V$, is the graph $G(I) = (I, E_I)$ such that $E_I = E \cap (I \times I)$. A *clique* of $G = (V, E)$ is a subset $C \subseteq V$ such that every two vertices in C are joined by an edge in E. Let $\overline{E} = (V \times V) \setminus E$ be the set made up of all pairs of vertices that are not neighbors in $G = (V, E)$. Let d be the degree of G, i.e. the maximal vertex degree among all vertices of G.

2.2 Reduction 1

A vertex reduction using the following property of the neighborhood of the vertices can be applied to the representative graph before any other computation with time complexity $O(|\overline{E}| * d)$, upper bounded by $O(N^3)$. Given a graph G, for each pair of vertices $x, y \in V$ such that $(x, y) \notin E$, if $\vartheta(y) \subseteq \vartheta(x)$ then y and its adjacent edges can be erased from the graph. Indeed, suppose that $k - 1$ colors are needed to color the neighbors of x. The vertex x can take the k^{th} color. Vertices x and y are not neighbors. Moreover, the neighbors of y are already colored with at most $k - 1$ colors. So, if $G \setminus \{y\}$ is k-colorable then G is k-colorable as well and we can delete y from the graph. This principle can be applied recursively as long as vertices are removed from the graph.

2.3 Reduction 2

Suppose that we are searching for a k-coloring of a graph $G = (V, E)$. Then we can use the following property: for each vertex x, if the degree of x is strictly

lower than k, x and its edges can be erased from the graph [11]. 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. The time complexity of this reduction is $O(N)$. We apply this principle recursively by examining the remaining vertices until having totally reduced the graph or being enable to delete any other vertex.

2.4 Vertex Fusion

Suppose that we are searching for a k-coloring of $G = (V, E)$ and that a clique C of size k has been previously determined. For each couple of non-adjacent vertices $x, y \in V$ such that $x \notin C$ and $y \in C$, if x is adjacent to all vertices of $C \setminus y$ then x and y can be merged by the following way: each neighbor of x becomes a neighbor of y, then x and its adjacent edges are erased from the graph. Indeed, since we are searching for a k-coloring, x and y must have the same color. Then $\forall z \in \vartheta(x)\ c(y) \neq c(z)$ and the edge (y, z) can be added to G. Then $\vartheta(x) \subseteq \vartheta(y)$ and x can be erased from the graph (cf Sect. 2.2). The time complexity of this pre-processing is $O(N * k)$.

2.5 Edge Addition

Suppose that we are searching for a k-coloring of $G = (V, E)$ and that a clique C of size k has been previously determined. For each couple of non-adjacent vertices $x, y \in V$, if $\forall z \in C$ we have $(x, z) \in E$ or $(y, z) \in E$, then the edge (x, y) can be added to the graph. Necessarly, x must take a color from the colors of $C \setminus \vartheta(x)$. Since $\vartheta(y) \supseteq C \setminus \vartheta(x)$, $c(x) \neq c(y)$. This constraint can be represented by an edge between x and y. The time complexity of this pre-processing is $O(|\overline{E}| * k)$, upper bounded by $O(N^2 * k)$.

Algorithm 1 Pre-processings

Input: a graph G and an integer k
Output: a graph G' k-colorable if and only if G is k-colorable
  repeat
    reduction 1
    reduction 2
    if $\exists$ at least 1 clique of size k **then**
      apply vertex fusion and edge addition on G
    end if
  until there is no more change in G
  $G' = G$

2.6 Pre-processing Experiments

Our algorithms have been implemented on a PC AMD Athlon Xp 2000+ in C language. The method used is as follows. To start with, we apply on the entry graph G a fast clique search algorithm: 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 [12] on the vertices of G. The size of the clique provided by this algorithm, denoted LB, constitutes a lower bound of the chromatic number of G. Then we apply on G the pre-processings described in Algorithm 1, supposing that we are searching for a k-coloring of the graph with $k = LB$. We performed tests on benchmark instances used at the computational symposium COLOR02, including well-known DIMACS instances (see description of the instances at http://mat.gsia.cmu.edu/COLOR02). Results are reported in Table 1. For each graph, we indicate the initial number of vertices N and the number of edges M. The column LB contains the size of the maximal clique found. The percentage of vertices deleted by the pre-processings is reported in column Del. The number of remaining vertices after the pre-processing step is reported in column new_N. Remark that some of the instances are totally reduced by the pre-processings when $k = LB$, and that some of them are not reduced at all.

Table 1: Pre-processings results

Graph	N	M	LB	new_N	Del	Graph	N	M	LB	new_N	Del
1-FullIns3	30	100	3	15	50%	1-FullIns4	93	593	3	35	62%
1-FullIns5	282	3247	3	75	73%	2-FullIns3	52	201	4	9	81%
2-FullIns4	212	1621	4	41	81%	2-FullIns5	852	12201	4	89	90%
3-FullIns3	80	346	5	11	86%	3-FullIns4	405	3524	2	51	87%
3-FullIns5	2030	33751	2	107	95%	4-FullIns3	114	541	6	13	89%
4-FullIns4	690	6650	2	58	92%	5-FullIns3	154	792	7	15	90%
5-FullIns4	1085	11395	2	65	94%	fpsol2.i.1	496	11654	65	228	54%
fpsol2.i.2	451	8691	30	175	61%	fpsol2.i.3	425	8688	30	149	65%
inithx.i.1	864	18707	54	443	49%	inithx.i.2	645	13979	31	215	67%
inithx.i.3	621	13969	31	190	69%	mulsol.i.1	197	3925	49	60	70%
mulsol.i.2	188	3885	31	88	53%	mulsol.i.3	184	3916	31	83	55%
mulsol.i.4	185	3946	31	85	54%	mulsol.i.5	186	3973	31	84	55%
school1	385	19095	14	360	6%	school1_nsh	352	14612	14	331	6%
3-Inser_3	56	110	2	56	0%	4-Inser_3	79	156	2	79	0%
le450_25a	450	8260	20	297	34%	le450_25b	450	8263	25	294	35%
anna	138	493	11	0	100%	david	87	812	11	0	100%
homer	561	1629	13	0	100%	jean	80	508	10	0	100%
mug100-1	100	166	3	100	0%	mug100-25	100	166	3	100	0%
mug88-1	88	146	3	88	0%	mug88-25	88	146	3	88	0%
miles250	128	387	7	34	73%	miles500	128	2340	20	0	100%
miles750	128	4226	31	0	100%	miles1000	128	6432	42	0	100%
miles1500	128	10396	73	0	100%	DSJR500_1	500	3555	12	28	94%
zeroin.i.1	211	4100	49	86	59%	zeroin.i.2	211	3541	30	55	74%
zeroin.i.3	206	3540	30	50	76%	games120	120	638	9	0	100%

3 Linear-Decomposition Applied to the k-Colorability Problem

In this section, we propose a method which uses linear-decomposition mixed with Dsatur heuristic in order to solve the k-colorability problem.

3.1 Definitions

We will consider a graph $G = (V, E)$. Let $N = |V|$ and $M = |E|$. A *vertex linear ordering* of G is a bijection $\mathcal{N} : V \rightarrow \{1, \ldots, N\}$. For more clarity, we denote i the vertex $\mathcal{N}^{-1}(i)$. 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. Let $F_i = \{j \in V / \exists (j, l) \in E \ j \leq i \ < l\} \ \forall i \in \{1, \ldots, |V|\}$. F_i is the *boundary set* of H_i. Let $H'_i = (V'_i, E'_i)$ be the subgraph of G such that $V'_i = (V \setminus V_i) \cup F_i$ and $E'_i = E \cap (V'_i \times V'_i)$. The boundary set F_i corresponds to the set of vertices joining H_i to H'_i (see Fig. 1).

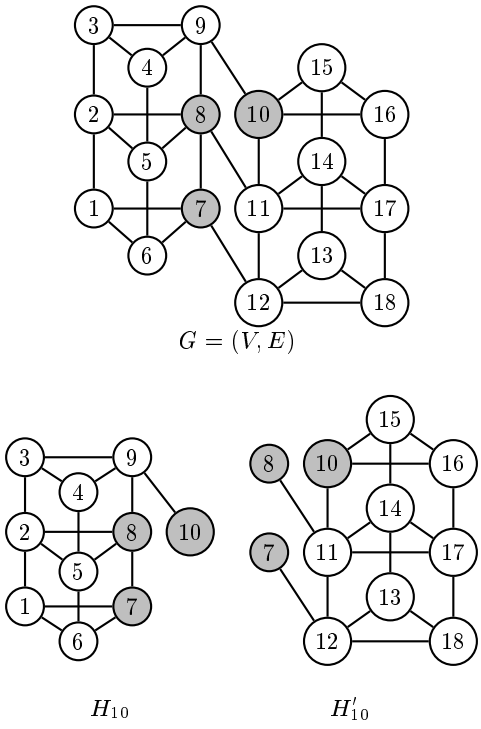

Fig. 1. A subgraph H_{10} of G and its boundary set $F_{10} = \{7, 8, 10\}$

The *linearwidth* of a vertex linear ordering $\mathcal{N}$ is $F_{max}(\mathcal{N}) = max_{i \in V}(|F_i|)$. We use a vertex linear ordering of the graph to resolve the k-colorability problem

with a linear-decomposition. The resolution method is based on a sequential insertion of the vertices, using a vertex linear ordering previously determined. This will be developed in the following section.

3.2 Linear-Decomposition Algorithm

The details of the implementation of the linear-decomposition method are reported in Algorithm 2. The vertices of G are numbered according to a linear ordering $\mathcal{N} : V \rightarrow \{1, \ldots, N\}$. Then, during the coloring, we will consider N subgraphs $H_1, \ldots, H_N$ and the N corresponding boundary sets $F_1, \ldots, F_N$, as defined in Sect. 3.1.

Algorithm 2 k-colorability

Input: a graph G and an integer k
Output: $Result : True$ if and only if G is k-coloriable

```
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
        val(C(H_i, y)) = min(val(C(H_i, y)), val(C(H_{i-1}, x)))
      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
      val(C(H_i, y)) = min(val(C(H_i, y)), val(part))
    end if
  end for
  i = i + 1
end while
```

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]. After some experiments on various heuristics of vertex numbering, we choose to begin the numbering from the biggest clique provided by our clique search heuristic (cf Sect. 2.6). Then we order the vertices by decreasing number of already numbered neighbors.

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. To 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 whose neighbors have all been numbered with at most i. Several colorings of H_i may correspond to the same coloring of F_i. 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.

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 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 possible partitions of a set with $|F_i|$ elements. When no edge exists between the boundary set vertices, all the partitions are to be considered. Their number $T(F_i)$ grows exponentially according to the size of F_i. 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. This algorithm uses the recursive principle of Stirling numbers of the second kind. The partitions of sets with at most four elements and their ordering number are reported in Table 2. 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.

At step i, fortunately 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. Moreover, for each configuration of F_{i-1} with value strictly lower than $k-1$, we generate also the configuration obtained by adding a new block containing the vertex i.

In order to improve the linear-decomposition, we apply the Dsatur heuristic evenly on the remaining graph H'_i, for different configurations of F_i. If Dsatur

Table 2. Classification of the partitions of sets containing from 1 to 4 elements

	j=1	j=2	j=3	j=4	$T(i)$
i=1	**1** [1]				$T(1) = 1$
i=2	**1** [12]	**2** [1][2]			$T(2) = 2$
i=3	**1** [123]	**2** [13][2]	**5** [1][2][3]		
		3 [1][23]			
		4 [12][3]			$T(3) = 5$
i=4	**1** [1234]	**2** [134][2]	**9** [14][2][3]	**15** [1][2][3][4]	
		3 [13][24]	**10** [1][24][3]		
		4 [14][23]	**11** [1][2][34]		
		5 [1][234]	**12** [13][2][4]		
		6 [124][3]	**13** [1][23][4]		
		7 [12][34]	**14** [12][3][4]		
		8 [123][4]			$T(4) = 15$

finds a k-coloring then the process ends and the result of the k-coloring is yes. Otherwise the linear-decomposition continues until a configuration is generated at step N, in this case the graph is k-colorable, or no configuration can be generated from the precedent step, in this case the graph is not k-colorable. The complexity of the linear-decomposition algorithm, upper bounded by $N * 2^{F_{max}}$, is exponential according to the linearwidth of the graph, but linear according to its number of vertices.

3.3 Example of Configuration Computing

Assume that we are searching for a 3-coloring of the graph G of Fig. 2. Suppose that at step $i-1$ we had $F_{i-1} = \{u, v\}$. The configurations of F_{i-1} were $C(H_{i-1}, 1) = [uv]$ of value α and $C(H_{i-1}, 2) = [u][v]$ of value β. The value of β is 2 or 3, since the corresponding configuration has 2 blocks and $k = 3$.

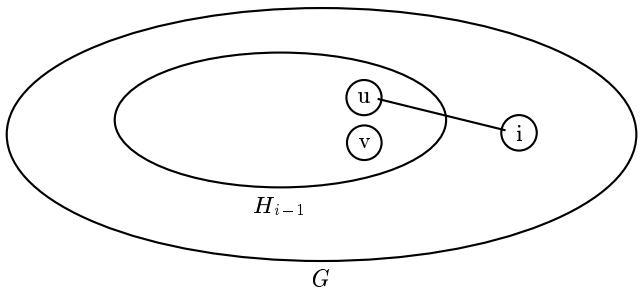

Fig. 2. Construction of $H_i = (V_{i-1} \cup \{i\}, E_{i-1} \cup \{(u, i)\})$

Suppose that at step i, vertex u is deleted from the boundary set (we suppose that it has no neighbor in H'_i), so $F_i = \{v, i\}$. We want to generate the configurations of F_i from the configurations of F_{i-1}. The insertion of i in the unique block of $C(H_{i-1}, 1)$ is impossible, since u and i are neighbors. It is possible to add a new block, it provides the partition [uv][i] of 2 blocks, corresponding to the configuration $C(H_i, 2) = [v][i]$ with $val(C(H_i, 2)) = max(\alpha, 2)$. Vertex i can be introduced in the second block of $C(H_{i-1}, 2)$. It provides the partition [u][vi] corresponding to the configuration $C(H_i, 1) = [vi]$ with value β. It is also possible to add a new block to $C(H_{i-1}, 2)$, it provides the partition [u][v][i] of 3 blocks corresponding to the configuration $C(H_i, 2) = [v][i]$. This configuration already exists, so $val(C(H_i, 2)) = min(val(C(H_i, 2)), max(\beta, 3))$. Thus two 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.

4 k-Colorability Experiments

We performed experiments on the reduced instances of Table 1. Obviously, we did not test instances that were already solved by pre-processings. Results of these experiments are reported in Table 3. For each instance, we tested successive k-colorings, k starting from LB and increasing by step 1 until a coloring exists. We report the result and computing time of our linear-decomposition algorithm $kColor$, for one or two relevant values of k. We give also in column F_{max} the linearwidth of the vertex linear ordering chosen $F_{max}(\mathcal{N})$. Most of these instances are easily solved. Configurations generated by instance 2-FullIns5 for a 6-coloring exceeded the memory capacity of our computer, so we give for this instance the results for a 5-coloring and for a 7-coloring. Instances 2-FullIns4, 3-FullIns4, 4-FullIns4, 5-FullIns4 and 4-Inser3 are solved exactly, whereas no exact method had been able to solve them at the COLOR02 computational symposium (see http://mat.gsia.cmu.edu/COLOR02/ summary.htm for all results).

Table 3: k-colorability results

Problem	F_{max}	k	kColor	Time	k	kColor	Time
1-FullIns3	17	3	no	0.00	4	yes	0.00
1-FullIns4	46	4	no	0.02	5	yes	0.02
1-FullIns5	132	5	no	436.17	6	yes	0.05
2-FullIns3	22	4	no	0.00	5	yes	0.00
2-FullIns4	93	5	no	0.02	6	yes	0.02
2-FullIns5	359	5	no	29.50	7	yes	0.15
3-FullIns3	36	5	no	0.00	6	yes	0.00
3-FullIns4	200	6	no	0.03	7	yes	0.03
4-FullIns3	49	6	no	0.00	7	yes	0.00
4-FullIns4	302	7	no	0.08	8	yes	0.13
5-FullIns3	64	7	no	0.00	8	yes	0.00

continued on next page

continued from previous page							
Problem	F_{max}	k	kColor	Time	k	kColor	Time
5-FullIns4	447	8	no	0.22	9	yes	0.20
3-Inser3	16	3	no	29.27	4	yes	0.00
4-Inser3	20	3	no	1772.95	4	yes	0.00
mug88-1	8	3	no	0.00	4	yes	0.00
mug88-25	8	3	no	0.00	4	yes	0.00
mug100-1	7	3	no	0.02	4	yes	0.00
mug100-25	8	3	no	0.00	4	yes	0.00
miles250	16	7	no	0.00	8	yes	0.00
le450-25a	293	24	no	0.00	25	yes	0.98
le450-25b	303	25	yes	0.00			
fpsol2.i.1	82	65	yes	0.00			
fpsol2.i.2	50	30	yes	0.00			
fpsol2.i.3	50	30	yes	0.00			
inithx.i.1	69	54	yes	0.00			
inithx.i.2	42	31	yes	0.00			
inithx.i.3	42	31	yes	0.00			
mulsol.i.1	61	49	yes	0.00			
mulsol.i.2	45	31	yes	0.00			
mulsol.i.3	46	31	yes	0.00			
mulsol.i.4	45	31	yes	0.00			
mulsol.i.5	47	31	yes	0.00			
school1	291	14	yes	0.00			
school1_nsh	258	14	yes	0.00			
zeroin.i.1	62	49	yes	0.00			
zeroin.i.2	45	30	yes	0.00			
zeroin.i.3	45	30	yes	0.00			
DSJR500_1	68	12	yes	0.02			

5 Conclusions

In this paper, we have presented some pre-processings that are effective to reduce the size of some of difficult coloring instances. We presented also 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 bounded linearwidth. The computational results obtained on literature instances are very satisfactory. We consider using the linear-decomposition mixed with heuristics approach to deal with unbounded linearwidth instances. We are also looking for more reduction techniques to reduce the size or the difficulty of these instances.

References

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

2. D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, april 1979.
3. M. Caramia and P. Dell'Olmo. Vertex coloring by multistage branch-and-bound. In *Computational Symposium on Graph Coloring and its Generalizations*, Corneil University, september 2002.
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. I. Mendez Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. In *Computational Symposium on Graph Coloring and its Generalizations*, Corneil University, september 2002.
8. 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.
9. P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
10. M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
11. F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In Trick and Johnson [23], pages 285–308.
12. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
13. A. Hertz and D. De Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.
14. M. Kubale and B. Jackowski. A generalized implicit enumeration algorithm for graph coloring. *Communications of the ACM*, 28(4):412–418, 1985.
15. C. Lucet. *Méthode de décomposition pour l'évaluation de la fiabilité des réseaux*. PhD thesis, Université de Technologie de Compiègne, 1993.
16. J.F. Manouvrier. *Méthode de décomposition pour résoudre des problèmes combinatoires sur les graphes*. PhD thesis, Université de Technologie de Compiègne, 1998.
17. 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.
18. A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.
19. C. A. Morgenstern. Distributed coloration neighborhood search. In Trick and Johnson [23], pages 335–357.
20. T. J. Sager and S. Lin. A pruning procedure for exact graph coloring. *ORSA Journal on Computing*, 3:226–230, 1991.
21. S. Sen Sarma and S. K. Bandyopadhyay. Some sequential graph colouring algorithms. *International Journal of Electronic*, 67(2):187–199, 1989.
22. E. Sewell. An improved algorithm for exact graph coloring. In Trick and Johnson [23], pages 359–373.
23. Michael A. Trick and David S. Johnson, editors. *Cliques, Coloring, and Satisfiability: Proceedings of the Second DIMACS Implementation Challenge*. American Mathematical Society, 1993.