

TECHNISCHE UNIVERSITEIT EINDHOVEN  
Department of Mathematics and Computer Science

MASTER'S THESIS  
Graph coloring in timetabling  
by  
P.H.M. de Louw

Supervisor: prof. dr. E.H.L. Aarts  
Advisors: dr. ir. H.M.M. ten Eikelder  
dr. ir. R.J. Willemen

Eindhoven, June 2002

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Graph coloring</b>	<b>3</b>
2.1	Basic graph coloring . . . . .	3
2.2	Timetabling . . . . .	4
2.3	Timetabling in terms of graph coloring . . . . .	5
2.3.1	Basic timetabling . . . . .	5
2.3.2	Color class cardinality constraint . . . . .	5
2.3.3	Color availability restriction . . . . .	6
2.3.4	Color nuance class cardinality constraint . . . . .	7
2.4	Algorithm for graph coloring . . . . .	7
<b>3</b>	<b>Proof of correctness of Coudert algorithm</b>	<b>9</b>
3.1	Coudert algorithm . . . . .	9
3.2	Maximum clique search . . . . .	9
3.2.1	Pruning rules . . . . .	14
3.3	Graph coloring using maximum clique . . . . .	16
<b>4</b>	<b>Basic graph coloring</b>	<b>19</b>
4.1	Maximum clique . . . . .	19
4.2	Minimum coloring . . . . .	22
4.3	k-coloring . . . . .	24
<b>5</b>	<b>Color class cardinality constraint</b>	<b>27</b>
<b>6</b>	<b>Restricted color availability</b>	<b>30</b>
<b>7</b>	<b>Color nuance class cardinality constraint</b>	<b>35</b>
<b>8</b>	<b>Future work</b>	<b>45</b>
8.1	Lower bound on the number of colors . . . . .	45
8.1.1	Coloring two cliques . . . . .	45
8.1.2	coloring quasi-cliques . . . . .	46
8.2	Improvements on the coloring . . . . .	46
8.2.1	The order of vertices . . . . .	46
8.2.2	Removing symmetry . . . . .	46
8.2.3	Too few colors available . . . . .	47

8.2.4 Cliques and quasi-cliques . . . . .	47
<b>9 Conclusions</b>	<b>48</b>

### Abstract

Due to the complexity of graph coloring problems, it is generally unreasonable to apply an exact algorithm to solve them. But for some special instances that are derived from practical problems it may be very well possible to find a solution in a reasonable amount of time with some exact algorithm. A promising approach is the following. The first step is to find a maximum clique in the graph. The size of this clique is a valid lower bound on the number of colors that is needed. Then in the second step the graph is colored using this lower bound. This strategy is based on an algorithm that is discussed in [2] by Coudert.

The timetabling problem for Dutch secondary schools can be solved by converting it into a graph coloring problem. The graphs tend to have very specific structures, that allow the use of an exact coloring algorithm effectively.

We show that some different types of graph coloring problems can be solved effectively indeed using an exact algorithm. Some instances with graphs  $G = (V, E)$  of between 50 and 150 vertices are solved quickly using an exact algorithm as long as the density  $\frac{|E|}{|V|}$  is below 40. Some other instances with larger graphs of 654 and 670 vertices respectively with densities of about 40 appeared to be very hard to color. Not even a third part of the graph was colored.

# Chapter 1

## Introduction

A major problem in Dutch secondary schools is the construction of a reasonable timetable. This is usually modelled as a graph coloring problem. The basic idea of timetabling is to assign lessons to time slots in such a way that students don't have more than one lesson in a time slot and that teachers don't have to teach more than one lesson in a time slot. Usually some extra constraints are introduced to increase the quality of the timetable. If each lesson is represented by a vertex in a graph, the vertices of the pairs of lessons that can't be assigned to the same time slot are connected by an edge and each time slot is represented by a color, then the construction of a timetable boils down to a graph coloring problem with a limited amount of available colors. This is known to be NP-complete. Therefore, it is necessary to apply heuristics to find an optimum solution in order to limit the computation time to a reasonable level.

However Coudert found a solution method to solve the problem in an exact way. This is presented in [2]. His algorithm determines a lower bound on the number of colors that is needed to color the graph by finding a maximum clique. After that, the rest of the clique is colored using some sequential coloring algorithm. Although this is not a heuristic we are very interested in it, because in some specific cases this approach shows to be very successful.

The aim of the project is find out whether it is useful to use this algorithm as a base for graph coloring problems and some related problems that come from timetabling problems. These related problems are graph coloring problems where some extra restrictions on the allowed colorings exist. The graphs that are built from the timetabling instances appear to have specific properties that allow Coudert's algorithm to be successful.

In Chapter 2 the graph coloring problem is defined and the algorithm by Coudert is discussed. After this the prove of correctness of the algorithm is given in Chapter 3. Subsequently four different types of graph coloring are considered the results for some instances are shown.

1. The basic graph coloring as described in [2] is considered in Chapter 4.
2. Graph coloring with limited color class cardinality is considered in Chapter 5. This means a color may be used only a limited number of times in the graph coloring.
3. Graph coloring with restrictions on the available color classes is considered in Chapter 6. For each vertex a number of colors may be prohibited. Restrictions as mentioned in the two previous types of graph coloring problems are also taken into account.

4. Graph coloring with color nuance restrictions on subsets of vertices is considered in Chapter 7. The collection of vertices is divided into subsets. Also the collection of color classes is divided into subsets, the color nuance classes. Only a limited number of vertices from such a subset can be put into a color nuance class. Restrictions as mentioned in the three previous types of graph coloring problems are also taken into account.

After considering the various types of graph coloring as mentioned above, conclusions are drawn.

## Chapter 2

# Graph coloring

In this chapter some important terms in graph coloring are introduced first. After that, the different types of graph coloring that are considered in the research are introduced. Finally, the algorithms that are used, are shown in an informal way.

### 2.1 Basic graph coloring

The graph coloring problem can be defined as follows.

**Definition 1 (Graph coloring)** . Given are a graph  $G = (V, E)$  with vertices  $V$  and edges  $E$  and a set of colors  $C$ . A coloring of graph  $G$  is a mapping  $color : V \rightarrow C$ .

With this definition we can define a proper coloring as follows.

**Definition 2 (Proper coloring)** . A proper coloring is a graph coloring in such a way that  $(\forall (v_0, v_1) \in E : color(v_0) \neq color(v_1))$ .

In this project the goal is always be to find a proper coloring. In our case two different types of the graph coloring problem are interesting, the minimum coloring problem and the k-coloring problem.

The algorithm by Coudert, that is discussed in detail in Chapter 3, is based on the minimum coloring problem. The aim of the minimum coloring problem is to find a proper coloring for a graph  $G$  with a number of colors equal to  $\chi(G)$ , the chromatic number, which is defined as follows.

**Definition 3 (Chromatic number)** . Given is a graph  $G = (V, E)$ . Then the chromatic number  $\chi(G)$  is the smallest possible size of set  $C$ , such that a proper coloring for  $G$  exists.

Using this the minimum coloring problem can be defined as follows.

**Definition 4 (Minimum coloring)** . Given is a graph  $G = (V, E)$  and a set of colors  $C$ . Then the minimum coloring problem is to find a proper coloring with  $|C| = \chi(G)$ .

Another type of graph coloring problem is the k-coloring problem. For that problem the aim is to find a coloring containing at most a predefined  $k$  number of colors.

**Definition 5 (k-Coloring)** . Given are a graph  $G = (V, E)$  and a set of colors  $C$  of a predefined size  $|C| = k$ . Then the  $k$ -coloring problem is to find a proper coloring on graph  $G$  with the colors in  $C$ .

We have tried to use an exact algorithm to find a solution for some graph coloring problems. An important property that enables an exact solution is the 1-perfectness of the graphs. In order to define this we first introduce the clique of a graph and the clique number  $\gamma(G)$  of a graph  $G$ .

**Definition 6 (Clique)** . Given is a graph  $G = (V, E)$ . A subset  $G'$  of  $G$  is called a clique if  $(\forall v_0, v_1 \in G' : (v_0, v_1) \in E)$ .

**Definition 7 (Clique number)** . Given is a graph  $G = (V, E)$ . Then the clique number  $\gamma(G)$  is the largest possible size of a subset of  $G$  that forms a clique.

**Definition 8 (Maximum clique problem)** . Given is a graph  $G$ . Then the maximum clique problem is to find a clique of size  $\gamma(G)$ .

Now, 1-perfectness can be defined as follows.

**Definition 9 (1-perfect)** . Given is a graph  $G = (V, E)$  with chromatic number  $\chi(G)$  and clique number  $\gamma(G)$ . Graph  $G$  is called a 1-perfect graph if  $\gamma(G) = \chi(G)$

The minimum coloring problem is known to be NP complete. The same goes for the  $k$ -coloring problem. See [4]. That's why graph coloring problems are usually solved by using advanced heuristics. The maximum clique problem is also known to be NP-complete.

In a clique, any possible two vertices are each others neighbors, so they should get different colors when coloring the graph. A graph  $G$  has a largest clique of size  $\gamma(G)$ . Therefore, at least  $\gamma(G)$  colors are needed to color the graph. This implies  $\gamma(G) \leq \chi(G)$ . Only in the special case of 1-perfectness  $\gamma(G) = \chi(G)$ .

## 2.2 Timetabling

We have used graph coloring techniques to solve a timetabling problem as it occurs in Dutch secondary schools. In our problem, teachers and students are already assigned to lessons. These lessons are to be assigned to time slots in such a way that the following conditions are satisfied.

- Teachers and students have at most one lesson in a time slot.
- The number of available rooms is limited. This constraint is called the room constraint.
- Teachers may not be available for certain times. This constraint is called the teacher availability constraint.
- Courses consist of a number of lessons. On each day these lessons may be scheduled just a limited number of times. This constraint is called the educational constraint.

## 2.3 Timetabling in terms of graph coloring

The timetabling problem should be defined in terms of graph coloring. Therefore, it is defined as a  $k$ -coloring problem first. Later some extra constraints are added in order to satisfy the conditions mentioned above.

### 2.3.1 Basic timetabling

In timetabling lessons that are scheduled on the same hour, can't have the teacher or a student in common. To satisfy this condition we use a basic graph coloring model. The vertices of the graph represent the lessons. When 2 lessons have the same teacher or have students in common, an edge is inserted between the corresponding vertices. Vertices that are connected by an edge have to be assigned different colors, when coloring this graph. These colors represent the hours. Now, vertices that are assigned the same color, are not connected. So, lessons that are scheduled on the same hour can't have the teacher or students in common. Of course in this case it's also important that the necessary number of hours is below the upper bound, defined by the number of available hours. This implies that a  $k$ -coloring should be applied.

Before the actual coloring takes place we try to find a lower bound on the number of colors. As we pointed out in the previous section, we can use the clique number for this. Therefore, a maximum clique is searched.

The clique isn't be used for its size only. The first part of the actual coloring is the coloring of the clique. Each vertex in the clique is assigned a different color. This coloring is always proper for the clique. If it is possible to find a proper coloring for the graph, then it will still be the case after coloring the clique, so there won't be the need to recolor the clique later. Thus, it is very easy to find a feasible coloring for the clique.

Coloring a graph is hard. A coloring for a graph  $G$  containing  $\gamma(G)$  colors may be found quickly. But in the worst case such a coloring doesn't exist and then it may be necessary to try any possible coloring. This takes exponential time. The same goes for the coloring with an upper bound. A coloring containing a number of colors that is at most equal to the upper bound may be found quickly, but in the worst case no such coloring exists and then any possible coloring is tried, which takes exponential time.

But not only the actual coloring of the graph is hard. Finding the largest clique is hard as well. In fact every possible order of the vertices in the graph is tried as an order for building up a clique. As soon as it is sure that no larger clique, than the largest clique so far, can be found by continuing to build up the clique, the search can be pruned. In the worst case this takes exponential time. We apply pruning rules in order to prune unsuccessful branches very early in the build-up process. But still, in the worst case it takes exponential time.

### 2.3.2 Color class cardinality constraint

In the considered timetabling problem the number of available rooms should be taken into account. This was previously referred to as room constraint. In terms of graph coloring, the room constraint is called the color class cardinality constraint.

**Definition 10 (k-Coloring with color class cardinality constraint)** . *Given are a graph  $G = (V, E)$  and a set of colors  $C$ . The color class cardinality constraint defines a maximum on the number of vertices in  $V$  that are mapped on the same color in  $C$ .*

If the maximum allowed size for a color class is one, then the coloring can be done in an efficient way, but for larger values the  $k$ -coloring problem with color class cardinality constraint is NP-complete again. If the cardinality constraint has value one, then each vertex should be mapped onto a different color. If at least  $|V|$  colors are allowed, then finding a proper coloring is straightforward. Otherwise no proper coloring can be found, satisfying the color class cardinality constraint.

The general problem however is NP-complete. If the value of cardinality constraint is high enough, then the computation is done in the same way as in the previously described  $k$ -coloring problem.

### 2.3.3 Color availability restriction

Teachers aren't available for every hour. Therefore, a lesson may be scheduled only on a restricted number of hours. This is previously referred to as teacher constraint. In terms of graph coloring this is called the color availability restriction.

**Definition 11 (k-Coloring with color availability restriction)** . *Given are a graph  $G = (V, E)$ , a set of colors  $C$  and a mapping  $\varphi: V \rightarrow \mathcal{P}(C)$  that defines the colors in  $C$  that are allowed for a given vertex  $v \in V$ . The  $k$ -Coloring problem with color availability restriction is to find a proper coloring for graph  $G$  such that for each  $v \in V$   $color(v) \in \varphi(v)$ .*

In this graph coloring problem the colors actually have some meaning. Before the actual coloring takes place it should be known which color relates to which hour in order to get the forbidden colors for each vertex right. This implies that not every coloring for the clique may be feasible, like in the previous graph coloring problems. Therefore, the clique should be colored in a different way. We have considered to use a matching algorithm to establish a clique coloring.

**Definition 12 (Matching)** . *Given is a bipartite graph  $G = (V, E)$ . The maximum matching problem is to find the largest possible subset  $E'$  of  $E$  such that no vertex appears in more than one element from  $E'$ .*

A matching should be computed between a number of available colors and the vertices in the clique. A matching can be computed in an efficient way and when there is a matching possible, one will be found. But not every matching may be right to color the entire graph. So we may need to change the coloring of the clique by trying to find different matchings.

Figure 2.1 shows two graphs. In the left graph the clique is colored and there is one vertex left to be colored. But in the case that color 2 is forbidden for the vertex, no further coloring is possible unless an extra color is allowed. However the graph on the right shows an other coloring for the clique. Then it is possible to color the entire graph.

Another difference is the selection of the next vertex for coloring. This can be done in the same way, but it's also possible to take the color availability into account in selecting a new vertex for coloring.

Still the coloring is hard. If teachers are always available, then coloring is basically the same as in the two previous problems. Only when teachers are not always available, then parts of the computation tree may be pruned. Then we can change the graph by adding some extra vertices. For each available color, a vertex is added, representing that color. Let's call the set of extra vertices  $G_{color}$ . The elements in  $G_{color}$  constitute a clique. For each pair of vertices



Figure 2.1: If color 2 is prohibited for the uncolored vertex on the left, the clique coloring is wrong.

$\{v_0, v_1\}$ , where  $v_0 \in G$  and  $v_1 \in G_{color}$  an edge is inserted if the color that is represented by  $v_1$  is not in the set  $\varphi(v_0)$ . Now the problem is to find a  $k$ -coloring for this extended graph, which is known to be NP-complete.

### 2.3.4 Color nuance class cardinality constraint

A class has a number of lessons of a course per week. For each of their courses there's a limit on the number of times they can have that course on a day. This constraint is previously referred to as educational constraint. In terms of graph coloring this is called the color nuance class cardinality constraint.

**Definition 13 (k-Coloring with color nuance class cardinality constraint)** . Given are a graph  $G = (V, E)$ , where  $V$  is divided into a partition  $V_0, \dots, V_m$ , a set of colors  $C$ , that is divided into a partition  $C_0, \dots, C_n$ , and for each  $V_i$  a maximum  $max(V_i)$ . Then the  $k$ -Coloring problem with color nuance class cardinality constraint is to find a proper coloring on graph  $G$  where for each  $V_i$  and  $C_j$  at most  $max(V_i)$  vertices are mapped on the elements in  $C_j$ .

In the graph coloring problem with color availability restriction colors are assigned to the vertices in the clique. This can be done efficiently by using a matching algorithm. But for the  $k$ -coloring problem with color nuance class cardinality constraint, this is not possible anymore. The graph should be modified in course of the matching process. Therefore, an other way of coloring the clique should be found or the clique coloring should be skipped as a separate part of the algorithm.

If the color nuance constraint is such that no restrictions are imposed on the coloring, then we have the  $k$ -coloring problem, which is NP-complete. Therefore, the  $k$ -coloring problem with color nuance class constraint is NP-complete as well.

## 2.4 Algorithm for graph coloring

The algorithm for graph coloring is divided into three parts. The first part determines the maximum clique. The second part executes the maximum clique search and subsequently colors the vertices of the maximum clique and executes the third part. In the third part a coloring for the uncolored part of the graph is found. The three parts are as follows.

Maximum clique: {  
Apply pruning:

```

Choose a vertex to add to the clique;
Add this vertex to clique found so far;
Apply Maximum clique search with vertex added to clique;
Apply Maximum clique search leaving vertex out of clique;
Return best known result;
}

```

For this algorithm a clique and a set of vertices are given. The elements of this set have the clique as a subset of their neighborhoods. Therefore, the vertices in this set can all be used to extend the clique. First pruning rules are applied to remove those vertices that will certainly not lead to better results than the best results found so far. After pruning a vertex is chosen and the maximum clique is computed for two cases. In the first case the assumption is made that the chosen vertex will be part of the clique while in the second case the assumption is made that the chosen vertex won't be part of it. The best result of these two possibilities and the previously computed possibilities is finally returned.

```

Sequential coloring: {
  Determine a maximum clique of  $G$ ;
  Color vertices of clique;
  Color resting vertices;
}

```

For the sequential coloring algorithm a graph is given. First a maximum clique of this graph is computed. After that, the vertices of this clique are colored. The last step is to color the remaining vertices of the graph, which is done in the algorithm below.

```

Color resting vertices: {
  Choose an uncolored vertex  $v$ ;
  For each available color  $c$ : {
    If constraints are satisfied: {
      Assign color  $c$  to vertex  $v$ ;
      Color resting vertices;
      Uncolor  $v$ ;
      If a satisfying coloring is found, then return;
    }
  }
}

```

In the last part a number of vertices are already colored. These are the vertices in the previously found clique and a number of vertices among the vertices outside the clique. An uncolored vertex is selected for coloring. After that a set of colors is determined. These colors can be assigned to the vertex. Each of the colors is tried unless a valid coloring satisfying all constraints is found. If such a coloring is found then this coloring is saved and the computation stops. If no such coloring is found, then each of the colors is tried. For each of the partial colorings found in this way, the procedure is repeated.

## Chapter 3

# Proof of correctness of Coudert algorithm

### 3.1 Coudert algorithm

Coudert doesn't use an advanced heuristic to tackle the graph coloring problem. He tries to do it by using an exact coloring algorithm. His algorithm solves the minimum coloring problem. There's the risk that this approach will fail, because the problem is known to be NP-complete. See [4]. But in specific cases graphs may have properties that enable such an algorithm to be successful however. This appears to be the case in some different problems from practise that are easily reduced to a graph coloring problem.

He assumes that these graphs are always 1-perfect. Although he didn't find a proof for this assumption, he didn't find counter examples among his about 600 test graphs either. First the clique number is defined with an exact maximum clique search algorithm using some pruning rules in order to find a maximum clique quickly. However, finding a maximum clique is also known to be an NP-complete problem. But it is supposed to perform well on the considered types of graphs.

If a graph is 1-perfect and the largest clique is found by the clique search algorithm, then the size of the clique is equal to the minimum number of colors. Even the 1-perfectness of a graph  $G$  implies that the search of a minimum coloring boils down to the the search of a coloring containing  $\gamma(G)$  colors. In a 1-perfect graph this seems to be easy. Therefore, the 1-perfectness of a graph may be a good measure for easy colorability.

First the maximum clique search and proof the correctness of the algorithm are discussed. After that some suggested pruning rules are explained and their correctness is proved. Finally, the algorithm for the actual coloring of the graph and the correctness of it are discussed.

### 3.2 Maximum clique search

The first step in the algorithm of Coudert is to find a maximum clique. The size of this clique is used as a lower bound for the number of colors. He uses an exact algorithm to find the maximum clique and he has defined some pruning rules for this algorithm in order to find a clique more quickly. These pruning rules are discussed in Section 3.2.1.

First the annotated version of the clique algorithm is discussed. The first part is the *MaxClique* function. This part is pretty straightforward.

```

function MaxClique( $G$ ) {
  return MaxCliqueRec( $G, \emptyset, \emptyset, +\infty$ );
}

```

The second part is the *MaxCliqueRec* function. The actual computation is done in this part. Some abbreviations are used in the annotation. These are defined first.

- $clique(C) = C$  forms a clique.
- $indset(G) = G$  forms an independent set.
- Pre:  $clique(C) \wedge clique(best) \wedge (\forall c \in C, g \in G : \{c, g\} \in E(C \cup G)) \wedge \chi(C \cup G) \leq ub$   
 $\wedge G = \emptyset \Rightarrow |C| \geq |best|$
- Ret:  $G' : |G'| \geq |best| \wedge |G'| \geq \gamma(C \cup G) \wedge clique(G') \wedge (G' = best \vee G' \subseteq C \cup G)$

With these abbreviations, we have the following annotated *MaxCliqueRec* function.

```

function MaxCliqueRec( $G, C, best, ub$ ) {
  {Pre. Ret. 1,  $ub = UB, best = BEST$ }
  if  $G = \emptyset$  return  $C$ ;
  { $G \neq \emptyset$ }
   $k :=$  the size of a coloring of  $G$ ;
  {
     $\chi(C \cup G) \leq |C| + k,$ 
     $indset(G) \rightarrow k-1 \wedge \chi(C \cup G) = |C| + 1$ 
  }
   $ub := \min(ub, |C| + k)$ ;
  {2.  $ub = UB', UB' = \downarrow(UB, |C| + k), \chi(C \cup G) \leq UB'$ }
  if  $ub \leq |best|$  return  $best$ ;
  { $UB' > |BEST|$ }
   $v :=$  a maximum degree vertex of  $G$ ;
  {}
   $G_1 :=$  graph induced by  $N(v)$ ;
  {3.  $G$  independent set  $\Leftrightarrow G_1 = \emptyset$ }
   $best := MaxCliqueRec(G_1, C \cup \{v\}, best, ub)$ ;
  {
    1.  $best = BEST'$ ,
     $BEST' : BEST' \geq BEST \wedge BEST' \geq \gamma(C \cup G_1) \wedge clique(BEST')$ 
     $\wedge (BEST' = BEST \vee BEST' \subseteq C \cup G)$ 
  }
  if  $ub = |best|$  return  $best$ ;
  { $UB' \neq |BEST'|$ }
   $G_0 :=$  graph induced by  $V(G) - \{v\}$ ;
  {5.  $G_0 = G \setminus \{v\}$ }
  return MaxCliqueRec( $G_0, C, best, ub$ );
}

```

In order to show the correctness of the algorithm, we have the following proof obligations. The actual proofs are added.

1: If  $G = \emptyset$ , then the return condition *Ret* should be satisfied.

$$G = \emptyset \Rightarrow (|C| \geq |best| \wedge |C| \geq \gamma(C \cup G) \wedge clique(C) \wedge (C = best \vee C \subseteq C \cup G))$$

Given is  $best = BEST$ , therefore:

$$G = \emptyset \Rightarrow (|C| \geq |BEST| \wedge |C| \geq \gamma(C \cup G) \wedge clique(C) \wedge (C = BEST \vee C \subseteq C \cup G))$$

$$\begin{aligned} & \text{Assume } G = \emptyset \\ \Rightarrow & \{Pre\} \\ & |C| \geq |BEST| \wedge clique(C) \\ \Rightarrow & \{\gamma(C \cup G) = \gamma(C \cup \emptyset) = \gamma(C) \leq |C|\} \\ & |C| \geq |BEST| \wedge \gamma(C) = |C| \wedge |C| \geq \gamma(C \cup G) \\ \equiv & \{Trivial\} \\ & |C| \geq |BEST| \wedge clique(C) \wedge |C| \geq \gamma(C \cup G) \wedge (C = BEST \vee C \subseteq C \cup G) \end{aligned}$$

□

2: If  $ub \leq |best|$ , then the return condition *Ret* should be satisfied.

$$ub \leq |best| \Rightarrow (|best| \geq |best| \wedge |best| \geq \gamma(C \cup G) \wedge \gamma(best) = |best| \wedge (best = best \vee best \subseteq C \cup G))$$

Given is  $best = BEST$  and  $ub = UB'$ , therefore,

$$UB' \leq |BEST| \Rightarrow (|BEST| \geq |BEST| \wedge |BEST| \geq \gamma(C \cup G) \wedge \gamma(BEST) = |BEST| \wedge (BEST = BEST \vee BEST \subseteq C \cup G))$$

$$\begin{aligned} & \text{Assume } UB' \leq |BEST| \\ \Rightarrow & \{\gamma(C \cup G) \leq \chi(C \cup G), \chi(C \cup G) \leq UB'\} \\ & \gamma(C \cup G) \leq |BEST| \\ \Rightarrow & \{\} \\ & |BEST| \geq |BEST| \wedge |BEST| \geq \gamma(C \cup G) \wedge clique(BEST) \\ & \wedge (BEST = BEST \vee BEST \subseteq C \cup G) \end{aligned}$$

□

3: The precondition of *MaxCliqueRec* should be satisfied at the beginning of the function execution. This is to be established before the function call.

$$clique(C \cup \{v\})$$

$clique(C)$  and  $v$  is connected with each of the vertices in  $C$ , so  $clique(C \cup \{v\})$ .

$$clique(best)$$

$best = BEST$  and from the precondition  $clique(BEST)$  follows.

$$(\forall c \in (C \cup \{v\}), g \in G_1 : \{c, g\} \in E(C \cup \{v\} \cup G_1))$$

From the precondition  $(\forall c \in C, g \in G : \{c, g\} \in E(C \cup G))$  follows.  $G_1 \subseteq G$  and therefore,  $(\forall c \in C, g \in G_1 : \{c, g\} \in E(C \cup G_1))$ . And  $(\forall g \in G_1 : \{v, g\} \in E(\{v\} \cup G_1))$  implies that  $(\forall c \in C \cup \{v\}, g \in G_1 : \{c, g\} \in E(C \cup \{v\} \cup G_1))$

$$\chi(C \cup \{v\} \cup G_1) \leq ub$$

Given  $ub = UB'$  and  $UB' = UB \vee UB' = |C|+k$ . We can distinguish between 2 possibilities.

A.  $UB' = UB$

$$\begin{aligned} & \chi(C \cup \{v\} \cup G_1) \\ \subseteq & \{C \cup \{v\} \cup G_1 \subseteq C \cup G\} \\ & \chi(C \cup G) \\ \subseteq & \{Pre\} \\ & UB \\ \subseteq & \{ \} \\ & UB' \end{aligned}$$

B.  $UB' = |C|+k$

$$\begin{aligned} & \chi(C \cup \{v\} \cup G_1) \\ \subseteq & \{C \cup \{v\} \cup G_1 \subseteq C \cup G\} \\ & \chi(C \cup G) \\ \subseteq & \{Trivial\} \\ & |C|+k \end{aligned}$$

$$G_1 = \emptyset \Rightarrow (|C \cup \{v\}| \geq |best| \wedge C \cup \{v\} \geq \gamma(C \cup \{v\} \cup G_1) \wedge clique(C \cup \{v\}) \wedge (C \cup \{v\} = best \vee C \cup \{v\} \subseteq C \cup G))$$

Given is  $best = BEST$  and  $ub = UB'$ , therefore,

$$G_1 = \emptyset \Rightarrow (|C \cup \{v\}| \geq |BEST| \wedge C \cup \{v\} \geq \gamma(C \cup \{v\} \cup G_1) \wedge clique(C \cup \{v\}) \wedge (C \cup \{v\} = best \vee C \cup \{v\} \subseteq C \cup G))$$

$$\begin{aligned} & G_1 = \emptyset \\ \Rightarrow & \{UB' \leq BEST\} \{UB' \leq |C|+k\} \\ & |BEST| < |C|+k \\ \Rightarrow & \{G \text{ is an independent set, therefore, } k = 1 \text{ and } |C \cup \{v\}| = |C|+1 \} \\ & BEST \leq |C \cup \{v\}| \\ \Rightarrow & \{C \cup \{v\} \subseteq C \cup G\} \\ & |C \cup \{v\}| \geq |BEST| \wedge C \cup \{v\} \leq \gamma(C \cup \{v\} \cup G_1) \wedge clique(C \cup \{v\}) \\ & \wedge (C \cup \{v\} = BEST \vee C \cup \{v\} \subseteq C \cup G) \end{aligned}$$

□

⊣: If  $ub = |best|$ , then the return condition Ret should be satisfied.

$$ub = |best| \Rightarrow (|best| \geq |best| \wedge |best| \geq \gamma(C \cup G) \wedge clique(best) \wedge (best = best \vee best \subseteq C \cup G))$$

Given is  $ub = UB'$  and  $best = BEST'$ , therefore,

$$UB' = |BEST'| \Rightarrow (|BEST'| \geq |BEST'| \wedge |BEST'| \geq \gamma(C \cup G) \wedge clique(BEST') \wedge (BEST' = BEST' \vee BEST' \subseteq C \cup G))$$

$$\begin{aligned}
& \text{Assume } UB' = |BEST'| \\
\Rightarrow & \{ \chi(C \cup G) \leq UB' \} \\
& |BEST'| \geq \chi(C \cup G) \\
\Rightarrow & \{ \gamma(X) \leq \chi(X) \} \\
& |BEST'| \geq \gamma(C \cup G) \\
\Rightarrow & \{ \} \\
& |BEST'| \geq |BEST'| \wedge |BEST'| \geq \gamma(C \cup G) \wedge \text{clique}(BEST') \\
& \wedge (BEST' = BEST' \vee BEST' \subseteq C \cup G)
\end{aligned}$$

□

5: The precondition of the function call to *MaxCliqueRec* should be established.

*clique(C)*

Follows from the precondition.

*clique(best)*

(Given is  $best = BEST'$ . Therefore, this follows from 8.

$(\forall c \in (C), g \in G_0 : \{c, g\} \in E(C \cup G_0))$

From the precondition  $(\forall c \in (C), g \in G : \{c, g\} \in E(C \cup G))$  follows and considering  $G_0 \subseteq G$ .  $(\forall c \in (C), g \in G_0 : \{c, g\} \in E(C \cup G_0))$  is a valid statement.

$\chi(C \cup G_0) \leq ub$

(Given  $ub = UB'$  and  $UB' = UB \vee UB' = |C| + k$ . We can distinguish between 2 possibilities.

A.  $UB' = UB$

$$\begin{aligned}
& \chi(C \cup G_0) \\
& \leq \{ C \cup G_0 \subseteq C \cup G \} \\
& \chi(C \cup G) \\
& \leq \{ Pre \} \\
& UB \\
& \leq \{ \} \\
& UB'
\end{aligned}$$

B.  $UB' = |C| + k$

$$\begin{aligned}
& \chi(C \cup G_0) \\
& \leq \{ C \cup G_0 \subseteq C \cup G \} \\
& \chi(C \cup G) \\
& \leq \{ \text{Trivial} \} \\
& |C| + k
\end{aligned}$$

$G_0 = \emptyset \Rightarrow (|C| \geq |best| \wedge |C| \geq \gamma(C \cup G_0) \wedge \text{clique}(C) \wedge (C = best \vee C \subseteq C \cup G_0))$

Assume  $G_0 = \emptyset$

$\Rightarrow \{ \text{Trivial} \}$

$$\begin{aligned}
& G_0 = \emptyset \wedge G = \{v\} \\
\Rightarrow & \{\text{Trivial}\} \\
& G_0 = \emptyset \wedge G = \{v\} \wedge G_1 = \emptyset \\
\Rightarrow & \{G_1 = \emptyset, \text{Trivial}\} \\
& G_0 = \emptyset \wedge G = \{v\} \wedge G_1 = \emptyset \wedge (BEST' = BEST \vee BEST' \subseteq C \cup \{v\} \cup G_1) \\
& \quad \wedge |BEST'| \geq \gamma(C \cup \{v\} \cup G_1) = |C|+1 \wedge |BEST| \leq |C \cup \{v\}| = |C|+1 \\
\Rightarrow & \left\{ \begin{array}{l} \text{Case analysis} \\ 1. BEST' = BEST \\ 2. BEST' \subseteq C \cup \{v\} \cup G_1 \end{array} \right\} \\
& \text{A. } BEST' = BEST \\
& \quad \Rightarrow \{\text{Substitution in } |BEST'| \geq |C|+1 \wedge |BEST| \leq |C|+1\} \\
& \quad \quad |BEST'| \geq |C|+1 \wedge |BEST'| \leq |C|+1 \\
& \text{B. } BEST' \subseteq C \cup \{v\} \cup G_1 \\
& \quad \Rightarrow \{|C \cup \{v\} \cup G_1| = |C|+1\} \\
& \quad \quad |BEST'| \geq |C|+1 \wedge |BEST'| \leq |C|+1 \\
\Rightarrow & \{\} \\
& \quad |BEST'| = |C|+1 \\
\Rightarrow & \{G \text{ is an independent set, therefore, } k = 1, UB' = |C|+1\} \\
& \quad BEST' = UB' \\
\Rightarrow & \{|BEST'| \neq UB'\} \\
& \quad \text{false} \\
\Rightarrow & \{\} \\
& \quad |C| \geq |best| \wedge |C| \geq \gamma(C \cup G_0) \wedge \text{clique}(C) \wedge (C = best \vee C \subseteq C \cup G_0)
\end{aligned}$$

□

### 3.2.1 Pruning rules

Coudert defines some pruning rules to improve the results. We give these rules with a proof of correctness.

**Rule a.** When  $|C|+|V(G)| \leq |best|$ , the recursion is pruned, because it is impossible to find a larger clique.

Proof: If  $|C|+|V(G)| \leq |best|$ , the recursion can be pruned. Because  $\gamma(C \cup G) \leq |C|+|V(G)|$  is valid, no larger clique can be found by continuing the search. Therefore, in this case  $\gamma(C \cup G) \leq |best|$ . Then no larger clique can be found by continuing the search. This is verifiable in  $\mathcal{O}(1)$ .

□

**Rule b.** Every vertex  $v$  such that  $v.degree < |best| - |C|$  must be put in the clique under construction, since excluding it can't produce a larger clique.

Proof: Each vertex  $v$  for which  $d_v < |best| - |C|$  should be removed from the graph, because it can never be an element of a larger clique than  $best$ . With  $v$  a clique of at most  $|C| + d_v + 1$  vertices can be constructed. So, if  $|C| + d_v + 1 \leq |best|$  that is  $d_v < |best| - |C|$ , then no larger clique including  $v$  than  $|best|$  can be found. For each vertex this is verifiable in  $\mathcal{O}(1)$ .

□

**Rule c.** Proof: Every vertex  $v$  such that  $v.degree \geq |V(G)| - 2$  must be put in the clique under construction, since excluding it can't produce a larger clique.

Each vertex  $v$ , for which  $d_v = |V(G)| - 1$ , should be put in the clique. In this case  $v$  is connected with all other vertices in  $G$  and in  $C$ .  $v$  can be added to every possible clique, not containing  $v$ , and so the clique becomes larger. Furthermore each vertex  $v$  in  $G$  with  $d_v = |V(G)| - 2$  should be added to the clique. Without vertex  $v$  a bigger clique can never be created.

Let  $\gamma(G - v) = k$ . Now, considering such a  $k$ -clique, there are 2 possibilities.

1.  $v$  is connected to every vertex in the  $k$ -clique. By adding  $v$  a new clique of size  $k+1$  is formed. This clique is larger than the largest possible clique not including  $v$ .
2.  $v$  isn't connected to every vertex in the clique. Then there's 1 vertex  $v'$  in the clique, to which  $v$  is not connected. By removing  $v'$  from the clique, a new clique of size  $k-1$  is formed.  $v$  is connected to any of the vertices in the clique. By adding  $v$  to the clique, the size becomes  $k$ . Thus also in this case, without  $v$  no larger clique can be created than by including  $v$  in the clique.

□

**Rule d.** More generally, a vertex  $v$  such that  $V(G) - N(v)$  is an independent set must be put in the clique under construction, since excluding it can't produce a larger clique.

Proof: A vertex  $v$  for which  $V(G) - N(v)$  is an independent set, has to be added to the clique. There can be at most 1 vertex from  $V(G) - N(v)$  in a possible clique in  $V(G)$ . Let the size of the largest clique in  $N(v)$  be  $k$ , then the largest clique in  $V(G)$  has size  $k+1$ . Because  $v$  is connected to all vertices in  $N(v)$ , the largest clique in  $N(v) \cup \{v\}$  has size  $k+1$  as well. Then without  $v$  no larger clique can be built than with  $v$ .

□

**Rule e.** One can force the choice of at least 2 non-neighbors of  $v$  in  $G_0$ . In other words, the maximum clique of  $G$  is either

$$\{v\} \cup \text{MaxClique}(N(v)),$$

or

$$\{v_1, v_2\} \cup \text{MaxClique}(N(v_1) \cap N(v_2)),$$

where  $v_1, v_2 \in V(G) - N(v) - \{v\}$ , and  $v_2 \in N(v_1)$ .

Proof: If  $MaxClique(V(G)) \subseteq \{v\} \cup N(v)$ , then  $v$  is always contained in the clique. Otherwise, if there is a larger clique then  $MaxClique(V(G)) \not\subseteq \{v\} \cup N(v)$  and  $|MaxClique(V(G))| > |MaxClique(\{v\} \cup N(v))|$ . In this case there is a clique without  $v$  larger than the largest possible clique containing  $v$ . The largest possible clique in  $N(v)$  has a size that is 1 smaller than  $MaxClique(\{v\} \cup N(v))$ . Therefore, the clique that is larger than  $MaxClique(\{v\} \cup N(v))$  contains at least two vertices that are not in  $N(v)$ .

□

Besides these pruning rules there's theorem that can be used to reduce the search space as well.

Let  $G$  be the graph at some point of the recursion,  $C$  the clique under construction, and  $best$  the current best solution. Let  $\{l_1, \dots, l_k\}$  be a  $k$ -coloring obtained on  $G$ . Then every vertex  $V$  that can be colored with  $q$  colors, where  $q > |C| - |best| + k$ , can be removed from the graph.

Proof: Given are graph  $G$ , a clique  $C$ , the best clique found so far  $best$ , a coloring of  $G$  with  $k$  colors and a vertex  $v$  that can be colored with  $q$  colors.  $q > |C| - |best| + k$ . The neighbors of  $v$  use at most  $k - q$  colors.

$$\begin{aligned} C' &= C \cup \{v\} \cup MaxClique(N(v)) \\ |C'| &= |C| + 1 + \gamma(N(v)) \\ &\leq |C| + 1 + \chi(N(v)) \\ &\leq |C| + 1 + k - q \\ &\leq |best| \end{aligned}$$

The largest possible clique that can be constructed with  $C \cup \{v\}$  is  $C'$ . This  $C'$  is smaller than  $best$ .

□

### 3.3 Graph coloring using maximum clique

In the last part of this chapter we discuss the actual coloring of the graph. Coudert is using two functions for this. In the  $SC$  function, a maximum clique is found and subsequently, these vertices are colored.

```
function SC(G) {
  C := a clique of G;
  k := 0;
  foreach v ∈ C {
    k := k+1;
    color v with k;
  }
}
```

```

}
{(\forall i : 1 \le i \le k : (\exists v \in C : color(v) = i))}
return SCrec(G, k, |V(G)|+1, |C|);
}

```

The *SCrec* function is the part where the rest of the graph is colored. To prove the correctness of this part, we introduce two definitions of  $\chi'$  and some abbreviations that are used in the annotation first.

**Definition 14** ( $\chi'$ ). *Given: a graph  $G$ , partially colored using colors  $1 \dots k$  colors, and a vertex  $v$ .  $\chi'(G)$  is the minimum number of colors needed to color graph  $G$ , given the existing partial coloring on  $G$ . So for the uncolored graph  $G$ :  $\chi'(G) = \chi(G)$ . If the clique  $C$  is colored with colors  $1 \dots |C|$  and the rest of the graph is uncolored, then also  $\chi'(G) = \chi(G)$ .*

$\chi'(G, v, c) = \chi'(G)$  given that  $color(v) = c$ .

$\chi'(G) = (\downarrow i : 1 \le i \le ((k+1) \downarrow (best-1)) \wedge (\forall v' \in N(v) : color(v') \neq i) : \chi'(G, v, i))$

Based on this definition, the precondition *Pre* and the return value *Ret* are as follows.

Pre:  $\gamma(G) = lb \leq k \leq best$

Ret:  $R : R = BEST \downarrow \chi'(G)$

We introduce invariants  $P_0$  and  $P_1$ .

$P_0$ :  $best = BEST \downarrow (\downarrow i : 1 \le i \le n \wedge M(\forall v' \in N(v) : color(v') \neq i) : \chi'(G, v, i))$

$P_1$ :  $0 \leq n \leq (k+1, best-1)$

Now the annotated version of the *SCrec* function is as follows.

```

/* G is a partially colored graph, using k colors */
/* best is the chromatic number found so far */
/* lb is a lower bound on the number of colors */
function SCrec(G, k, best, lb) {
  {1. Pre. G is entirely colored  $\Rightarrow$  Ret}
  if G is entirely colored return k;
  {G is not entirely colored}
  v := an uncolored vertex of G;
  {}
  for (c := 1; c  $\leq$  min(k+1, best-1); c := c+1) {
    {P0, P1}
    if ( $\forall v' \in N(v) :: color(v') \neq c$ ) {
      {}
      color v with c;
      {2}
      best := SCrec(G, max(c, k), best, lb);
      {}
      uncolor v;

```

```

    {3}
    if  $lb=best$  return  $best$ ;
    { $lb < best$ }
  }
}
{4}
return  $best$ ;
}

```

To show the correctness of the *SCrec* algorithm, we have to prove the validity of some assertions. We give these assertions with their proofs.

1. At the start of the algorithm, the return condition *Ret* should be satisfied in the case that  $G$  is already entirely colored. Then a coloring of size  $k$  is found. So we have to prove that  $G$  is entirely colored  $\Rightarrow k = BEST \downarrow \chi'(G)$

Assume that  $G$  is entirely colored. Then  $\gamma'(G)$  equals the number of colors that is used,  $k$ , so  $k = \gamma'(G)$ . Also  $k \leq best$  and  $best = BES$ , and therefore,  $k = BEST \downarrow \gamma'(G)$ .

□

2. Before a function call to *SCrec*, the precondition for the function should be established.

Graph  $G$  is a partially colored graph using  $k \uparrow c$  colors. According to invariant  $P_0$  and the precondition  $best = BEST \downarrow (\downarrow i : 1 \leq i \leq n \wedge M(\forall v' \in N(v) : color(v') \neq i) : \chi'(G, v, i))$ ,  $best$  is the size of the best coloring found so far.  $k \uparrow c \leq best$  is equivalent to  $k \leq best \wedge c \leq best$ .  $k \leq best$  follows from the precondition and  $c \leq best$  follows from  $c \leq best - 1$ . A coloring for  $G$  using  $best$  colors exists. Therefore,  $best \geq \chi(G)$  and because  $lb = \gamma(G)$  and  $\gamma(G) \leq \chi(G)$   $lb \leq best$ .

□

3. If  $lb = best$  then the return condition *Ret* should be satisfied. Therefore, we prove that  $lb = best \Rightarrow best = \downarrow (BEST, \chi'(G))$

Assume  $lb = best$ . As  $best$  is the size of an already found coloring and  $lb = \gamma(G) \leq \chi(G) \leq best$ , no coloring for  $G$  can be found, that contains less than  $best$  colors. Therefore,  $best = \downarrow (BEST, \chi'(G))$ .

□

4. The return condition *Ret* should be satisfied upon return at the end of the function execution.

According to invariant  $P_0$  and the condition of the repetition  $best = BEST \downarrow (\downarrow i : 1 \leq i \leq n \wedge M(\forall v' \in N(v) : color(v') \neq i) : \chi'(G, v, i)) = BEST \downarrow \chi'(G)$  holds after execution of the repetition.

□

## Chapter 4

# Basic graph coloring

In the Chapter 3 the algorithm by Coudert is introduced. In this chapter the implementation of the algorithm is considered in order to verify whether the results that Coudert is claiming are correct. This subject contains two important issues, i.e. the maximum clique algorithm and the actual coloring of the graph. Subsequently the necessary modifications to use the algorithm for the timetabling problem are discussed.

### 4.1 Maximum clique

The first thing to do to find a coloring for the graph is to find a feasible lower bound for the number of colors. This is done by the maximum clique algorithm. We implemented the clique algorithm without the pruning rules. The results for this approach are presented. After this we added some of the suggested pruning rules. These results are also presented.

After the implementation of the clique search algorithm the question remains what pruning rules we should implement. This is discussed first.

**Rule a.** This rule is trivial to implement and it takes constant time to verify the condition  $|C| + |V(G)| \leq |best|$ . We chose to implement this rule.

**Rule b.** This rule is also trivial to implement. It takes constant time for verifying the condition and if this rule is successful, then the vertex should be removed from the graph which takes  $\mathcal{O}(|V(G)|)$  time. But in Chapter 3 it is mentioned that if this rule would be successful on a vertex  $v$ , then the condition to remove vertex  $v$  by using the  $q$ -color pruning theorem is satisfied as well. So, because the  $q$ -color pruning theorem is a more powerful means to find vertices that can be removed we chose not to implement this rule.

**Rule c.** This rule again is trivial to implement. The condition  $v.degree \geq |V(G)| - 2$  is verifiable in constant time and adding a vertex to the clique under construction, while removing it from the resting graph  $G$ , takes  $\mathcal{O}(|V(G)|)$  time. It may be wise to use case analysis in the implementation. Given a vertex  $v$ , there are two interesting cases possible,  $v.degree = |V(G)| - 1$  and  $v.degree = |V(G)| - 2$ . Then vertices  $v$  satisfying the former condition are added to the clique under construction first and when no more vertices satisfying this condition are available, the vertices satisfying the latter condition are added until there are no more vertices left satisfying this condition. To construct a

clique, usually a vertex is chosen and subsequently a decision should be made whether to add this vertex to the clique or to leave it out. When this rule is successful for a vertex, the possibility to leave this vertex out isn't tried, because it can't give a larger clique. So it saves up to almost 50 % of the computation time in the branch on each successful application and therefore, we chose to implement this rule.

**Rule d.** This rule is quite easy to implement. Applying the rule to some vertex however introduces an overhead that is too large. Verification whether  $V(G) - N(v)$  is an independent set takes  $\mathcal{O}(|V(G)|^2)$  time. Considering that this is always needed when applying the rule on some vertex, it's better to leave this rule out.

**Rule e.** This rule is more delicate to implement. Finding two non-neighbors is easily done in  $\mathcal{O}(|V(G)|)$  time. But still we chose not to implement this rule, because we think that applying this rule won't cut the search tree considerably when applying rules a, c and the q-color pruning theorem, especially when applying the algorithm on the type of graphs we use. Also the application of the coloring heuristic to determine an upper bound for the possible clique size mostly indicates that no better clique can be found by checking alternative ways.

**q-color pruning rule.** This rule may be trivial to implement. In the clique algorithm a coloring heuristic is used to generate some upper bound for the size of the largest clique. The same algorithm can be applied to generate a color that can be used for this pruning rule. We implemented a heuristic that would take  $\mathcal{O}(|V(G)|^2)$  time. After this the verification of the condition  $q > |C| - |best| + k$  takes  $\mathcal{O}(|V(G)|)$  per vertex while removing the vertex in case of success also takes  $\mathcal{O}(|V(G)|)$ . By applying this rule some vertices are eliminated in an early stage in the construction of the clique. If the application of this q-color pruning rule is successful, then subsequently the application of rule a is more likely to be successful. We applied this rule, which makes the application of rule b questionable.

So we implemented rules a, c and the q color theorem rule. These are inserted at the beginning of the *MaxCliqueRec* function. The execution is repeated until none of them is successful anymore.

Table 4.1, Table 4.2 and Table 4.3 show computation times for the maximum clique search while pruning rules are added. Note that in some cases the computation time is 0.00s. This means that the computation time is smaller than the precision interval of the timer. The graphs mentioned in Table 4.1 are found on the Internet and are frequently used for benchmark purposes. The graphs in Table 4.2 and Table 4.3 are taken from intermediate results of a graph coloring heuristic for timetabling purposes. Some computations didn't give results within 24 hours. These computations are aborted and the corresponding places in the tables are left empty.

The times  $t_1, \dots, t_4$  in Table 4.1, Table 4.2 and Table 4.3 are the computation times for the following maximum clique search algorithms.

$t_1$ : Maximum clique search without using pruning rules.

$t_2$ : Maximum clique search using pruning rule a.

$t_3$ : Maximum clique search using pruning rules a and c.

Table 4.1: Computation times for maximum clique search on benchmark graphs.  $t_1$ : without pruning rules.  $t_2$ : only rule a.  $t_3$ : with rule a and c.  $t_4$ : with rule a, c and the q-color theorem pruning rule. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	$ V $	$ E $	clique size	$t_1$	$t_2$	$t_3$	$t_4$
empty	0	0	0	0.00	0.00	0.00	0.00
singleton	1	0	1	0.00	0.00	0.00	0.00
K4	4	6	4	0.00	0.00	0.00	0.00
school1_nsh	352	14612	14	32.4	32.2	7.04	6.83
school1	385	19095	14	375	235	86.3	81.3
fpsol2.i.1	496	11654	65	0.62	0.17	0.16	0.15
fpsol2.i.2	451	8691	30	0.31	0.09	0.07	0.07
myciel3	11	20	2	0.00	0.00	0.00	0.00
myciel4	23	71	2	0.00	0.00	0.00	0.00
myciel5	47	236	2	0.00	0.00	0.00	0.00
le450_25a	450	8260	25	0.03	0.05	0.03	0.03
le450_25b	450	8263	25	0.01	0.02	0.02	0.02
le450_5c	450	9803	5	2.30	2.75	0.89	0.88
queen5_5	25	160	5	0.01	0.00	0.00	0.00
queen6_6	36	290	6	0.02	0.00	0.00	0.00
queen7_7	49	476	7	0.02	0.01	0.00	0.00
flat1000_50_0	1000	245000		> 24h	> 24h	> 24h	> 24h
flat300_20_0	300	21375	11	78.8	78.8	79.2	65.0

Table 4.2: Computation times for maximum clique search on timetabling graphs.  $t_1$ : without pruning rules.  $t_2$ : only rule a.  $t_3$ : with rule a and c.  $t_4$ : with rule a, c and the q-color theorem pruning rule. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	$ V $	$ E $	clique size	$t_1$	$t_2$	$t_3$	$t_4$
CDath4TS00	670	6937	40	0.13	0.05	0.03	0.03
CDhavo4TS00	670	7362	40	0.05	0.03	0.03	0.03
CDhavo4TS01	670	7778	40	0.05	0.02	0.02	0.02
CDhavo4TS02	670	7522	44	0.05	0.02	0.02	0.02
CDhavo4TS03	670	8498	42	0.06	0.03	0.03	0.03
CDhavo4TS00	670	9070	40	66.6	0.13	0.09	0.08
CDhavo4TS01	670	8803	40	0.22	0.08	0.05	0.05
CDhavo4TS02	670	8758	40	0.27	0.09	0.05	0.05
CDhavo4TS03	670	8853	41	0.30	0.07	0.05	0.05
CDhavo4TS04	670	9505	40	1.73	0.17	0.13	0.07
CDhavo5TS00	670	8342	42	0.17	0.06	0.04	0.04
CDhavo5TS01	670	8070	38	0.15	0.06	0.04	0.04
CDhavo5TS02	670	8377	42	0.16	0.05	0.04	0.04

Table 4.3: Computation times for maximum clique search on timetabling graphs that are used for experiments with different graph coloring and graph coloring related problems.  $t_1$ : without pruning rules.  $t_2$ : only rule a.  $t_3$ : with rule a and c.  $t_4$ : with rule a, c and the q-color theorem pruning rule. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	$ V $	$ E $	clique size	$t_1$	$t_2$	$t_3$	$t_4$
JEhavo4.0	136	2179	35	0.04	0.04	0.01	0.01
JEhavo4.1	136	3083	39	0.06	0.06	0.01	0.02
JEhavo5.0	90	670	28	0.01	0.01	0.00	0.00
JEhavo4.1	90	3440	39	0.53	0.49	0.12	0.03
JEvwo4.0	67	617	22	0.01	0.00	0.00	0.00
JEvwo4.1	67	1689	36	0.03	0.03	0.01	0.01
JEvwo5.0	78	1462	40	0.03	0.02	0.01	0.01
JEvwo5.1	78	1759	40	0.03	0.03	0.01	0.01
JEvwo6.0	66	693	28	0.01	0.01	0.00	0.00
JEvwo6.1	66	1961	40	0.01	0.04	0.00	0.00
CDvwo5.0	100	1068	28	0.03	0.02	0.01	0.01
CDvwo5.1	100	1452	36	0.04	0.04	0.01	0.01
CDvwo5_initial	100	433	17	0.00	0.00	0.00	0.01

$t_4$ : Maximum clique search using pruning rules a and c and the q-color pruning theorem.

As we can see from the results, some computation times decrease significantly as pruning rules are added. On the other hand there are also some cases that meet increases of computation time, although these are less significant and occur less frequently. Besides that there are cases where computation times hardly change. From Table 4.1, Table 4.2 and Table 4.3 we can see that a higher density roughly implies longer computation times. In Table 4.1 we can see that the longest computation times are in large graphs with relatively small clique sizes. A large clique size may suggest locally high densities. The maximum clique search algorithm is such that it probably finds these large cliques quickly, while other smaller cliques are rejected soon due to pruning. Many possible cliques of about the maximum clique size may send the maximum clique search algorithm into the wrong direction. Compare for examples the results of the graphs 'school1\_nsh', 'school1', 'fpsol2.i.1' and 'fpsol2.i.2', or the graphs 'le450.25a', 'le450.25b' and 'le450.5c'.

Even after adding the pruning rules some computations on the benchmark graphs still take quite a lot of time, like seconds, minutes or days. But the maximum cliques in the timetabling graphs are all found within 0.20 s.

## 4.2 Minimum coloring

With the maximum clique algorithm working, we started to implement the minimum coloring algorithm. It appeared that Couderc wasn't clear about how to select the next vertex for coloring. Even worse this choice appeared to be essential for the performance of the algorithm. So we have considered some different approaches and we have implemented some of them in order to compare their performances.

Table 4.1: Computation times on graph coloring on benchmark graphs for highest saturation degree first ( $t_\chi$ ).  $t_\gamma$  is the time to compute the clique size. If nothing is filled in, then we were not able to find the corresponding results. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	$ V $	$ E $	clique size	colors	$t_\gamma$	$t_\chi$
empty	0	0	0	0	0.00	0.00
singleton	1	0	1	1	0.00	0.00
K4	4	6	4	4	0.00	0.00
school1_nsh	352	14612	14	14	6.55	0.02
school1	385	19095	14	14	81.3	0.02
fpsol2.i.1	496	11654	65	65	0.14	0.02
fpsol2.i.2	451	8691	30	30	0.08	0.02
myciel3	11	20	2	4	0.00	0.00
myciel4	23	71	2	5	0.00	0.00
myciel5	47	236	2	6	0.00	0.00
le450_25a	450	8260	25	25	0.03	0.02
le450_25b	450	8263	25	25	0.02	0.02
le450_5c	450	9803	5	5	0.85	0.03
queen5_5	25	160	5	5	0.02	0.01
queen6_6	36	290	6	7	0.01	0.00
queen7_7	49	476	7	7	0.00	0.00
flat1000_50_0	1000	245000			> 24h	
flat300_20_0	300	21375	11	20	34.2	> 24h

- Lowest ID first. Every vertex has its unique ID, just to distinguish between different vertices. In fact it doesn't matter what vertex has what ID, so depending on the way of initializing the graph, this way of coloring may perform well or not. But this is the most straightforward approach and this approach is very easy to implement.
- Selection at random. This is also easy to implement, but it doesn't depend heavily on the way the graph is initialized.
- Highest degree first. This is based on the assumption that the larger the neighborhood, the more difficult it is to assign a color to a vertex. In this way the vertices that are the hardest to color are colored first.
- Selection on saturation degree. The saturation degree is the number of colors that are available for a vertex. This means that the vertex, that has the smallest number of colors available, is colored first. This is based on the assumption that the higher the saturation degree, the more difficult it is to assign a proper color to the vertex. In this way again the vertices that are the hardest to color are colored first.

If computations are not successful within 24 hours, they are aborted. As we can see from the results in Table 4.4 and Table 4.5, the computation times tend to be very low for graph coloring. And if they can't be colored within some hundreds of seconds, they still aren't colored after 24 hours.

Table 4.5: Computation times on graph coloring on benchmark graphs for highest saturation degree first ( $t_\chi$ ).  $t_\gamma$  is the time to compute the clique size. CPU: Pentium III 533 MHz, RAM: 256 MB. OS: Linux.

graph	$ V $	$ E $	clique size	colors	$t_\gamma$	$t_\chi$
CDath4TS00	670	6937	40	40	0.02	0.04
CDdmavo4TS00	670	7362	40	40	0.02	0.04
CDdmavo4TS01	670	7778	40	40	0.02	0.04
CDdmavo4TS02	670	7522	44	44	0.02	0.04
CDdmavo4TS03	670	8498	42	42	0.03	0.04
CDhavo4TS00	670	9070	40		0.07	> 24h
CDhavo4TS01	670	8803	40	40	0.05	0.03
CDhavo4TS02	670	8758	40	40	0.05	0.04
CDhavo4TS03	670	8853	41	41	0.04	0.04
CDhavo4TS04	670	9505	40		0.06	> 24h
CDhavo5TS00	670	8342	42	42	0.04	0.04
CDhavo5TS01	670	8070	38	38	0.04	0.03
CDhavo5TS02	670	8377	42	42	0.04	0.04

We also tried to color the graphs with some other selection methods for the vertices. But the results were worse. The results for selection on ID and selection at random were the worst. The results for highest degree first were better than that. This showed some slightly shorter computation times, but it showed computation times of more than 24 hours more often and in all cases were the highest saturation degree first approach didn't give a solution within 24 hours, the highest degree first approach didn't do it either.

The maximum clique problem is known to be NP-complete, but as it shows, the algorithm is very successful on our instances. In this way it serves as a good base for the graph coloring. The coloring part is not as successful as the clique part. It was not possible to color two of the graphs among the timetabling graphs. Note that these two graphs have the highest densities. For those graphs that were colored, we know that they are either not 1-perfect or there is no evidence that they are 1-perfect. Furthermore we should take into account that Coudert is not really clear about the way a new vertex should be chosen for coloring. So, he may have used other conditions for the selection of a vertex than those we used.

### 4.3 k-coloring

Finally we want to apply the graph coloring algorithm in timetabling problems. For that purpose the minimum coloring approach is not always appropriate, so we used a k-coloring instead. This provides us with the means to use the number of available hours for teaching as an upper bound for the number of colors. The k-coloring are based on the techniques that Coudert introduces in his algorithm for minimum graph coloring. So, some modifications are necessary.

When the clique size is larger than the number of allowed colors  $k$ , it won't be possible to find a coloring containing at most  $k$  colors. In this case the computation can be stopped with

Table 4.6: Computation times for basic k-coloring on a set of timetabling graphs that are used for our experiments on graph coloring problems and graph coloring related problems.  $t_\gamma$  is the computation time for the maximum clique search and  $t_\chi$  is the computation time for the actual coloring. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	$ V $	$ E $	clique size	colors	$t_\gamma$	$t_\chi$
JEhavo4.0	136	2179	35	35	0.01	0.01
JEhavo4.1	136	3083	39	39	0.02	0.00
JEhavo5.0	90	670	28	28	0.00	0.00
JEhavo5.1	90	3440	39	41	0.03	42.6
JEvwo4.0	67	617	22	22	0.00	0.00
JEvwo4.1	67	1689	36	36	0.01	0.00
JEvwo5.0	78	1462	40	40	0.01	0.00
JEvwo5.1	78	1759	40	40	0.01	0.00
JEvwo6.0	66	693	28	28	0.00	0.01
JEvwo6.1	66	1961	40	40	0.00	0.00
CDvwo5.0	100	1068	28	28	0.01	0.00
CDvwo5.1	100	1452	36	36	0.01	0.00
CDvwo5.initial	100	433	17	17	0.01	0.00

a negative result. Otherwise the actual coloring of the graph can begin.

The condition to stop the coloring, when a feasible coloring is found, changes as well. It was  $ub = best$  and it changes into  $ub \geq best$ .

In the graphs that we consider, the upper bound always is 40.

Table 4.6 shows the results of tests using the k-coloring on some graphs. These graphs are intermediate results in a timetabling heuristic. Those graphs ending in '.0' are known to be colorable with 40 colors and taking all constraints, that are added later, into account. There exist colorings for these graphs that were generated by the coloring heuristic. For the graphs ending in '.1' there is no evidence about the question whether they are colorable using 40 colors at most and taking all constraints into account. The coloring heuristic didn't find a coloring for these graphs. The graph ending in '.initial' is a graph where the coloring started in the coloring heuristic. In that run it was the first graph that was found to be colorable.

The graph JEhavo5.1 shows a clique size that is unequal to the number of colors used. The coloring containing 41 different colors is the best one we could find within 24 hours. The algorithm could give a coloring containing 42 colors in less than a second and for a coloring containing 41 colors, it took 42.6 seconds.

Here again, we can see that the clique coloring is very successful. Only one graph can't be colored using our graph coloring algorithm within 24 hours. The algorithm itself is an exact coloring algorithm. After 24 hours the algorithm was aborted. This may suggest that this graph is not 1-perfect, but we couldn't prove this. It may also be tricky to find a coloring with 40 colors. Therefore, we considered again some measure for the density of the graph,  $\frac{|E|}{|V|}$ . Table 4.7 gives the results. Indeed the graph that appears hard to color, obviously has the highest density of all of them. The clique sizes are about the same. With a high density, large cliques are more likely to be present. This may make it more difficult to color the graph.

Table 4.7: Graph densities.

Graph	$ V $	$ E $	$\frac{ E }{ V }$
JEhavo4.0	136	2179	16.0
JEhavo4.1	136	3083	22.7
JEhavo5.0	90	670	7.4
JEhavo5.1	90	3440	38.2
JEvwo4.0	67	617	9.2
JEvwo4.1	67	1689	25.2
JEvwo5.0	78	1462	18.7
JEvwo5.1	78	1759	22.6
JEvwo6.0	66	693	10.5
JEvwo6.1	66	1961	29.7
CDvwo5.0	100	1068	10.7
CDvwo5.1	100	1452	14.5
CDvwo5_initial	100	433	4.3

## Chapter 5

# Color class cardinality constraint

After implementing the basic k-coloring, we have added the color class cardinality constraint. This is discussed in this chapter.

We already gave the definition of k-coloring in Chapter 2, but we recall it here.

**Definition 15 (k-Coloring with color class cardinality constraint)** . *Given are a graph  $G = (V, E)$  and a set of colors  $C$ . The color class cardinality constraint defines a maximum on the number of vertices in  $V$  that are mapped on the same color in  $C$ .*

The purpose of the color class cardinality constraint is to limit the sizes of each of the color classes. The value of the color class constraint is defined by introducing an extra variable. The selection of the next vertex for coloring is different. Some colors may not be allowed anymore because of the color class constraint. This means that the number of colors that is still available decreases, which makes a vertex harder to color.

In the k-coloring algorithm in Chapter 4 a color is selected first. After that a verification step is executed to determine whether the color is admitted considering the neighborhood. Here an extra check should be added to determine whether the color class constraint is satisfied as well. In order to do this we need the value of the color class constraint and the cardinality of the actual color class.

The most efficient way to keep track of the color class cardinality is to introduce an integer array where each element represents a color class. Every time when a color is assigned to a vertex, the respective element of the array is increased by one and when the vertex is uncolored again, the element is decreased by one. In this way, the cardinality can be checked and updated in constant time.

Table 5.1 shows the results of the computations. For each of the graphs the size of the maximum clique was considered and based on this we chose some values for the upper bound and the maximum number of color classes that would make the coloring as hard as possible. For the maximum number of color classes we took the clique size. Then we computed a number  $X$ , which is the smallest natural number, larger than  $\frac{|V|}{\gamma(G)}$ . This number  $X$  was taken as a value for the color class cardinality constraint. After that some smaller values were used as a value for  $X$ , the color class constraint, while the upper bound on the number of colors was computed to be the smallest natural number, larger than  $\frac{|V|}{X}$ . The color class constraint didn't seem to make the coloring harder. In almost all cases where there was a chance to find a coloring, based on the upper bound, the maximum clique size and the number

Table 5.1: Computation times for different graphs with an upper bound on the size of the color classes.  $t$  is the computation time for the actual coloring of the graph. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	upper bound	colorst	
JEhavo4_0	4	35	0.00
JEhavo4_1	4	39	0.00
JEhavo5_0	2	45	0.00
JEhavo5_0	3	30	0.00
JEhavo5_0	4	28	0.00
JEhavo5_1	2	51	0.00
JEhavo5_1	2	50	1.40
JEhavo5_1	2	49	22.3
JEhavo5_1	3	42	0.00
JEhavo5_1	3	41	42.4
JEhavo5_1	4	42	0.00
JEhavo5_1	4	41	42.4
JEvwo4_0	2	35	0.00
JEvwo4_0	3	23	0.00
JEvwo4_0	4	22	0.00
JEvwo4_1	2	36	0.01
JEvwo5_0	2	40	0.00
JEvwo5_1	2	40	0.00
JEvwo6_0	2	33	0.00
JEvwo6_0	3	28	0.00
JEvwo6_1	2	40	0.00
CDvwo5_0	3	34	0.00
CDvwo5_0	4	28	0.00
CDvwo5_1	3	36	0.00
CDvwo5_initial	3	34	0.00
CDvwo5_initial	4	25	0.00
CDvwo5_initial	5	21	0.00
CDvwo5_initial	5	20	0.51

of vertices in the graph, we actually found one. Only JEhavo5\_1 was difficult, but that's in line of expectation, because that was also the case for the basic graph coloring.

## Chapter 6

# Restricted color availability

In this chapter we are discussing adding restricted color availability to the  $k$ -coloring with color class constraint. This has already been defined in Chapter 2, but we recall this definition shortly.

**Definition 16 (k-Coloring with color availability restriction)** . Given are a graph  $G = (V, E)$ , a set of colors  $C$  and a mapping  $\varphi : V \rightarrow \mathcal{P}(C)$  that defines the colors in  $C$  that are allowed for a given vertex  $v \in V$ . The  $k$ -Coloring problem with color availability restriction is to find a proper coloring for graph  $G$  such that for each  $v \in V$   $color(v) \in \varphi(v)$ .

In order to use restricted color availability a data structure is extended to save the color availability for each of the vertices. For each vertex an array is defined containing the color availability for the vertex.

Now, the colors should be assigned to the vertices of the graph. Here a problem arises, because the clique can't be colored in the same way as it was done before. This is due to the color availability, which may prohibit colorings of the clique. For each vertex a number of colors is available. The vertices are to be matched with these colors. This can be done using an efficient matching algorithm.

But an other problem arises. Using the matching to color the clique may lead to a partially colored graph where it is impossible to find a proper coloring for the rest of the graph, while in fact it is possible to color the graph. See Figure 6.1 for an example of this. So, an other coloring for the clique should be found. This is done by finding different matchings. Our matching algorithm uses a start vertex as a parameter. By taking a different vertex as a start vertex for the matching, a different matching may be computed. Each vertex in the clique

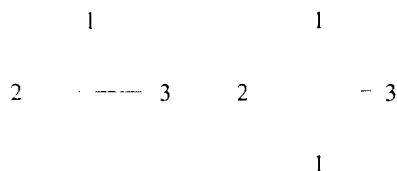


Figure 6.1: If color 2 is prohibited for the uncolored vertex on the left, the clique coloring is wrong.

can be used once. It is possible to try all of them until a matching is found that allows the coloring algorithm to color the rest of the graph quickly.

Additionally the coloring of the rest of the graph should be limited. It is possible to limit the number of backtracks, but we chose an other method. The vertices that are supposed to be the hardest to color are colored first. After that it should be relatively easy to color the other vertices. If we choose to limit the number of backtracks, then it is unlikely that other colorings than the first coloring are tried for the hardest vertices. In fact it may be useful to come back there. Besides that a coloring may be missed very closely on the first attempt, so some backtracking should be allowed in the end. Mostly if a coloring is not found quickly it is likely to take a very long time, so after some backtracking in the end it may be more useful to restart the coloring in a different way. Based on this we decided to allow two colors to be tried for each of the first ten and for each of the last ten vertices in the coloring order. Of course these are vertices outside the clique. In this way at most  $2^{10}$  paths are tried in the first part. For each of these paths at most  $2^{10}$  paths are tried in the last part. That gives at most  $2^{20}$  different paths to look for a coloring, that is satisfying the constraints. If one of these paths is successful, a coloring is found. Otherwise the coloring algorithm returns a negative result.

The selection of the next vertex for coloring is different as well. In fact we can still use the same selection criteria as we used before, but the color availability constraint implies that fewer colors may be available for a vertex. So we changed the selection. First we counted the colors that were not used by neighbors, but now we have to omit the colors that are prohibited.

As we mentioned in Chapter 5, after choosing a color for a vertex a verification step is executed to check whether the color is allowed, considering the colors in the neighborhood and the color class cardinality constraint. Now, in the verification step the color availability should be checked as well.

In the previous coloring problem, colorings containing more than 40 color classes were allowed. But now the color availability data are based on a maximum of 40 color classes. Therefore, no more than 40 color classes are allowed.

Table 6.1 and Table 6.2 show results of computations of a  $k$ -coloring with color class cardinality constraint and color availability constraint. The instances are same as in Chapter 5. For each of the graphs there's a color availability matrix available. The upper bound in the table is the value of the color class cardinality constraint. 'Colors' means the maximum number of colors that was allowed for the coloring. In the algorithm, different matchings are computed. Then a '-' in the column match, means that none of these matchings actually led to a coloring. If there's a number in the column, then this number of matchings was computed to find the first successful matching. The time gives the computation time to color the graph for the run with the successful matching. So unsuccessful matchings are not included in that time.

From the results in Table 6.1 and Table 6.2, we can see that no colorings are found for JEHavo5.1 and JEvwo6.1 containing a number of colors that is equal to the clique size. For JEvwo6.1, 40 color classes are needed at least because of the size of the maximum clique. Therefore, the number of allowed color classes couldn't be increased. But for JEHavo5.1 it was possible to allow an extra color class and this was leading to a successful coloring. This is quite remarkable, because we couldn't find a coloring for this graph containing 40 color classes without the color availability constraint.

We can also see that in most cases the first matching didn't give a successful coloring. In general it is more difficult to find a coloring. But there is one exception among our graphs,

Table 6.1: Computation results for graph coloring with color class cardinality constraint and color availability constraint. Runs with unsuccessful matchings are not included in the computation times in the  $t$ -column. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	upper bound	colors	match	$t$
JEhavo4_0	4	35	19	0.00
JEhavo4_0	4	36	18	0.01
JEhavo4_0	4	37	17	0.01
JEhavo4_0	4	38	16	0.00
JEhavo4_0	4	39	12	0.00
JEhavo4_0	4	40	11	0.00
JEhavo4_0	5	40	11	0.00
JEhavo4_1	4	39	18	0.03
JEhavo4_1	4	40	17	0.02
JEhavo4_1	5	39	18	0.03
JEhavo4_1	5	40	17	0.02
JEhavo5_0	3	35	1	0.00
JEhavo5_0	4	28	1	0.00
JEhavo5_1	3	39	-	-
JEhavo5_1	3	40	10	0.01
JEhavo5_1	4	39	-	-
JEhavo5_1	4	40	10	0.01

Table 6.2: Computation results for graph coloring with color class cardinality constraint and color availability constraint. Runs with unsuccessful matchings are not included in the computation times in the  $t$ -column. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	upper bound	colors	match	$t$
JEvwo4_0	2	35	1	0.00
JEvwo4_0	3	23	1	0.00
JEvwo4_0	4	22	1	0.00
JEvwo4_1	2	36	1	0.00
JEvwo4_1	3	36	1	0.00
JEvwo5_0	2	40	14	0.00
JEvwo5_0	10	40	14	0.00
JEvwo5_1	2	40	16	0.00
JEvwo5_1	10	40	16	0.00
JEvwo6_0	2	33	1	0.00
JEvwo6_0	3	28	1	0.00
JEvwo6_1	2	40	-	-
JEvwo6_1	3	40	-	-
JEvwo6_1	4	40	-	-
CDvwo5_0	4	28	2	0.00
CDvwo5_0	4	29	1	0.00
CDvwo5_0	10	28	2	0.00
CDvwo5_1	3	36	2	0.00
CDvwo5_1	3	37	1	0.00
CDvwo5_1	10	36	2	0.00
CDvwo5_initial	6	17	-	-
CDvwo5_initial	6	18	-	-
CDvwo5_initial	6	19	-	-
CDvwo5_initial	6	20	1	0.00
CDvwo5_initial	7	17	-	-
CDvwo5_initial	7	18	5	0.26
CDvwo5_initial	7	19	1	0.03
CDvwo5_initial	8	17	1	0.00

JEhavo5\_1. The coloring algorithm gives better results on this graph than the previous graph coloring problems, that didn't consider the color availability constraint.

## Chapter 7

# Color nuance class cardinality constraint

In this chapter the k-coloring problem with color nuances is discussed. The definition of this problem is already given in Chapter 2, but we recall it here.

**Definition 17 (k-Coloring with color nuance class cardinality constraint)** . *Given are a graph  $G = (V, E)$ , where  $V$  is divided into a partition  $V_0, \dots, V_m$ , a set of colors  $C$ , that is divided into a partition  $C_0, \dots, C_n$ , and for each  $V_i$  a maximum  $\max(V_i)$ . Then the k-Coloring problem with color nuance class cardinality constraint is to find a proper coloring on graph  $G$  where for each  $V_i$  and  $C_j$  at most  $\max(V_i)$  vertices are mapped on the elements in  $C_j$ .*

In fact we have considered the k-coloring with color availability restriction and color class nuances cardinality constraint. We can take the color class cardinality constraint into account, but in the computations in this chapter, we left this one out. In previous computations it showed that the color class cardinality constraint didn't make the problem much more difficult. The last step is the addition of color nuances. The set of color classes is partitioned into a number of subsets and the set of vertices is also partitioned into a number of subsets. For each subset of vertices an upper bound exists for the number of times that elements can occur in a subset of color classes. The partition of the set of vertices, the partition of the set of color classes and the maximums for each of the elements of the partition of the set of vertices are available with the definition of each of the graph instances and the accompanying color availability matrices.

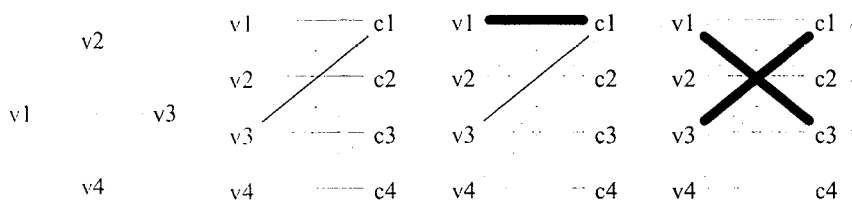


Figure 7.1: A clique with the corresponding matching graph and some steps in the computation process of the matching.

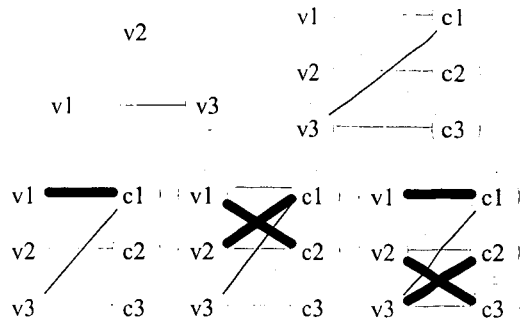


Figure 7.2: With a partition of  $V = \{\{v1\}, \{v2, v3\}\}$ , a partition of  $C = \{\{c1\}, \{c2, c3\}\}$  and all maximums equal to one, this matching ends up wrong.

The way of clique coloring that is used in Chapter 6 isn't useful anymore. When a vertex is matched with a color, then it can imply that edges should be removed from the bipartite matching graph. Even worse, when an augmenting path is found and the matching is augmented, then edges should be added to and removed from the existing matching. Then the case may occur that two edges  $e_1$  and  $e_2$  are added to the matching, while the addition of  $e_1$  implies that  $e_2$  can't exist in the matching, because  $e_1$  and  $e_2$  match two respective vertices from the same subset with two respective color classes from the same subset, while the upper bound for the subset of vertices is equal to one.

We have some examples of matchings with color nuance class cardinality constraint. In Figure 7.1 a matching is illustrated. There's a clique of 4 vertices shown in the upper left corner. These vertices should be matched with colors  $c1$ ,  $c2$ ,  $c3$  and  $c4$ .  $V$  is partitioned into two subsets and so is  $C$ .  $V = \{\{v1, v2\}, \{v3, v4\}\}$  and  $C = \{\{c1, c2\}, \{c3, c4\}\}$ . The maximum for both of the subsets of  $V$  is one. The corresponding matching problem is given in the upper right corner. In the bottom part of the figure, two steps in the matching process are shown. The edges that are forbidden because of the color nuance class cardinality constraint are shown as a discontinuous line. As we can see, there's one edge forbidden in the bottom left matching step. In the next step this edge is allowed again, but then two other edges are forbidden.

Figure 7.2 shows a matching that goes wrong. The upper two graphs are a clique and the corresponding matching problem. The color nuance class cardinality constraint is defined as follows.  $V$  is partitioned into  $\{\{v1\}, \{v2, v3\}\}$ .  $C$  is partitioned into  $\{\{c1\}, \{c2, c3\}\}$ . The maximums for the subsets of  $V$  are all equal to one. The three graphs on the bottom are the steps in the matching process. First  $(v1, c1)$  is inserted into the matching. Then an augmenting path is found:  $(v2, c1, v1, c2)$  and the matching is augmented. Still there are no complications. At last the matching is augmented another time, using the augmenting path  $(v3, c2, v1, c1, v2, c3)$ . But not both  $(v2, c3)$  and  $(v3, c2)$  can be in the matching, because of the color nuance class cardinality constraint.

As far as we know, there is no efficient way to color the clique with the available colors. We can use the same algorithm as for the coloring of the rest of the graph. There's no special way to color the clique, as far as we know. That means that the clique coloring part is NP-complete now.

Table 7.1: Computation times for the initial approach with color nuances. A '-' means that no coloring exists with  $\chi = 40$  with the constructed clique coloring and an empty space means the computation was aborted without results after 24 hours. CPU: Pentium III 533 MHz, RAM: 256 MB. OS: Linux.

graph	$t_{\text{ord}}$	$t_{r0}$	$t_{r1}$	$t_{r2}$	$t_{r3}$	$t_{r4}$	$t_{\text{mod}}$
JEhavo4.0	13.1	-	11.6	-	-	-	0.39
JEhavo4.1		-	-	-	-	-	
JEhavo5.0	0.08	0.09	0.08	0.08	0.08	0.08	0.08
JEhavo5.1							
JEvwo4_0	0.65	0.04	0.05	0.04	0.04	0.04	0.03
JEvwo4_1							
JEvwo5_0			-		-		-
JEvwo5_1		-	-		-	-	-
JEvwo6_0	0.06	0.08	0.07	0.07	0.07	0.07	0.04
JEvwo6_1					-		-
CDvwo5.0	0.07	0.07	0.08	0.08	0.07	0.07	0.07
CDvwo5.1	0.55	0.13	0.13	0.13	0.13	0.62	0.12
CDvwo5.initial	0.03	0.04	0.03	0.03	0.04	0.04	0.03

The verification step, after choosing a color, should be modified. An extra check should be added to the existing checks.

Our first implementation gives quite unsatisfactory results. As we can see in Table 7.1, no results were found for most of the graphs. We are still using the same way of limiting the search tree as we did in the previous graph coloring problem. The different columns give results for different color orders. These color orders are as follows.

$t_{\text{ord}}$ : The original color order is used, which is  $0, 1, 2, \dots, C - 1$ , where  $C$  is the number of colors that is tried.

$t_{r0}, \dots, t_{r4}$ : These are all random color sequences. In fact a random permutation of the colors that are available is generated and according to this permutation colors are tried to assign to the chosen vertex. The number represents the random seed that is used.

$t_{\text{mod}}$ : In our instances there are always 5 different color nuance classes, each of them containing 8 successive color classes and this color order is based on this property. First each of the first color classes in the color nuance classes are chosen, the second color classes, and so on. So the order is like  $0, 8, 16, 24, 32, 1, 9, \dots, 23, 31, 39$ .

The first improvement we were considering was the removal of the clique coloring. Instead the clique should be colored with the rest of the graph. This means that the approach of Couderc with a separate clique coloring is gone now. The results are shown in Table 7.2.

When comparing the results in Table 7.1 and Table 7.2, we can see that the computations that gave results in both cases generally take less time without separate clique coloring. A lot of computations finished without results with the separate clique coloring. Therefore, we think the coloring without separate clique coloring is preferable. If we don't use separate

Table 7.2: Computation times for the initial approach with color nuances. A '-' means that no coloring exists with  $\chi = 40$  with the constructed clique coloring and an empty space means the computation was aborted without results after 24 hours. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

graph	$t_{\text{ord}}$	$t_{r0}$	$t_{r1}$	$t_{r2}$	$t_{r3}$	$t_{r4}$	$t_{\text{mod}}$
JEhavo4_0		1.69			0.08		
JEhavo4_1							
JEhavo5_0	0.00	0.03	0.03	0.03	0.03	0.03	0.00
JEhavo5_1							
JEvwo4_0	0.86	0.02	0.02	0.02	0.03	0.02	0.01
JEvwo4_1		0.03		0.02	0.03	0.02	0.00
JEvwo5_0							
JEvwo5_1							
JEvwo6_0			0.03	0.02	0.02	0.03	
JEvwo6_1							
CDvwo5_0	0.00	0.04	0.03	0.03	0.04	0.04	0.00
CDvwo5_1		0.03	0.03	0.04		0.61	0.00
CDvwo5_initial	0.00	0.03	0.04	0.03	0.04	0.03	0.00

clique coloring, then the random color orders are the most promising. These orders have the advantage that they are very well suitable for a multiple run approach.

In order to improve the performance of the algorithm we change the choice of the next vertex for coloring. For each vertex the number of available colors is counted and based on this, the vertex that is supposedly the hardest to color is chosen. We also applied a multiple run approach with random color orders.

After this we have considered the specific properties of our coloring instances in order to improve the algorithm even more. We found two possibilities for further improvement of the performance. Here the aim is to speed up the computation by pruning branches in the search tree.

- We can try to remove symmetry in the colorings. In our instances, vertices in the same subset always have the same neighbors and most of the maximums of the subsets of vertices are equal to one. So, if colors are assigned to the vertices in a subset, then it doesn't matter which subset has which of the colors. They can always be exchanged. This implies that after trying one possibility, the other possibilities with the same colors don't need to be tried.

In our experiments we found reductions of the computation time of up to 99%. On the other hand there were also cases where there was no significant improvement in the computation times.

- Each of the subsets in the partition of the vertices has a maximum value for the number of colors from each color nuance class that can be used. As most of these maximums are equal to one and vertices in the same subset have the same neighborhood in our instances, we can compute the number of color nuances from which colors are available

for each of the subsets. If there's a subset that needs more color nuances than there are available, then there won't be a color possible, given the already existing coloring. The subsets should all be checked before a vertex is colored. We think this will take to much time for practical use.

But we can apply a weaker form of this theorem efficiently for subsets of vertices that have a maximum for each color nuance equal to one. When the first vertex from such a subset is chosen for coloring, then the number of available color nuances is calculated. If the number is insufficient, then it's useless to continue coloring these vertices and the coloring can be pruned here. Otherwise the coloring can continue.

It appears that vertices of a subset are all colored consecutively and the colors are chosen at random. Therefore, it's unlikely that repeating the calculation for the other vertices of the subset saves even more time.

A number of colors are already forbidden because of the colors in the neighborhood and the color availability constraint. We only need to find the subsets to which the other colors belong.

The reduction in computation time is not as spectacular as in the first improvement. Still we found reductions in the computation time of up to 85%. And again there were instances where we didn't see a significant improvement of the computation times.

Note that these improvements are only possible if the vertices from the same subset have the same neighborhood outside this subset. Otherwise essentially different colorings may be found by exchanging colors among the vertices in a subset.

Finally we have implemented these improvements. The search space has to be limited again. This time this is done by limiting the allowed number of backtracks. First the number of allowed backtracks is limited to 100.000. It appears that it becomes more difficult to find a coloring. As soon as a coloring approaches the last 10 vertices, it tends to go straight to a complete proper coloring. If the strategy of different colorings for the first vertices leads to a complete coloring, then it probably leads to a complete coloring anyway because of the multiple run strategy with different random color orders.

Table 7.3 shows the results of two slightly different algorithms. *Random A* uses the same random color order for each of the vertices. Instead *Random B* uses a different random color order for each of the vertices. *Random A* has shorter computation times per successful run. For unsuccessful runs however the differences are not that obvious. *Random A* also mostly has more successful runs.

In general we can conclude that the success of the coloring depends highly on the size of the maximum clique. Also the density of the graph is important. The larger the maximum clique and the higher the density, the harder it becomes to find a coloring.

Finally we wanted to see how the algorithm performs on a new range of graphs. Some of the graphs in the new range were also among the instances we have used before. Most of them however are new graphs. The graphs are intermediate results of an other algorithm. The graphs ending in `_initial` are the smallest graphs that were colored and the graphs ending in `_final` are the final graphs that represent the generated timetable. So, these graphs are colorable when taking all of the constraints into consideration.

The final tests for the algorithm are the really big graphs where all of the graphs are combined to create a timetable for all considered classes together. If it can find proper colorings for

Table 7.3: Computation times for *Random A* and *Random B*. '+', '-': number of successful respectively unsuccessful runs. 't+', 't-': accumulated time for successful respectively unsuccessful runs. In any case there were 100 different runs. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

Graph	<i>Random A</i>				<i>Random B</i>			
	+	t+	-	t-	+	t+	-	t-
JEhavo4_0	93	27.9	7	158	76	75.2	24	577
JEhavo4_1	61	108	39	902	11	58.8	89	2187
JEhavo5_0	100	1.54	0	0.00	100	4.28	0	0.00
JEhavo5_1	4	3.21	96	1559	2	16.8	98	1812
JEvwo4_0	100	1.04	0	0.00	100	2.89	0	0.00
JEvwo4_1	100	1.13	0	0.00	100	4.33	0	0.00
JEvwo5_0	0	0.00	100	1429	0	0.00	100	1463
JEvwo5_1	0	0.00	100	1359	0	0.00	100	1401
JEvwo6_0	100	0.98	0	0.00	100	3.09	0	0.00
JEvwo6_1	41	38.3	59	405	47	26.5	53	389
CDvwo5_0	100	2.25	0	0.00	100	5.81	0	0.00
CDvwo5_1	100	2.00	0	0.00	100	5.10	0	0.00
CDvwo5_initial	100	2.06	0	0.00	100	5.16	0	0.00

these graphs, taking all constraints into consideration, then it is powerful enough to be used in practise.

It appears that not only the size of the largest clique and the density are important in the success probability, but also the size is important. For larger graphs, the computation time is increasing, but still the final results are worse. Considering the sizes of the largest cliques of both the CDall.final and JEall.final graphs and also the sizes of these graphs, it is very unlikely that a coloring can be found.

To confirm this assumption, we decided to do some extra experiments on these graphs. CDall.initial and JEall.initial are unsuccessful in three cases. We decided to find out how many vertices are actually colored in those graphs. Therefore, we ran the algorithms *Random A* and *Random B* on both of the graphs to find out. That gave us the following results.

- *Random A* colors 423 of 670 vertices of CDall.initial with random seed 57.
- *Random A* colors 225 of 654 vertices of JEall.initial with random seed 79.
- *Random B* colors 227 of 654 vertices of JEall.initial with random seed 66.

Subsequently we have tried to find out how many vertices were colored in CDall.final and JEall.final by both of the algorithms *Random A* and *Random B*. We put those data in histograms. These are shown in Figure 7.3, Figure 7.4, Figure 7.5 and Figure 7.6. In fact both of the algorithms succeeded in coloring at most 169 vertices of the CDall.final graph and 202 vertices of the JEall.final graph. These graphs have a larger maximum clique size, a far higher density and now we can also see that in the best case the number of vertices that are

Table 7.4: Computation times for *Random A* and *Random B* on Cambreur instances. '+' , '-': number of successful respectively unsuccessful runs. 't+', 't-': accumulated time for successful respectively unsuccessful runs. In any case there were 100 different runs. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

Graph	V	E	$\frac{ E }{ V }$	$\gamma$	<i>Random A</i>				<i>Random B</i>			
					+	t+	-	t-	+	t+	-	t-
CDmavo4_initial	94	395	4	16	100	1.59	0	0.00	100	4.49	0	0.00
CDmavo4_final	94	3747	40	40	5	16.3	95	1580	7	15.8	93	1635
CDmavo4_initial	72	424	6	16	100	1.01	0	0.00	100	3.25	0	0.00
CDmavo4_final	72	2136	30	38	89	31.3	11	120	76	54.2	24	278
CDhavo4_initial	142	878	6	22	100	3.53	0	0.00	100	7.90	0	0.00
CDhavo4_final	142	7079	50	40	0	0.00	100	3321	0	0.00	100	3871
CDhavo5_initial	100	642	6	21	100	2.00	0	0.00	100	5.11	0	0.00
CDhavo5_final	100	4162	42	40	3	2.49	97	1976	0	0.00	100	2339
CDath4_initial	77	462	6	19	100	1.20	0	0.00	100	3.57	0	0.00
CDath4_final	77	2044	27	39	93	10.56	7	70.3	33	12.7	67	704
CDath5_initial	100	433	4	17	100	2.06	0	0.00	100	5.16	0	0.00
CDath5_final	100	4277	43	40	5	30.7	95	1682	0	0	100	2148
CDath6_initial	85	372	4	20	100	1.36	0	0.00	100	4.00	0	0.00
CDath6_final	85	3117	37	39	14	18.3	86	1037	12	23.0	88	1107
CDall_initial	670	6248	9	24	99	76.5	1	78.9	100	179	0	0.00
CDall_final	670	26079	39	40	0	0.00	100	4-5 h	0	0.00	100	4-5 h

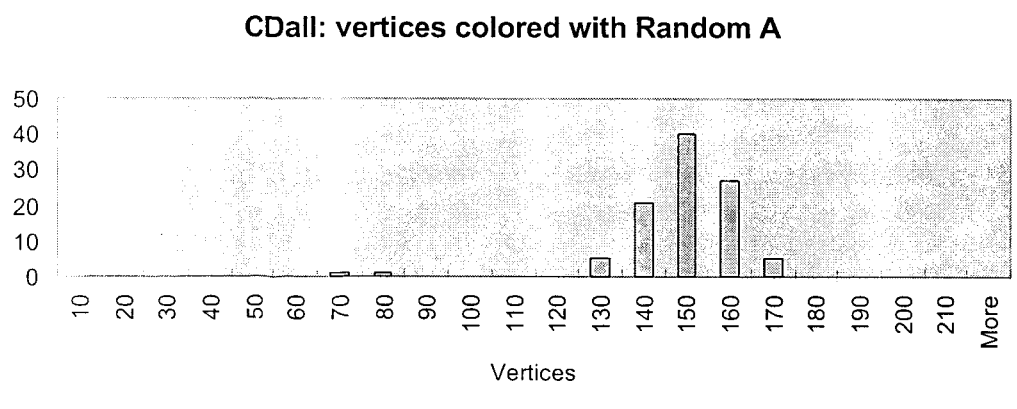


Figure 7.3: The maximum number of vertices that are colored by *Random A* in the graph *CDall\_final*.

Table 7.5: Computation times for *Random A* and *Random B* on Joris instances. '+', '-': number of successful respectively unsuccessful runs. 't+', 't-': accumulated time for successful respectively unsuccessful runs. In any case there were 100 different runs. CPU: Pentium III 533 MHz, RAM: 256 MB, OS: Linux.

Graph	V	E	$\frac{ E }{ V }$	$\gamma$	<i>Random A</i>				<i>Random B</i>			
					+	t+	-	t-	+	t+	-	t-
JEmavo3_initial	99	800	8	22	100	1.87	0	0.00	100	4.90	0	0.00
JEmavo3_final	99	2563	26	40	70	41.0	30	275	40	17.8	60	381
JEmavo4_initial	111	469	4	12	100	2.22	0	0.00	100	5.58	0	0.00
JEmavo4_final	111	4709	42	39	0	0.00	100	1599	0	0.00	100	1912
JEhavo4_initial	136	827	6	22	100	3.31	0	0.00	100	7.44	0	0.00
JEhavo4_final	136	6712	49	40	1	0.82	99	2972	0	0.00	100	3053
JEhavo5_initial	90	670	7	28	100	1.55	0	0.00	100	4.27	0	0.00
JEhavo5_final	90	3040	34	40	30	5.92	70	748	0	0.00	100	1229
JEvwo4_initial	67	617	9	22	100	0.94	0	0.00	100	2.91	0	0.00
JEvwo4_final	67	1571	23	35	100	1.06	0	0.00	100	3.09	0	0.00
JEvwo5_initial	78	673	9	28	100	1.20	0	0.00	100	3.59	0	0.00
JEvwo5_final	78	2731	35	40	30	85.3	70	874	34	59.2	66	885
JEvwo6_initial	66	693	11	28	100	0.93	0	0.00	100	2.93	0	0.00
JEvwo6_final	66	1909	29	39	86	9.24	14	136	84	15.4	16	159
JEall_initial	654	7956	12	30	99	72.2	1	139	99	91.2	1	136
JEall_final	654	26497	41	40	0	0.00	100	4-5 h	0	0.00	100	4-5 h

CDall: vertices colored with Random B

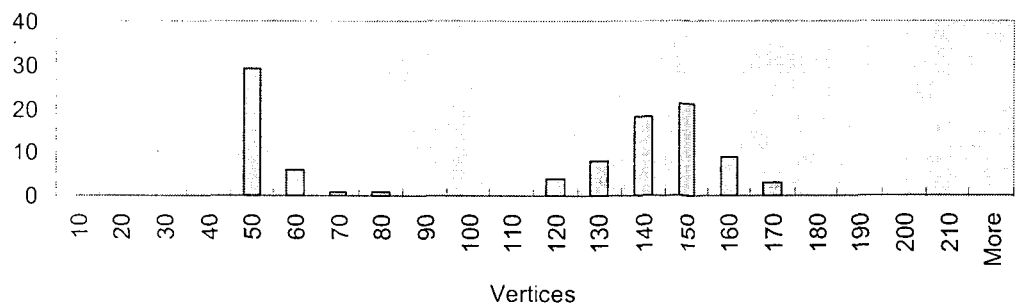


Figure 7.4: The maximum number of vertices that are colored by *Random B* in the graph CDall\_final.

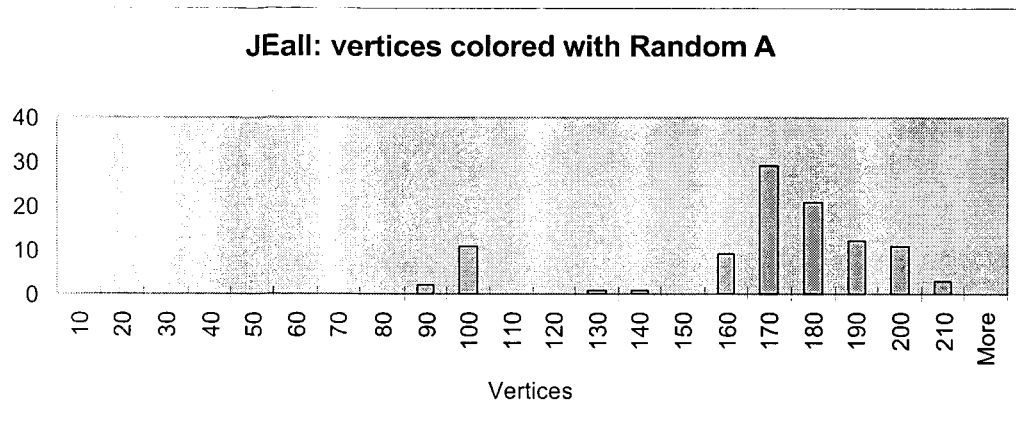


Figure 7.5: The maximum number of vertices that are colored by *Random A* in the graph *JEall\_final*.

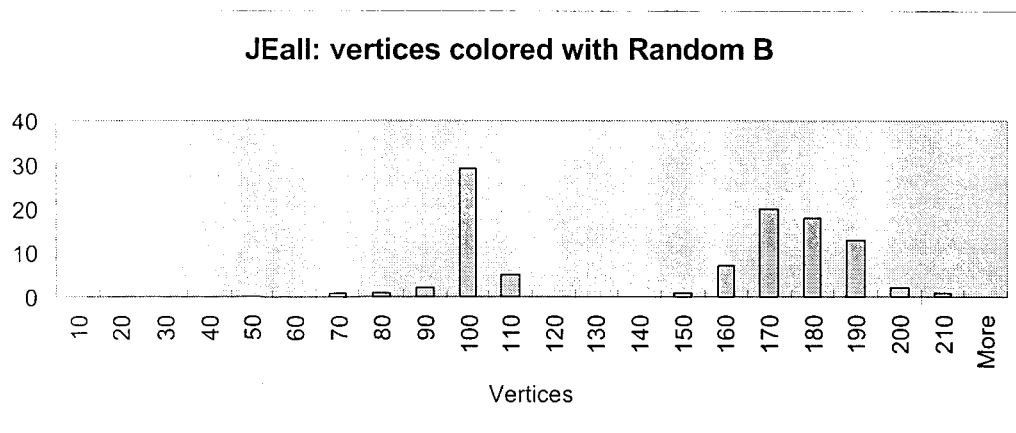


Figure 7.6: The maximum number of vertices that are colored by *Random B* in the graph *JEall\_final*.

colored is smaller than the number of vertices that is colored in each of the CDall\_initial and the JEall\_initial graphs in the worst case.

So probably the algorithm is not powerful enough to color these large graphs. Still we know that there exists a coloring for these graphs, because these graphs have proved to be colorable by Willemen's algorithm. Although the comparison is not a fair one.

His timetabling algorithm does something more than our algorithm does. In fact, in our case the algorithm assigns lessons with the related students and teacher to time slots. Willemen's algorithm however considers a valid timetable, which is a colored graph, and adds students to some lessons, which means that some edges are inserted in the graph. The consequence is that this may make the existing coloring invalid. Then a new proper coloring should be found. If the algorithm doesn't find such a proper coloring, then the insertion of the edges is undone and another insertion is done in order to find a proper coloring. The choice between different possibilities for inserting edges corresponds to the question whether to add a student to class A or class B for some subject. This implies that in fact there may be lots of different graphs that are colored in parallel. Finally a coloring should be found for just one of them, but it doesn't matter which one. And when recoloring the graph, a major part of the graph may already be colored properly. Then the existing coloring may be closer to a complete coloring than our algorithm, at least for the largest graphs we have been considering. In our algorithm, the graph is given and a coloring for just that graph should be found from a completely uncolored graph.

# Chapter 8

## Future work

In this chapter some topics are discussed for future work. They are not investigated in detail in the project.

### 8.1 Lower bound on the number of colors

For the minimum coloring, we used the size of the largest clique of the graph as a lower bound on the number of colors for the coloring. For the later  $k$ -coloring based problems, the size of the largest clique was used to check whether the value of the  $k$  was not too low. The largest clique is also used to find a partial coloring of the graph easily. But it is also possible to find minimum colorings for even larger subsets of vertices.

#### 8.1.1 Coloring two cliques

One way to color a larger subset of a graph than a maximum clique is to try to find two large cliques and find the minimum number of colors that is needed to color the two cliques. This can be done in an efficient way as we show below.

Two cliques can be found in several different ways. One way to find them is to find a maximum clique first and then find another maximum clique in the rest of the graph. In general we want to find two large cliques that also have a lot of edges between them. If we have a maximum clique, then we know that the lower bound for the number of colors is at least the same as the maximum clique size that we were using in our graph coloring algorithms.

Let's say in some way we have found two cliques,  $C_1$  and  $C_2$ . We want to color them efficiently. In each of the cliques, each color is used once. But there are pairs of vertices  $\{v_1 \in C_1, v_2 \in C_2\}$  in our subset, that can have the same color. This is the case if  $v_1$  and  $v_2$  are not connected by an edge.

The complementary graph of  $C_1 \cup C_2$  is a bipartite graph, because it only has edges that connect a vertex from  $C_1$  with a vertex from  $C_2$ . The edges connect vertices that can have the same color in the original graph. Now the problem is to find a maximum matching in the complementary graph. If an edge is in the matching, then the two corresponding vertices can have the same color. Now the goal is to find a maximum matching. After the computation of the maximum matching, the colors should be assigned to the vertices in the graph. Each pair of vertices that is in the matching is assigned the same color. Other vertices are assigned unique colors. Let's say that the number of edges in the matching is equal to  $x$ . Then the

number of colors that we need to color the clique is equal to  $\chi(C_1 \cup C_2) = |C_1| + |C_2| - x$ . This can be used as a lower bound on the total number of colors to color the whole graph. The coloring that we used to compute the lower bound may not lead to a minimum coloring of the whole graph.

We can apply some efficient heuristic to find two good cliques. The maximum matching problem can also be solved efficiently. Therefore, we can determine this lower bound efficiently.

### 8.1.2 coloring quasi-cliques

Another way to find a lower bound is to color a quasi-clique. This is considered in detail in [1]. The general idea is to find a subset of vertices that almost form a clique. Each vertex may miss some edges, at most  $y$ . Then the complementary graph of this subset is computed. In this graph, the vertices have degree at most  $y$ . If  $y = 1$ , then the vertices that are connected by an edge with an other vertex, are assigned the same color. If  $y = 2$ , then it is also easy to color the subset with a minimum number of colors.

1. Uncolored vertices that have one neighbor in the uncolored subset of the complementary graph are assigned the same colors as their neighbors. This is repeated until no such uncolored vertices exist.
2. If an uncolored vertex  $v_1$  with two neighbors in the uncolored subset of the complementary graph exists, then one of the neighbors is chosen at random. We call this vertex  $v_2$ . Now  $v_1$  and  $v_2$  are assigned the same color and then step 1 is repeated. Otherwise if no uncolored vertices with two neighbors in the uncolored subset of the complementary graph exist, then only vertices without neighbors exist. These are all assigned a different color that is unique in the quasi-clique.

So we can see that in this way also larger subsets of a graph can be colored easily in an exact way. Of course a precondition is that the quasi-clique should be found in an efficient way.

## 8.2 Improvements on the coloring

There are still several possibilities for improvements on the coloring process. We found some of them to be interesting for future research.

### 8.2.1 The order of vertices

While coloring the graph, the choice of the next vertex for coloring is quite essential. One way to find a good order for the vertices is to start with some order and find out which vertex is the one that blocks the further coloring of the graph. That should a vertex that can't be colored in some branch of the coloring tree, that is chosen. This vertex is put first in the order. Then the coloring is restarted from the beginning. In this way we hope to find a better order that enables the algorithm to color the graph within a reasonable amount of time.

### 8.2.2 Removing symmetry

In the graph coloring problem with color nuances we tried to remove symmetry from the coloring tree. In general we can try to find these subsets by analyzing the set of vertices.

If two vertices  $v_1$  and  $v_2$  have the same degree and they are each others neighbors, then we can see whether they have their neighborhoods in common, except for each other. If so, then they can be put together in a subset. More vertices can be added if they form a clique with the vertices in the subset and if they have the same neighborhood outside the clique. Considering this, we can build the subsets. The subsets form a partition together. These subsets of vertices are taken as a base to remove symmetry. These subsets may even be larger than the subsets we have used for the color nuances.

### 8.2.3 Too few colors available

For each subset of vertices there is a number of colors available within some color nuance classes. And there is a maximum on the number of colors that is used from a certain color nuance class. Considering this, the number of colors that can be used, can be computed. We only considered this for the subset from which a vertex is chosen for coloring and only if the vertex is the first of that subset. In fact this computation can be made for each of the subsets that still have uncolored vertices and the computation can be repeated for each new vertex. If it is sure that not enough colors are available to color the vertices in some subset, than the recursion can be pruned. The extra computation time may be too much compared to the practical gain, but there may also be some special cases where this may be profitable. In our approach we assumed that this would have negative effects on the computation time on our instances. In an other approach, with different instances, this may be an improvement. We don't know.

### 8.2.4 Cliques and quasi-cliques

For some types of subsets of vertices it is easy to find a minimum coloring. Examples are cliques, quasi-cliques and subsets that consist of two different cliques. If the number of colors that is needed for such a subset of uncolored vertices, is higher than the number of colors that are available for these vertices, then the recursion in the coloring can be pruned.

## Chapter 9

# Conclusions

We have considered some different graph coloring problems. The first goal was to verify the theorem by Coudert that graph coloring in practise would be easy in the case that a graph was 1-perfect, using his algorithm. We implemented this minimum graph coloring algorithm to prove it. Among our instances were some graphs that were not colored. These instances had graphs that were either not 1-perfect or a counter example for his theorem. We couldn't find a proof for either two cases.

We continued with a timetabling problem. Because of the specific requirements, we decided to use a k-coloring algorithm, based on the minimum coloring algorithm by Coudert. We considered three extra constraints for the graph coloring algorithm. In this way we considered four different graph coloring problems for timetabling. For all of the problems the same graphs were used.

**k-Coloring** . We couldn't find a valid coloring for one instance among our 13 instances within 24 hours of computation time. The others were solved quickly.

**Graph coloring with color class cardinality constraint** . This is the k-coloring with color class cardinality constraint. Here we found solutions for the same instance as in the previous graph coloring problem. But we also found a solution for the instance for which we couldn't find a solution by using k-coloring.

**Graph coloring with color availability restriction** . This is the previous graph coloring problem with the extension of color availability restriction. Now, it appeared to be a little more difficult to find solutions. We applied a multiple run strategy. In this way we still had a reasonable result. The algorithm wasn't successful on one of the instances. This was an other instance as for the k-coloring.

**Graph coloring with color nuance class cardinality constraint** . This is the previous graph coloring problem with the extension of the color nuance class cardinality constraint. With some extra improvements we finally succeeded in finding solutions for 11 of the 13 instances.

Finally some computations were done with some extra graphs and considering all of the constraints. This showed comparable results for graphs that were comparable in size and density with the original instances. These were graphs of between 50 and 150 vertices and a density of at most 40 edges per vertex (average degree at most 80). Only some graphs were

larger. They had 654 and 670 vertices respectively and a density of about 40. These instances appeared too hard for our algorithm. Not even a third of those graphs could be colored.

# Bibliography

- [1] M.W. Carter and D.G. Johnson. *Extended clique initialisation in examination timetabling*. Journal of the Operational Research Society 52, 538-544, 2001.
- [2] O. Coudert. *Exact coloring of real-life graphs is easy*. 34th Design automation conference, pp. 121-126, 1997.
- [3] John Englebretson & Max Hailperin. *Investigation into the Near-Optimality of Merge-Enhanced Graph Coloring*.
- [4] D.S. Johnson & M.R. Garey. *A guide to the theory of NP-completeness*. W.H. Freeman & Company, New York, 1979.
- [5] Darko Kirovski and Miodrag Potkonjak. *Efficient Coloring of a Large Spectrum of Graphs*. DAC 98, June 15-19, 1998, San Francisco, CA USA.
- [6] D. de Werra. *On a multiconstrained model for chromatic scheduling*. Discrete Applied Mathematics 94, 171-180, 1999.
- [7] R.J. Willemsen. *School timetable construction: Algorithms and complexity*. Ph. D. thesis, department of computing science, Eindhoven University of Technology, The Netherlands, 2002.