SUBOPTIMAL COLORINGS AND SOLUTION OF LARGE CHROMATIC SCHEDULING PROBLEMS

THÈSE N^{O} 3363 (2005)

PRÉSENTÉE À LA FACULTÉ SCIENCES DE BASE

Institut de mathématiques

SECTION DE MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Ivo BLÖCHLIGER

ingénieur mathématicien diplômé EPF
de nationalité suisse et originaire d'Ernetschwil (SG)

acceptée sur proposition du jury:

Prof. D. de Werra, directeur de thèse
Prof. M.-C. Costa, rapporteur
Prof. M. Hasler, rapporteur
Prof. M. Widmer, rapporteur

Lausanne, EPFL
2005

Je pense, donc je suis.

René Descartes

Je désire, donc j'existe.

B. & B.

Remercîments

Tout d'abord je tiens à remercier le professeur Dominique de Werra pour son soutien, sa confiance en moi et son enthousiasme. Je remercie les professeurs Marie-Christine Costa, Martin Hasler et Marino Widmer d'avoir accepté de figurer dans mon jury de thèse. Je suis très reconnaissant pour le temps qu'ont pris plusieurs de mes amis pour relire des bouts de ma thèse et qui ont ainsi permis d'améliorer ce travail grâce à leur remarques. Je tiens en particulier à remercier mon ami Robert d'avoir corrigé intégralement l'anglais du présent texte. C'était un grand plaisir de travailler au sein de la ROSE où règne une atmosphère particulièrement agréable. Un grand merci à Jocelyne pour sa disponibilité et sa gentillesse, autant pour avoir envoyé mes lettres que pour avoir écouté mes histoires. Pour l'ambiance dans la chaire je remercie mes collègues : Nicolas pour son hospitalité et sa collaboratation fructueuse, David pour avoir supporté la même musique en boucle et pour m'avoir laissé la vache volante, Sacha pour sa compagnie lors de la découverte des fjords norvégiens, Tınaz pour sa joie de vivre et sa capacité d'écoute, Bernard pour me battre au squash de temps en temps, Benjamin pour dépanner le dépanneur, Sem cette espèce de collègue de bureau pour son amitié et Daniela pour ses idées brillantes et son sourire rayonnant. Ma vocation de mathématicien n'est pas du pur hasard : déjà mon grand-père Rudolf Blöchliger était un mathématicien autodidacte et passionné et c'est avec ses livres que j'ai eu mon premier contact avec les chiffres à l'école. J'ai eu la chance d'avoir un excellent professeur de mathématiques au gymnase de St-Gall, le professeur W. Nüesch. C'est également son enthousiasme qui m'a amené à étudier les mathématiques. C'est lors de mon travail de diplôme avec le professeur A. Hertz que j'ai pris goût pour la coloration de graphes. Faire des études en mathématiques n'aurait pas été possible sans le support inconditionnel de mes parents. Je leur suis très reconnaissant. Étant devenu le seul romand dans ma famille, je prie mes proches de m'excuser de ne pas avoir visité ma terre natale plus fréquemment.

Je remercie tous mes amis qui m'ont fait prendre des racines au bord du Léman et avec qui j'ai partagé tant de bon moments.

Abstract

Graph Coloring is a very active field of research in graph theory as well as in the domain of the design of efficient heuristics to solve problems which, due to their computational complexity, cannot be solved exactly (no guarantee that an optimal solution will be reported), see [Cul] for a list of over 450 references. The graph coloring problem involves coloring the vertices of a given graph in such a way that two adjacent vertices never share the same color. The goal is to find the smallest number of colors needed to color all vertices in a fashion that satisfies this requirement. This number is called *chromatic number* and is denoted by χ.

In the first chapter, we present our research on suboptimal colorings and graphs which can be colored in such a way that the number of different colors appearing on the *closed neighborhood* (a vertex plus its neighbors) of any vertex v is less than χ. We call such graphs *oligomatic*. The most interesting result is the following: given a graph and a coloring using $\chi + p$ colors, there exists a vertex v such that there are at least χ different colors among all vertices which are at a distance of $\lceil \frac{p}{2} \rceil + 1$ or less from v. We also study the existence of oligomatic graphs in special classes. Additionally, we present results of research on *universal graphs* which are "generic" oligomatic graphs in the sense that most properties of oligomatic graphs can be analyzed by restricting ourselves to universal graphs.

Chapters Two to Four deal with the development of heuristics for two types of graph coloring problems. The *tabu search heuristic* was a central focus of our research. A *tabu search* iteratively modifies a candidate solution (which becomes the new candidate solution) with the goal of improving it. In such a procedure it is forbidden (tabu) to undo a modification for a certain number iterations. This mechanism allows to escape from local minima.

In Chapter Two, we propose general improvements for *tabu search* based on some new and simple mechanisms (called *reactive tabu schemes*) to adapt the *tabu tenure* (which corresponds to the number of iterations a modification stays tabu). We also introduce distance and similarity measures for graph coloring problems which are needed in iterative procedures such as those of the *tabu search*.

In the third chapter, we present the PARTIALCOL heuristic for the graph coloring problem. This method obtains excellent results compared to other similar methods and is able to color the well known DIMACS [JT96] benchmark graph flat300_28_0 optimally with 28 colors. (The best colorings found to date by other researchers use at least 31 colors.) PARTIALCOL uses partial solutions in its search space, which is a little explored way of approaching the graph coloring problem. Most approaches either use colorings and try to minimize the number of colors, or they use improper colorings (having conflicts, i.e. adjacent vertices which may have the same color) and try to minimize the number of conflicts.

In the final chapter, we present a weighted version of the graph coloring problem which

has applications in batch scheduling and telecommunication problems. We present two different adaptations of *tabu search* to the weighted graph coloring problem and test several of the reactive tabu schemes presented in Chapter Two. Further, we devise an adaptive memory algorithm AMA inspired by genetic algorithms. A large set of benchmark graphs with different properties is presented. All benchmark graphs with known optima have been solved to optimality by AMA. A key element of this algorithm is its capacity to determine the number k of colors to be used. Most other graph coloring heuristics need this parameter to be supplied by the user. Considering the results obtained on various graphs, we are confident that the methods developed are very efficient.

Version abrégée

La coloration de graphes est un sujet de recherche très actuel, dans la théorie des graphes ainsi que pour la conception d'heuristiques pour la résolution de problèmes qui ne peuvent pas être résolus exactement parce que le temps de calcul nécessaire croît trop vite avec la taille du problème. Le problème de la coloration de graphes (Gcp) consiste à colorer les sommets d'un graphe d'une manière que deux sommets adjacents ne reçoivent jamais la même couleur. La question est quel est le nombre minimum de couleurs nécessaires pour colorer un graphe donné ? Ce nombre est appelé *nombre chromatique* et est noté par χ (prononcé ki).

Au premier chapitre nous présentons les résultats de nos recherches sur les *colorations suboptimales* et des graphes qui peuvent être colorés de sorte que le nombre de couleurs apparaissant sur chaque sommet et ses voisins soit inférieure à χ. Nous appelons ces graphes *oligomatiques*. Le résultat le plus important est le théorème suivant : Pour un graphe coloré avec $\chi + p$ couleurs, il existe un sommet v tel qu'il y a au moins χ couleurs différentes sur l'ensemble des sommets qui se trouvent à une distance d'au plus $\lceil \frac{p}{2} \rceil + 1$ de v. Nous avons également étudié l'existence de graphes oligomatiques dans différent classes de graphes. Nous présentons également des résultats sur les *graphes universels* qui servent comme modèle pour les graphes oligomatiques : il suffit souvent, en effet, d'étudier les graphes universels pour en dériver des propriétés des graphes oligomatiques.

Nous développons des améliorations de l'heuristique *tabu search* appliquée au Gcp ainsi qu'à une généralisation pondérée du Gcp. Puis nous introduisons plusieurs méthodes, appelées *reactive tabu schemes*, pour ajuster automatiquement la *tabu tenure*, le paramètre crucial pour *tabu search*. Pour ces méthodes, nous introduisons des mesures de distance et de similarité entre deux colorations d'un graphe.

Au troisième chapitre nous présentons l'heuristique Partialcol pour le Gcp. Cette heuristique obtient des résultats excellents sur des graphes de test Dimacs. En particulier cette méthode est capable de colorer le graphe flat300_28_0 avec le nombre optimal de 28 couleurs, ce qu'aucune autre méthode connue à ce jour n'a pu obtenir (la meilleure coloration trouvé utilisait 31 couleurs). Partialcol utilise des colorations partielles (certains sommets ne sont pas colorés), ce qui est une méthode peu explorée dans le cadre du Gcp.

Nous présentons deux adaptations de *tabu search* à une version pondérée du Gcp. La première utilise des colorations avec un nombre variable de couleurs, tandis que la deuxième utilise des colorations avec conflits (des sommets adjacents peuvent avoir la même couleur) avec un nombre de couleurs fixé par l'utilisateur. Avec les deux approches, plusieurs *reactive tabu schemes* ont été testés. Basé sur la première approche, nous avons développé un algorithme à mémoire adaptative (Ama). Pour tester ces heuristiques, nous avons généré un jeux d'instances avec différentes caractéristiques. Toutes les instances avec un optimum connu ont été résolues par Ama. Une propriété importante de cet algorithme est le fait qu'il détermine automatiquement le nombre de couleurs à utiliser, contrairement à beaucoup d'autres heuristiques de coloration qui demandent ce paramètre comme entrée.

Les résultats obtenus avec les heuristiques développées confirment leur efficacité et le bien-fondé des idées sous-jacentes. Ces techniques vont très certainement permettre d'aborder d'autres problèmes d'optimisation combinatoire et d'améliorer substantiellement les performances atteintes jusqu'ici.

Zusammenfassung

Graphenfärbung ist ein aktives Forschungsfeld der Graphentheorie und der angewandten Mathematik als Testproblem für die Entwicklung von Heuristiken, um Probleme zu lösen, die nicht exakt gelöst werden können, weil der Rechenzeitbedarf sehr schnell ins Unermässliche steigt. Das Graphenfärbungsproblem (kurz GCP für *graph coloring problem*) besteht darin, die Knoten eines Graphen so einzufärben, dass verbundene Knoten immer verschiedene Farben haben. Die Frage ist nun, wie viele Farben sind mindestens nötig, um einen gegebenen Graphen zu färben? Diese minimale Anzahl Farben ist *chromatische Zahl* genannt und wird mit χ ("chi") bezeichnet. [Cul] führt über 450 Verweise im Zusammenhang mit dem GCP auf.

Im ersten Kapitel präsentieren wir die Forschungsergebnisse bezüglich *suboptimaler Färbungen* und speziell Graphen, die so eingefärbt werden können, dass die Anzahl verschiedener Farben eines Knoten und seiner Nachbarn kleiner ist als χ. Wir nennen solche Graphen *oligomatisch*. Das wichtigste Resultat ist das folgende Theorem. Gegeben sei ein Graph und eine Färbung mit $\chi + p$ Farben. Dann gibt es einen Knoten v, so dass mindestens χ verschiedene Farben auf den Knoten vorkommen, die nicht weiter als $\lceil \frac{p}{2} \rceil + 1$ von v entfernt sind. Des weiteren haben wir für verschiedene Graphenklassen untersucht, ob darin oligomatische Graphen enthalten sind. Wir präsentieren ebenfalls unsere Resultate bezüglich universellen Graphen, welche oligomatische Modellgraphen darstellen. Für viele Fragen über oligomatische Graphen ist es ausreichend, universelle Graphen zu betrachten.

In den folgenden Kapiteln beschäftigen wir uns mit der Entwicklung und Verbesserung von Heuristiken für das GCP und für eine gewichtete Verallgemeinerung des GCP. Im zweiten Kapitel stellen wir mehrere Methoden vor, um die *tabu search* Heuristik zu verbessern, indem der wichtigste Parameter mittels einer *reactive tabu tenure* automatisch angepasst wird. Für diese Methoden führen wir verschiedene Masse ein, um die Ähnlichkeit zweier Färbungen zu quantifizieren.

Im dritten Kapitel stellen wir PARTIALCOL, eine Heuristik für das GCP vor. Diese Heuristik erzielt sehr gute Resultate und hat für den DIMACS Testgraphen flat300_28_0 erstmals eine optimale Färbung mit 28 Farben bestimmen können. Die beste je erreichte Färbung mit anderen Methoden brauchte 31 Farben. PARTIALCOL verwendet Teilfärbungen im Suchraum, was eine kaum erforschte Methode für das GCP darstellt. Einige Methoden benutzen Färbungen des ganzen Graphen und versuchen die Anzahl verwendeter Farben zu reduzieren. Viele andere verwenden unechte Färbungen, die Konflikte enthalten können (benachbarte Knoten gleicher Farbe) und versuchen die Anzahl Konflikte auf Null zu reduzieren.

Im letzten Kapitel beschäftigen wir uns mit einer gewichteten Version des GCP. Wir präsentieren zwei Heuristiken, die auf *tabu search* beruhen. Die erste verwendet Färbungen mit einer variablen Anzahl Farben und die zweite verwendet unechte Färbungen mit einer fixen, vom Benutzer zu definierenden Anzahl Farben k. Basierend auf der ersten Methode haben wir zudem einen genetischen Suchalgorithmus entwickelt. Um die Heuristiken zu

testen, haben wir mehrere Testinstanzen mit verschiedenen Eigenschaften generiert. Alle Testinstanzen mit bekanntem Optimum wurden von unserem genetischen Algorithmus gelöst. Eine wichtige Eigenschaft unseres Algorithmus ist, dass er die Anzahl Farben für eine beste Färbung selbst bestimmt. Im Gegensatz muss bei den meisten anderen Graphenfärbungsheuristiken dieser Parameter vom Benutzer eingegeben werden. In Anbetracht der Resultate dieser Heuristiken sind wir überzeugt, dass die entwickelten Methoden sehr effizient sind.

Contents

Introduction

The *graph coloring problem* is one of the oldest problems in graph theory and combinatorial optimization.

Its origins go back to the famous *four color theorem*, which states that the countries of any geographical map can be colored with four colors in such a way that two countries having a common border are never colored with the same color (provided all countries are contiguous). This question has already been raised in the middle of the 19$^{\text{th}}$ century, and the first published reference dates back to 1879 [Cay79]. The theorem was proved only in 1976 [AH77, AHK77] with the help of a computer to check thousands of configurations. A short and elegant proof is still not available.

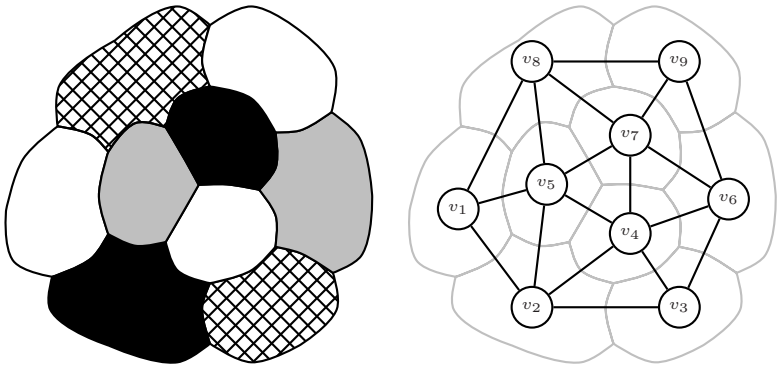

FIGURE 1 A four-colored map and the graph associated to the map.

Figure 1 illustrates the *four color theorem* on a map with 9 countries. The associated graph is constructed as follows. Every country corresponds to a vertex, and two vertices are joined by an edge if the two countries associated with the vertices have a common border.

A *coloring of the vertices of a graph*, or simply a *graph coloring*, is defined as an association of a color with each vertex such that two adjacent vertices never have the same color.

The graph given in Figure 1 cannot be colored with fewer than four colors. To see that, consider the set of vertices $\{v_2, v_3, v_6, v_7, v_5\}$, which forms a pentagon. To color a pentagon, three colors are needed. Because the vertex v_4 sees all five vertices of the mentioned pentagon, one necessarily needs to introduce a fourth color for the vertex v_4.

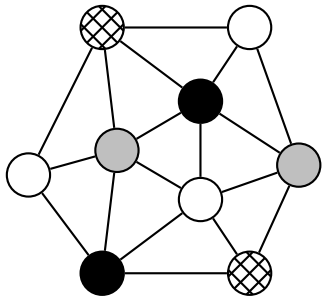

FIGURE 2 A coloring of the graph associated with the map of Figure 1 using four colors.

However, in this thesis we will not be concerned with the four color theorem. Neither will we restrict ourselves to the study of planar graphs (graphs which can be drawn in a plane without crossing edges).

The *graph coloring problem* (GCP) is a well studied combinatorial optimization problem with a wide range of applications. There are two versions of the GCP. In the decision version, one asks whether, for a given graph and a given k, a coloring with k colors exists. In the optimization version, one asks what is the smallest number k such that a k-coloring exists (i.e. a coloring using at most k colors).

The GCP is very simple to define and understand, but nevertheless, it is impossible to solve the problem exactly (in either version) as soon as the graph becomes large, because the execution time required for an exact algorithm (i.e. an algorithm which guarantees a correct answer) grows exponentially as the size of the instance grows in a linear manner.

Despite the fact that mapmakers never bothered to minimize the number of colors they used (they rather tried to balance the number of times each color appears), the GCP has practical applications. Classic examples are timetabling or frequency assignments.

Timetabling and Graph Coloring

Suppose that there is a set of courses to be scheduled and that it is already known who will teach each course to which class. There will be some courses which can be scheduled at the same time; but, for others, this may not be possible because a teacher or a class cannot be in two places at the same time. The goal is to assign a time period to each course such that the total number of different time periods used is minimal in order to obtain the shortest possible timetable.

This translates into a GCP in the following way. A vertex is associated with each course. Two vertices are joined by an edge if the two associated courses cannot be held at the same time. On the defined graph, the GCP is solved to find a coloring with a minimal number of different colors. The colors can now be interpreted as different time periods which results in a timetable using the smallest possible number of time periods.

For some examples of graph coloring formulations of timetabling problems, see [WP67, NT74, Car86, dW97].

Frequency Assignments and Graph Coloring

Suppose that a set of antennae is placed over a large area for a mobile phone network. Our task will be to assign a frequency to each antenna respecting two types of constraints. First, two antennae which are close to each other cannot transmit on the same frequency due to interference problems. Second, we would like to minimize the number of frequencies to be used.

This problem can be translated in the GCP by associating a vertex with each antenna. Two vertices are joined if the two corresponding antennae are too close to receive the same frequency. A coloring of the constructed graph can be interpreted as a valid assignment of frequencies to the antennae by simply associating a frequency with each color. A coloring using a minimal number of colors will accomplish our task.

For general formulations of frequency assignment problems, see [Gam86, GR92, Vas02, Zuf02, AvHK+03].

In the following we will define the objects and notation we will use throughout this thesis. We will start with some basic notions of graph theory and then give a formal definition of the GCP. For all terms related to graphs which are not defined here, the reader is referred to [Ber76].

Notions of Graph Theory

A graph is an object composed of *vertices* (sometimes also called *nodes*), and some of these vertices are joined by *edges*. See Figure 3 for an example of a graph.

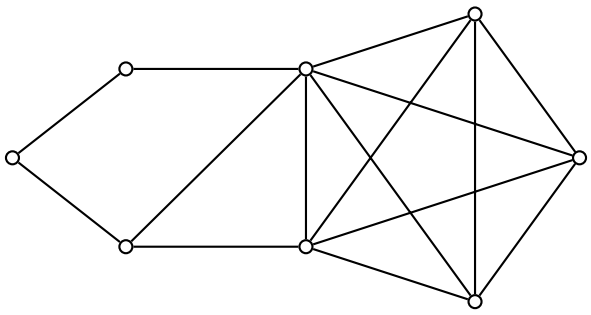

FIGURE 3 A graph with eight vertices and 15 edges.

Definition 1

A *simple (non-oriented) graph* $G = (V, E)$ is defined by its *vertex set*

$$V = \{v_1, \ldots, v_n\}$$

and its *edge set*

$$E = \big\{\{v_i, v_j\} \mid v_i \text{ and } v_j \text{ are joined}\big\}.$$

If two vertices v and w are joined by an edge, we say that v and w are *neighbors* or

are *adjacent*. The set of vertices adjacent to v is called the *neighbors of* v. It is usually denoted by $N_G(v)$ or $N(v)$ if the graph is implied.
The *complement* $\overline{G}$ of a graph G has the same vertex set as G and two vertices are joined by an edge in $\overline{G}$ if and only if they are not joined by an edge in G.
Having defined the notion of a graph, we can now introduce a vertex coloring of a graph.

Definition 2

A *vertex coloring*, or simply a *coloring* of a graph $G = (V, E)$ is a function

$$c : V \to \mathcal{C}$$

from the set of vertices V into a set of colors $\mathcal{C}$ (typically a set of natural numbers) such that

$$c(v_i) \neq c(v_j) \quad \text{whenever } \{v_i, v_j\} \in E$$

A *k-coloring* is a coloring which uses at most k different colors.

If not stated otherwise, a k-coloring will be represented as a function

$$c : V \to \{1, \ldots, k\}.$$

Colorings can also be represented as a partition of the vertex set into *color classes*, which are *stable sets*.

Definition 3

Given a graph G, a *stable set* is a set of mutually non-adjacent vertices.

The size of the largest stable set in a graph G is called its *stability number* and is denoted by $\alpha(G)$ or α if the graph is implied.

Definition 4

Given a graph G, a *clique* is a set of mutually adjacent vertices.

The size of the largest clique in graph G is denoted by $\omega(G)$ or ω if the graph is implied. Note that the complement of a stable set is a clique and vice versa.
Given a k-coloring c as an integer valued function $c : V \to \{1, \ldots, k\}$, the equivalent partitioning into color classes $\mathcal{C}_1, \ldots, \mathcal{C}_k$ can be defined as follows:

$$\mathcal{C}_i = \{v \in V \mid c(v) = i\} \quad \text{for } i = 1, \ldots, k.$$

The optimization version of the GCP concerns itself with minimizing the number of colors used to color a given graph. This number χ is an important invariant of a graph.

Definition 5

The *chromatic number* χ of a graph G, also denoted by $\chi(G)$, is defined to be the smallest k such that a k-coloring of G exists.

For the purpose of graph coloring, we can restrict ourselves to *connected* graphs. In order to define *connectedness* we first define a *chain*.

Definition 6

A *chain* C in a graph $G = (V, E)$ is a sequence of edges $C = (e_1, \ldots, e_l)$ such that $|e_i \cap e_{i+1}| = 1$ for all $i = 1, \ldots l-1$ and $e_i \cap e_{i+1} \neq e_{i+1} \cap e_{i+2}$ for all $i = 1, \ldots, l-2$.

With the notion of a chain, we can now define *connectedness*.

Definition 7

A graph $G = (V, E)$ is *connected* if, for every pair of vertices v_i, v_j, there exists a chain $C = (e_1, \ldots, e_l)$ such that $v_i \in e_1$ and $v_j \in e_l$.

If the GCP has to be solved on a non-connected graph G, one can solve the GCP independently for each connected component of G. Hence, all graphs are assumed to be connected unless specified otherwise.

Proofs in this thesis will be terminated with the two letters *h.x.* [Kel66].

Graph Coloring in Theory

There are several research directions in graph theory dealing with coloring problems as, for instance, studies about the complexity of the graph coloring problem restricted to special classes of graphs. Our studies focus on properties of neighborhoods in graph colorings.

Suppose there is a graph G with an optimal coloring c (i.e. a coloring using χ different colors). Then there is a vertex v of G such that every color is present on v and its neighbors. We say that v *sees* χ different colors. It is easy to realize why such a vertex v must exist. (If it were not the case, consider all vertices colored with the largest color χ. For every such vertex w, there is at least one color i which does not appear on w and its neighbors. Therefore, one can change the color of w and give it color i (which is necessarily smaller than χ). Because the vertices of a color class are all mutually non-adjacent, the recoloring of those vertices can be done independently. One ends up with a coloring using fewer than χ different colors which contradicts the definition of χ.)

However, it turns out that there are graphs with colorings using more than χ colors such that every vertex sees fewer than χ colors. We investigate graphs with this property; these are called *oligomatic graphs*. We also consider larger neighborhoods and ask how large the neighborhood must be in order to guarantee that χ different colors appear around at least one vertex. Our research concludes that, for a $(\chi + p)$-coloring, the distance $\lceil \frac{p}{2} \rceil + 1$ is large enough. The technique for this proof is original and more involved than the procedure sketched above. Instead of iteratively recoloring the vertices of each color

class independently (to reach the contradiction that the graph is $(\chi - 1)$-colorable), here, $p + 1$ color classes are simultaneously recolored.

Graph Coloring in Practice

The decision version of the GCP is an $\mathcal{NP}$-complete problem [GJ79], which means that, unless $\mathcal{P}$=$\mathcal{NP}$, no polynomial algorithm exists to solve the GCP. In practice that means that, for an exact algorithm, the execution time required grows very fast with the size of the instance. The limit of today's best exact algorithms [MT96, HH02b, Sch04] lies somewhere between 80 and 150 vertices, depending on the nature of the instance. Because solving large instances of the GCP is not possible (the universe will probably cease to exist before an exact algorithm has terminated) *heuristics* must be developed. A heuristic is an algorithm which uses some strategy to produce a "good" solution in a "short" time. "Good" means that the solution should, first of all, satisfy the user of the heuristic. He might compare the heuristic's results to a known optimum (which is rare), to existing real-world solutions, or to solutions obtained with other methods, including other heuristics. "Short" means that the user of the heuristic accepts to wait for the heuristic to terminate (typically between a few seconds and several hours). Generally, heuristics do not provide a guarantee on the quality of the solution they produce.

The most straightforward heuristic is called *greedy algorithm*. This type of algorithm constructs a solution step by step. At every step, the "best" way (according to some *objective function*) of completing the current partial solution is applied. A possible greedy algorithm for the GCP consists of visiting each vertex in some order and assigning the smallest possible color to each vertex. Such an algorithm is very fast but results in solutions which are generally of a poor quality compared to other, more sophisticated heuristics.

We will investigate different variations of *tabu search*, an important and very successful type of heuristic. Our main contribution consists of different simple mechanisms to adjust a crucial parameter for the *tabu search heuristic* in an automated manner. We will apply the techniques developed to the GCP and to a weighted version of the GCP. The results obtained are very promising, and our method even finds an optimal coloring for a well-known benchmark graph (called flat300_28_0) for which no other known method (at the time of publication) is able to find an optimal coloring.

The GCP is often too simple a model to be applied directly to real, practical problems, that involve more specific objectives or constraints. In the final chapter, we will introduce the *weighted graph coloring problem* (WCP), which has applications in batch scheduling and telecommunications. We develop several adaptations of tabu search for the WCP, as well as an *adaptive memory* heuristic, which has proven to be very efficient. To test the heuristics, we present several ways of generating instances for the WCP.

We will conclude with some remarks on the procedures developed and on the possibility of applying the basic concepts to other combinatorial optimization problems.

Chapter 1

Oligomatic Colorings

1.1 Introduction

In the context of graph coloring, one is generally interested either in colorings that use the least possible number of colors or in simply determining this number, called the *chromatic number* of the graph, denoted by $\chi(G)$ or simply χ, if the graph is implied by the context. We call a coloring using exactly χ different colors an *optimal coloring*. A coloring using more than χ different colors will be called a *suboptimal coloring*.

Properties of optimal vertex colorings of graphs have been widely studied [Bro41]. On the other hand, structural properties of suboptimal colorings are mostly unknown. However, those properties do merit attention, as the graph coloring problem is used as a basis for numerous applications. Often, coloring methods do not provide optimal, but suboptimal colorings.

In what follows, graphs are always simple (no double edges), loop-free (no edge linking a vertex to itself), connected (there is a chain between any pair of vertices) and non-trivial (at least two vertices), unless stated otherwise.

Let us start with a very simple property of optimal colorings:

Property 8

Let $G = (V, E)$ be a graph and c an optimal coloring of G. Then there exists a vertex v such that for every $i \neq c(v)$ there is a neighbor w of v such that $c(w) = i$.

Proof

By contradiction, suppose that c is an optimal coloring for which the property does not hold, i.e. for every vertex v there exists a color $\mu(v) \neq c(v)$ such that there is no neighbor of v with color $\mu(v)$.

We can now recolor every vertex $w \in \mathcal{C}_\chi$ (vertices colored with the color χ) with the color $\mu(w)$ to obtain a coloring c' using $\chi - 1$ colors. However, this contradicts the optimality of c. h.x.

Note that this property holds, in fact, for at least one vertex of every color. For what follows, we will need to introduce some notation.

Definition 9

Let $G = (V, E)$ be a graph and $v, w \in V$ two vertices of G. We define the *distance* $d(v, w)$ between v and w to be the number of edges on a shortest chain linking v and w.

Now that we have defined a distance, we can define *closed neighborhoods*:

Definition 10

Let $G = (V, E)$ be a graph and $v \in V$ a vertex of G. Define the *closed neighborhood of radius r of v* as the set of all vertices at a distance r or less from v.

$$N_r[v] = \{w \in V \mid d(v, w) \leq r\}$$

We will be interested in the number of different colors occurring in the neighborhood of a vertex.

Definition 11

Let $G = (V, E)$ be a graph and c be a k-coloring of G. We define the *colors seen by v at a distance at most r* as

$$\mathcal{K}_r(v) = \{c(w) \mid w \in N_r[v]\}$$

We define *the colors seen by v* as $\mathcal{K}_1(v)$.

With the above definition, we can reformulate Property 8 as follows: For an optimal coloring, there exists a vertex v such that $|\mathcal{K}_1(v)| = \chi$.
Can this statement be generalized to suboptimal colorings? In other words, for a suboptimal coloring, is there always a vertex such that $|\mathcal{K}_1(v)| \geq \chi$?
In general, the answer is no. Colorings for which there is no vertex that sees χ different colors, will be called *oligomatic*.

Definition 12

A coloring c of a graph $G = (V, E)$ is called *oligomatic*, if $|\mathcal{K}_1(v)| < \chi(G)\ \forall v \in V$.

The term *oligomatic* comes from the Greek root *oligo*, which means few, and the word *chromatic*. This is due to the fact that every vertex sees only few colors.

It is clear that there are graphs for which oligomatic colorings do not exist, such as perfect graphs, because the size of a maximum clique ω equals the chromatic number χ. Clearly, a vertex in a maximum clique sees at least ω different colors for any coloring of the graph.

This leads us to the definition of *oligomatic graphs*:

Definition 13

A graph G is called *oligomatic* if there exists an oligomatic coloring for G.

An oligomatic coloring is characterized by two parameters: The number k of colors for the coloring, and λ, the maximal number of colors seen by a vertex.

Definition 14

A (k, λ)-*coloring* of a graph G is a k-coloring such that every vertex sees at most λ different colors; or, formally, $|\mathcal{K}_1(v)| \leq \lambda$ for all vertices $v \in V$.

With this definition, we can characterize oligomatic graphs in a concise and elegant way: A graph G is oligomatic if there exists a (k, λ)-coloring of G with $\lambda < \chi$.

Due to the existence of oligomatic graphs (shown later), the Property 8 cannot be generalized to arbitrary colorings. However, for 3-colorable graphs the generalization holds.

Property 15

A graph G with $\chi \leq 3$ is not oligomatic.

Or, in other words, for any graph G with $\chi \leq 3$ and any coloring there exist a vertex which sees at least χ colors.

Proof

By contradiction, suppose G has an oligomatic coloring.

Case 1 $\boxed{\chi = 2}$

Let c be an oligomatic coloring of G. With this coloring, every vertex sees at most one color. This implies that G contains only isolated vertices, and this contradicts the fact that $\chi = 2$.

Case 2 $\boxed{\chi = 3}$

An oligomatic coloring c has the property that every vertex sees at most two colors: its own and one other color of its neighbors. This implies that G is bipartite (and, therefore, 2-colorable), which is a contradiction.

h.x.

1.2 Literature Overview

A remarkable article with the title "Coloring graphs with locally few colors" has been published in 1986 by Erdős et al.[EFH+86].

Their research investigates so-called *local r-colorings*, which are, in our notation, (k, λ)-colorings with $\lambda = r$ and $k \geq \lambda$ arbitrary.

They also provide an example of a class of oligomatic graphs, so-called *shift graphs*, which were introduced in [EH68].

A shift graph $S(n, k)$ is a graph whose vertex set consists of all ordered subsets of size k of the set $\{1, \ldots, n\}$ and whose edge set consists of all pairs of the form $\{\{x_1, x_2, \ldots, x_k\}, \{x_2, x_3, \ldots, x_{k+1}\}\}$ where $\{x_1, x_2, \ldots, x_{k+1}\}$ is an ordered subset of $\{1, \ldots, n\}$.

The shift graph $S(n, 1)$ is a complete graph on n vertices. One can show that $\chi(S(n, 2)) = \lceil \lg n \rceil$.

For n large enough, the graphs $S(n, 3)$ are oligomatic. We define a $(n-2, 3)$-coloring c by setting $c(\{x_1, x_2, x_3\}) = x_2$. Clearly $\mathcal{K}_1(\{x_1, x_2, x_3\}) \subseteq \{x_1, x_2, x_3\}$. One can show that $\chi(S(n, 3))$ gets arbitrarily large when n grows:

Property 16 Theorem 2.2 in [FHRT91]

For each integer $n \geq 4$ the chromatic number of the shift graph $S(n, 3)$ is the least t for which there are at least n anti-chains in the lattice of all subsets of $\{1, \ldots, t\}$.

An anti-chain (also called a Sperner system) is a family of subsets such that, for any two of them, neither is a subset of the other. Determining the number of anti-chains on the set $\{1, \ldots, t\}$ (including the empty anti-chain) is known as Dedekind's problem, and the numbers in the associated sequence are sometimes called Dedekind numbers. The following sequence has been obtained from the *On-Line Encyclopedia of Integer Sequences* [Slo].

$$\begin{aligned}
A(0) &= 2 \\
A(1) &= 3 \\
A(2) &= 6 \\
A(3) &= 20 \\
A(4) &= 168 \\
A(5) &= 7'581 \\
A(6) &= 7'828'354 \\
A(7) &= 2'414'682'040'998 \\
A(8) &= 56'130'437'228'687'557'907'788
\end{aligned}$$

With these numbers we can compute the chromatic number of some $S(n, 3)$ shift graphs. For example $\chi(S(7581, 3) = 5$ (since $A(5) \geq 7581$) and $\chi(S(7582, 3) = 6$ (since $A(6) \geq 7582$ but not $A(5)$).

Therefore, for $n \geq 21$, the shift graphs $S(n, 3)$ are oligomatic. Note that the graph $S(21, 3)$ already has $\binom{21}{3} = 1330$ vertices and $\binom{21}{4} = 5985$ edges.

As a corollary in [FHRT91] there is the following estimation:

$$\chi(S(n,3)) = \lg\lg n + \left(\frac{1}{2}o(1)\right)\lg\lg\lg n.$$

The next definition arises from the following question: "Given a graph G, what is the smallest λ such that a (k,λ)-coloring exists?"

Definition 17 [KPS04]

The *local chromatic number* ψ of a graph G is the maximum number of different colors appearing in $N[v]$ of any vertex v, minimized over all colorings of G. More formally

$$\psi(G) = \min_c \left(\max_{v\in V} |\mathcal{K}_1(v)|\right)$$

where c runs over all possible colorings of G and $\mathcal{K}_1(v)$ is the set of colors seen by v with respect to c.

An optimal coloring uses χ colors. It follows that $\chi \geq \psi$.
Clearly, a graph G is oligomatic if and only if

$$\chi(G) > \psi(G).$$

There is a link between the local chromatic number and the *fractional chromatic number*.

Definition 18

The *fractional chromatic number* $\chi^\star$ of a graph G is

$$\chi^\star(G) = \min_w \sum_{A\in S(G)} w(A)$$

where $S(G)$ is the family of independent sets of G and the minimization is over all nonnegative weightings $w : S(G) \to \mathbb{R}$ satisfying $\sum_{A\ni v} w(A) \geq 1$ for every $v \in V$. Clearly, $\chi^\star(G) \leq \chi(G)$.

Property 19 Theorem 3 in [KPS04]

$$\psi(G) \geq \chi^\star(G)$$

Therefore, the fractional chromatic number $\chi^\star$ is a lower bound for the local chromatic number ψ. The proof of the Property 19 requires more background and is given in Section 1.4.4.

1.3 An Oligomatic Graph: penta$K_{3,3}$

By Property 15, an oligomatic graph is at least 4-chromatic. Therefore, an oligomatic coloring uses at least 5 colors. Based on these observations, the graph penta$K_{3,3}$ has been constructed at the very beginning of our research as an attempt to prove that Property 8 can be generalized to suboptimal colorings.

1.3.1 Construction of penta$K_{3,3}$

We have constructed penta$K_{3,3}$ while we were trying to prove that a $(5,3)$-colorable graph is 3-chromatic. In a $(5,3)$-colorable graph, every vertex has neighbors of only two other colors. We classify the vertices by their color i and the colors $\{i,j\}$ of their neighbors. For a $(5,3)$-coloring, we obtain 30 different classes of the form $C_{i,\{j,k\}}$ with $i,j,k \in \{1,2,3,4,5\}$ all distinct.

Vertices in $C_{i,\{j,k\}}$ can only be adjacent to vertices in $C_{i',\{j',k'\}}$ if

$$i \in \{j',k'\}$$

and

$$i' \in \{j,k\}.$$

The vertex set of penta$K_{3,3}$ consists of the vertices $v_{i,\{j,k\}}$, one for each class $C_{i,\{j,k\}}$. Two vertices are joined if they verify the above conditions.

This results in a graph with 30 vertices and 90 edges. The name penta$K_{3,3}$ comes from the facts that there are five colors (penta) and that there is a complete bipartite graph $K_{3,3}$ between each pair of colors. A representation of penta$K_{3,3}$ can be found in Figure 1.1 (at the end of this chapter).

1.3.2 Properties of penta$K_{3,3}$

The graph penta$K_{3,3}$ has chromatic number 4 and admits an oligomatic $(5,3)$-coloring. A direct proof for $\chi(\text{penta}K_{3,3}) = 4$ can be found in our paper [BdW04]. We will present an alternative proof of this fact later based on a more powerful theorem.

Any permutation σ of the 5 colors of penta$K_{3,3}$ induces an automorphism. This way, for any pair of vertices u, v we can find an automorphism that maps u onto v. As a consequence, penta$K_{3,3}$ is vertex transitive.

Moreover, penta$K_{3,3}$ is vertex-critical (i.e. the removal of any vertex leaves a 3-colorable graph), as can be seen from Figure 1.1 (at the end of this chapter).

In terms of the number of vertices, penta$K_{3,3}$ is also the smallest 4-chromatic graph with an oligomatic 5-coloring:

Theorem 20

The smallest 4-chromatic graph which has an oligomatic 5-coloring contains 30 vertices.

Proof

Let G be a 4-chromatic graph and c an oligomatic 5-coloring of G. Suppose G has at most 29 vertices. We will show that G is 3-colorable.

Define 30 classes $C_{i,\{j,k\}}$ to classify the vertices of G according to their color i and the colors j, k of their neighbors. Note that $C_{i,\{j,k\}} = C_{i,\{k,j\}}$. If a vertex has only one color j in its neighborhood, we place that vertex in the class $C_{i,\{j,k\}}$ with the smallest k. For a fixed color i we have $\binom{4}{2} = 6$ classes and for five colors, we have a total of 30 classes.

For penta$K_{3,3}$ and the 5-coloring defined by $c(v_{i,\{j,k\}}) = i$, we have a bijection between vertices and classes. Because G has fewer than 30 vertices, at least one class $C^\star$ will be empty. However, penta$K_{3,3}$ is vertex-critical. Let H be the subgraph of penta$K_{3,3}$ obtained after removal of the vertex corresponding to $C^\star$. H has a 3-coloring c'. We can now color the vertices of G by giving them the same color as the vertex in H corresponding to their class.

Let us show that this produces a coloring in G. All edges in G are between vertices in classes of type $C_{i,\{j,k\}}$ and $C_{j,\{i,l\}}$. However, in penta$K_{3,3}$, the corresponding vertices are always joined by an edge; and, therefore, since c' is a coloring of H, we have a 3-coloring of G, which gives us the desired contradiction. h.x.

1.4 Generalizations of penta$K_{3,3}$

The construction of penta$K_{3,3}$ can be generalized for more than 5 colors and for larger neighborhoods than N_1. These graphs have very interesting properties. In [EFH$^+$86] they are called *universal graphs*. In this paper, we will use a slightly more restrictive definition which will result in smaller graphs.

The name *universal graphs* comes from the fact that any (k, λ)-colorable graph can be mapped by a homomorphism onto some *universal graph*. Therefore, in order to study oligomatic graphs, it will be sufficient for most purposes to study the properties of these *universal graphs*.

1.4.1 Universal Graphs

Universal graphs are constructed such that they have a natural (k, λ)-coloring.

Definition 21

The *universal graph* $U(k, \lambda)$ is defined by its vertex set

$$V = \{(c, \mathcal{C}) \mid c \in \{1, \ldots, k\}, \mathcal{C} \subseteq \{1, \ldots, k\}, |\mathcal{C}| = \lambda - 1, c \notin \mathcal{C}\},$$

and its edge set

$$E = \{\{(c_1, \mathcal{C}_1), (c_2, \mathcal{C}_2)\} \mid c_1 \in \mathcal{C}_2, c_2 \in \mathcal{C}_1\}.$$

A vertex $v = (c, \mathcal{C})$ is sometimes denoted as $v_{c,\mathcal{C}}$.

By construction, the graphs $U(k, \lambda)$ have a k-coloring such that every vertex sees λ different colors. This can be demonstrated by simply coloring every vertex $v = (c, \mathcal{C})$ with color c. This coloring c is called the *natural coloring* of $U(k, \lambda)$.

In [EFH+86], the vertex set is defined with $|\mathcal{C}| \leq \lambda - 1$ instead of $|\mathcal{C}| = \lambda - 1$. The use of this inequality results in larger graphs, which is not necessary (as we will see later).

Note that penta$K_{3,3} = U(5, 3)$.

1.4.2 $U(k, \lambda)$ is Vertex-Transitive

The symmetry of universal graphs is a very useful property. All vertices "are the same," in the sense that, for any pair of vertices u, v, there exists an automorphism mapping u to v. An automorphism of $U(k, \lambda) = (V, E)$ can easily be constructed in the following manner:

$$h_\sigma : V \to V$$
$$h_\sigma((c, \mathcal{C})) = (\sigma(c), \sigma(\mathcal{C}))$$

where σ is a permutation of the set $\{1, \ldots, k\}$, and $\sigma(\mathcal{C}) = \{\sigma(l) \mid l \in \mathcal{C}\}$. From the fact that σ is a permutation, it follows that h_σ is bijective, and it is straight forward to see that h_σ maps edges onto edges.

To construct h_σ such that u is mapped onto v, suppose that $u = (c_u, \mathcal{C}_u)$ and $v = (c_v, \mathcal{C}_v)$. Now consider a permutation σ of the set $\{1, \ldots, k\}$ such that $\sigma(c_u) = c_v$ and $\sigma(\mathcal{C}_u) = \mathcal{C}_v$. Clearly, h_σ maps u to v.

1.4.3 Size of $U(k, \lambda)$

The number of vertices of $U(k, \lambda) = (V, E)$ is

$$|V| = k\binom{k-1}{\lambda-1} = \frac{k!}{(\lambda-1)!(k-\lambda)!}. \tag{1.1}$$

This follows directly from the definition of the vertices $v = (c, \mathcal{C})$ of $U(k, \lambda)$. We have to choose one color c from k colors, and then choose $\lambda - 1$ colors among $k - 1$ for $\mathcal{C}$.

The graph is regular with degree

$$\Delta = (\lambda - 1)\binom{k-2}{\lambda-2} = \frac{(k-2)!}{(\lambda-2)!(k-\lambda)!}. \tag{1.2}$$

Every vertex $v = (c_v, \mathcal{C}_v)$ has $\lambda - 1$ colors among its neighbors, and every neighbor $u = (c_u, \mathcal{C}_u)$ must have c_v in its set $\mathcal{C}_u$. To complete the set $\mathcal{C}_u$, we choose $\lambda - 2$ colors among $k - 2$ colors.

The size of the edge set will be

$$|E| = \frac{|V|}{2}\Delta = \frac{k(\lambda-1)}{2}\binom{k-1}{\lambda-1}\binom{k-2}{\lambda-2}. \tag{1.3}$$

The importance of the graphs $U(k, \lambda)$ comes from the fact that any graph G with a (k, λ)-coloring can be mapped by a homomorphism h onto $U(k, \lambda)$ such that every edge of G is mapped onto an edge of $U(k, \lambda)$. As a consequence, we have $\chi(G) \leq \chi(U(k, \lambda))$ because any coloring c of $U(k, \lambda)$ induces a coloring c' of G by setting $c'(v) = c(h(v))$.

We now have the ingredients to give a proof of Property 19.

1.4.4 The Fractional Chromatic Number of $U(k, \lambda)$

For a definition of the *fractional chromatic number* $\chi^\star$, refer to the Definition 18.

Property 22 Lemma 2 in [KPS04]

For all $k \geq \lambda \geq 2$ we have

$$\chi^\star(U(k, \lambda)) = \lambda.$$

Proof

In this proof we use the fact that if a graph $G = (V, E)$ is vertex-transitive, then

$$\chi^\star(G) = \frac{|V|}{\alpha(G)}.$$

For a proof of this fact and for further information about the fractional chromatic number we refer to the books [SU97, GR01].

It is easy to check that $\chi^\star(U(k, \lambda)) \geq \omega(U(k, \lambda)) = \lambda$ thus we only have to prove that λ is an upper bound. Consider the vertices $v_{c,\mathcal{C}}$ for which $c < c_i$ for all $c_i \in \mathcal{C}$. These form an independent set S. Thinking about the vertices $v_{c,\mathcal{C}}$ as λ-tuples with one distinguished element and the elements of S as those λ-tuples whose distinguished element is the smallest one, we immediately get

$$\chi^\star(U(k, \lambda)) = \frac{|V(U(k, \lambda))|}{\alpha(U(k, \lambda))} \leq \frac{|V(U(k, \lambda))|}{|S|} = \lambda$$

proving the statement. h.x.

Using Property 22 we can easily prove Property 19.

Proof of Property 19

Let G be a graph with $\psi(G) = \lambda$. This means that there is a homomorphism from G to $U(k, \lambda)$ for some k. Since a homomorphism cannot decrease the fractional chromatic number, from Property 22 we obtain

$$\chi^\star(G) \leq \chi^\star(U(k, \lambda)) = \lambda = \psi(G).$$

h.x.

To study oligomatic graphs and to determine whether a graph is oligomatic or not, it is crucial to know the chromatic number. For that reason, we introduce the following notation, which links the chromatic number of a graph G with the existence of a (k, λ)-coloring of G:

Definition 23

$P(k, \gamma, \lambda)$ abbreviates the following statement: there exists a graph G with $\chi(G) > \gamma$ and a (k, λ)-coloring of G.

Property 24

It is immediately apparent that, if $P(k, \gamma, \lambda)$ holds, then $P(k', \gamma', \lambda')$ holds as well for $k' \geq k$, $\gamma' \leq \gamma$ and $\lambda' \geq \lambda$.

Proof

Let G be a graph which proves that $P(k, \gamma, \lambda)$ is true, i.e. there exists a (k, λ)-coloring c for G and $\chi(G) > \gamma$. Now, c can also be considered as a (k', λ')-coloring of G with $k' \geq k$ and $\lambda' \geq \lambda$ (even though some colors may not be used). The fact that $\chi(G) > \gamma \geq \gamma'$ proves that $P(k', \gamma', \lambda')$ holds. h.x.

We give another immediate result to get a feel for $P(k, \gamma, \lambda)$:

Property 25

For $\lambda \leq \gamma$, the statement $P(\gamma + 1, \gamma, \lambda)$ is false.

Proof

A $(\gamma + 1, \lambda)$-coloring can be transformed into a γ-coloring by simply assigning to every vertex of color $\gamma + 1$ one of the colors that this same vertex does not see. This is possible because every vertex sees at most $\lambda < \gamma + 1$ colors.
Therefore, $\chi \not> \gamma$ and as a consequence $P(\gamma + 1, \gamma, \lambda)$ is false. h.x.

The next property is given as Lemma 1.1 in [EFH+86] for their definition of $U(k,\lambda)$. The property still holds for our more restrictive definition of $U(k,\lambda)$. The proof is similar.

Property 26

$P(k,\gamma,\lambda)$ holds if and only if $\chi(U(k,\lambda)) > \gamma$.

Proof

Clearly, $c((c_v,\mathcal{C}_v)) = c_v$ is a (k,λ)-coloring, so one direction is clear.
Suppose that $\chi(U(k,\lambda)) \leq \gamma$, and let c be a γ-coloring of $U(k,\lambda)$. Let $G' = (V',E')$ be an arbitrary graph with a (k,λ)-coloring c'. We need to show that $\chi(G') \leq \gamma$. We will construct a graph homomorphism h from G' to $U(k,\lambda)$ by defining $h(v') = v_{c'(v'),\mathcal{X}}$ where $\mathcal{X}$ is equal to the set $\mathcal{K}_1(v')$ augmented by the $\lambda - |\mathcal{K}_1(v')|$ smallest elements of $\{1,\ldots,k\} \setminus \mathcal{K}_1(v')$. It is necessary to complete the set $\mathcal{X}$ such that is has exactly λ elements to fit our definition of $U(k,\lambda)$.
Clearly, $(v',w') \in E'$ implies $(h(v'),h(w')) \in E_{U(k,\lambda)}$. The coloring c'' of G defined by $c''(v') = c(h(v'))$ uses γ colors, which shows that $\chi(G') \leq \gamma$. h.x.

Theorem 27

For every γ, p, $P(\gamma+p+1,\gamma,\lambda)$ holds with $\lambda = \min\limits_{q=1,\ldots,p} (\lfloor \gamma/(q+1) \rfloor + q + 1)$.

The above theorem strengthens Theorem 2.6 in [EFH+86], where the result is shown for $\lambda = \lfloor \gamma/(p+1) \rfloor + p + 1$.
In order to prove Theorem 27, we first need a definition and another property from [EFH+86].

Definition 28

The system $\{A_{\alpha,\beta} : 1 \leq \alpha < \beta \leq k\} \subseteq \mathcal{P}(\{1,\ldots,\gamma\})$ (where $\mathcal{P}(X)$ is the set of all subsets of the set X) is (k,γ,λ)-*independent* if and only if the following holds:
for every set $B \subset \{1,\ldots,k\}$ with $|B| = \lambda$ and every $\alpha \in B$, the set

$$\bigcap_{\substack{\delta<\alpha \\ \delta\in B}} A_{\delta,\alpha} \setminus \bigcup_{\substack{\delta>\alpha \\ \delta\in B}} A_{\alpha,\delta}$$

is non-empty.

Note that an empty intersection (for $\alpha = \min B$) equals the ground set, in our case, $\{1,\ldots,\gamma\}$.

Property 29 Lemma 1.2 in [EFH$^+$86]

$P(k, \gamma, \lambda)$ holds if and only if (k, γ, λ)-independent systems do not exist.

Proof of Theorem 27

Suppose, on the contrary, that $\{A_{\alpha,\beta} : 1 \leq \alpha < \beta \leq \gamma + p + 1\}$ is a $(\gamma + p + 1, \gamma, \lambda)$-independent system, with $\lambda = \min_{q=1,\ldots,p} (\lfloor \gamma/(q+1) \rfloor + q + 1)$. Recall that $A_{\alpha,\beta}$ are subsets of the set $\{1, \ldots, \gamma\}$. Define

$$\Pi_\epsilon = \bigcap_{\phi<\epsilon} A_{\phi,\epsilon} \setminus \bigcup_{\phi>\epsilon} A_{\epsilon,\phi} \quad \text{for } \epsilon = 1, \ldots, \gamma + p + 1. \tag{1.4}$$

For two values ρ, τ such that $1 \leq \rho < \tau \leq \gamma + p + 1$, we have $\Pi_\rho \cap A_{\rho,\tau} = \emptyset$ (because $A_{\rho,\tau}$ is in the union to be taken away of (1.4)) and $\Pi_\tau \subseteq A_{\rho,\tau}$ (because $A_{\rho,\tau}$ is in the intersection of (1.4)). Therefore, $\Pi_\rho \cap \Pi_\tau = \emptyset$.
Since there are $\gamma + p + 1$ disjoint subsets Π_ϵ of $\{1, \ldots, \gamma\}$, there is a set E with $|E| \geq p + 1$ such that $\Pi_\epsilon = \emptyset$ for every $\epsilon \in E$.
Let $q^\star = \arg \min_{q=2,\ldots,p+1} (\lfloor \gamma/q \rfloor + q)$. Since q runs over $2, \ldots, p + 1$ we have trivially that $q^\star \leq p + 1$. Let X be a subset of E such that $|X| = q^\star$. Now put

$$\Pi^X_\epsilon = \bigcap_{\substack{\phi<\epsilon \\ \phi\in X}} A_{\phi,\epsilon} \setminus \bigcup_{\substack{\phi>\epsilon \\ \phi\in X}} A_{\epsilon,\phi} \quad \text{for } \epsilon \in X$$

Again, for the same argument, the subsets Π^X_ϵ are disjoint. Therefore, there exists an $x \in X$ with $|\Pi^X_x| \leq \lfloor \gamma/q^\star \rfloor$.
Since $x \in X$, we have $\Pi_x = \emptyset$ (by definition of $X \subseteq E$).
Now, for every $a \in \Pi^X_x$ there exists a $\nu(a) \in \{1, \ldots, \gamma + p + 1\}$ such that we have either

$$\nu(a) < a \text{ and } a \notin A_{\nu(a),x}$$

or

$$\nu(a) > a \text{ and } a \in A_{x,\nu(a)}.$$

To see this, recall that $\Pi_x = \emptyset$. This means that either one of the following must hold:

1. a is not in the intersection of (1.4) or
2. a is in the intersection of (1.4) and, therefore, a is also in the union of (1.4).

Let $Y = X \cup \{\nu(a) : a \in \Pi^X_x\}$. Clearly, $|Y| \leq q^\star + \lfloor \gamma/q^\star \rfloor = \lambda$. Now consider the set

$$\Pi^Y_x = \bigcap_{\substack{\phi<x \\ \phi\in Y}} A_{\phi,x} \setminus \bigcup_{\substack{\phi>x \\ \phi\in Y}} A_{x,\phi} \quad \subseteq \quad \Pi^X_x.$$

For every $a \in \Pi^X_x$, by choice of $\nu(a)$, $a \notin \Pi^Y_x$. Therefore, $\Pi^Y_x = \emptyset$. We can now complete the set Y arbitrarily to obtain a set $Y^\star$ such that $|Y^\star| = \lambda$ to obtain $\Pi^{Y^\star}_x \subseteq \Pi^Y_x = \emptyset$,

which shows that our system is not $(\gamma+p+1, \gamma, \lambda)$-independent, and we reach the desired contradiction. h.x.

The Theorem 27 above settles a conjecture in [BdW04]:

Property 30

For each integer ν, there exists a graph G with $\chi = \nu$ admitting a (k, λ)-coloring with $k = \chi + 1$ and $\lambda = \left\lceil \frac{\chi}{2} \right\rceil + 1$.

Setting $p = 1$ and $\gamma = \chi - 1$ in Theorem 27 gives us that

$$P(\chi + 1, \chi - 1, \left\lceil \frac{\chi}{2} \right\rceil + 1) \text{ is true.}$$

Further, Theorem 2.7 in [EFH$^+$86] gives a negative result, which is tight where applicable:

Property 31

For every $\gamma \geq p(p+1)$,

$$P\left(\gamma + p + 1,\ \gamma,\ \left\lfloor \frac{\gamma}{p+1} \right\rfloor + p\right) \text{ is false.}$$

This result is tight in the sense that both $P(\gamma + p + 1, \gamma, \lfloor \gamma/(p+1) \rfloor + p)$ is false and, by Theorem 27, $P(\gamma + p + 1, \gamma, \lfloor \gamma/(p+1) \rfloor + p + 1)$ is true.
For the case where $\gamma < p(p+1)$, we formulate the following theorem, which is equivalent to Theorem 2 in [BdW04]:

Theorem 32

For a graph G with $\chi \geq 2$ and $p \geq 1$ and any $(\chi + p)$-coloring, we have

$$|\mathcal{K}_1(v)| \geq \left\lceil \frac{\chi}{p+1} \right\rceil + 1 \quad \forall v \in V.$$

In other words, every vertex sees at least $\left\lceil \frac{\chi}{p+1} \right\rceil + 1$ different colors.

Before proving the above theorem, let us reformulate it:
There is no graph G with admitting a (k, λ)-coloring with $k = \chi + p$ and $\lambda = \left\lceil \frac{\chi}{p+1} \right\rceil$. In the $P(k, \gamma, \lambda)$ notation, this statement becomes

$$P\left(\chi + p,\ \chi - 1,\ \left\lceil \frac{\chi}{p+1} \right\rceil\right) \text{ is false for } \chi \geq 2,\ p \geq 1$$

or, equivalently, by setting $\chi = \gamma + 1$,

$$P\left(\gamma + p + 1,\ \gamma,\ \left\lfloor \frac{\gamma}{p+1} \right\rfloor + 1\right) \text{ is false for } \gamma,\ p \geq 1.$$

Proof of Theorem 32

Assume the contrary: there exists a graph G and a $(\chi+p)$-coloring c, such that, for every vertex v we have $|\mathcal{K}_1(v)| \leq \left\lceil \frac{\chi}{p+1} \right\rceil = \left\lfloor \frac{\chi-1}{p+1} \right\rfloor + 1$. For graphs G with $\chi \leq 2(p+1)$, the restriction on $\mathcal{K}_1(v)$ implies that the graph G is bipartite and, therefore, the proposition is true, because it is either trivial (for $\chi = 2$) or we find immediately the desired contradiction (for $\chi \geq 3$).
For graphs G with $\chi > 2(p+1)$, let $P_1, \ldots, P_{p+1}$ be disjoint subsets of $\mathcal{X} = \{p+2, \ldots, p+\chi\}$ with $|P_i| \geq \left\lfloor \frac{\chi-1}{p+1} \right\rfloor$ for $i = 1, \ldots, p+1$. (Note that $|\mathcal{X}| = \chi - 1$.) Let V' be the set of vertices v with $c(v) \leq p+1$ such that there is no neighbor w of v with $c(w) \leq p+1$. Then define the new coloring

$$c'(v) = \begin{cases} \max \overline{\mathcal{K}_1(v)} & \text{if } v \in V' \\ \max\left(\overline{\mathcal{K}_1(v)} \cap P_{c(v)}\right) & \text{if } v \notin V' \text{ and } c(v) \leq p+1 \\ c(v) & \text{otherwise (i.e. } c(v) \geq p+2) \end{cases}.$$

In the first and third case, we create no conflicts in the recoloring procedure. In the second case ($v \notin V'$ and $c(v) \leq p+1$), we must check that $\overline{\mathcal{K}_1(v)} \cap P_{c(v)} \neq \emptyset$. This is the case since $|\mathcal{K}_1(v) \cap \{1, \ldots, p+1\}| \geq 2$ and, therefore, $|\mathcal{K}_1(v) \cap \mathcal{X}| \leq \left\lfloor \frac{\chi-1}{p+1} \right\rfloor - 1$, which implies that $P_{c(v)} \not\subseteq \mathcal{K}_1(v)$ and, therefore, $\overline{\mathcal{K}_1(v)} \cap P_{c(v)} \neq \emptyset$.
By definition of $P_{c(v)}$, it is clear that $c'(v) \geq p+1$. All that remains is to ensure that c' is a feasible coloring. For vertices $v \in V'$, there are no conflicts because they are not adjacent to vertices that change color and because they are recolored with a color in $\overline{\mathcal{K}_1(v)}$. The remaining vertices are colored with colors chosen in $p+1$ disjoint sets of colors and, therefore, do not conflict with each other. Hence c' colors G with $\chi - 1$ colors which is the desired contradiction. h.x.

Property 31 strengthens Theorem 32 for the case $\chi \geq p(p+1)$ and can be reformulated as follows.

Property 33

For any $p \geq 0$ and any graph G with $\chi \geq p(p+1)$ and any $(\chi+p)$-coloring, we have

$$|\mathcal{K}_1(v)| \geq \left\lceil \frac{\chi}{p+1} \right\rceil + p \quad \forall v \in V.$$

In other words, every vertex sees at least $\left\lceil \frac{\chi}{p+1} \right\rceil + p$ different colors.

Proof

By contradiction, suppose there is a graph $G = (V, E)$ and a $\chi + p$ coloring such that

$$|\mathcal{K}_1(v)| \leq \left\lceil \frac{\chi}{p+1} \right\rceil + p - 1 = \left\lfloor \frac{\chi - 1}{p+1} \right\rfloor + p.$$

Expressed in the $P(k, \gamma, \lambda)$ notation this affirms that

$$P(\chi + p, \chi - 1, \left\lfloor \frac{\chi - 1}{p+1} \right\rfloor + p) \text{ is true.}$$

Substituting χ by $\gamma + 1$ results in

$$P(\gamma + p + 1, \gamma, \left\lfloor \frac{\gamma}{p+1} \right\rfloor + p) \text{ is true,}$$

which contradicts Property 31. h.x.

1.5 Chromatic Number and Criticality of $U(k, \lambda)$

Using the properties and theorems from the previous sections, we are now able to determine the chromatic numbers of some of the graphs $U(k, \lambda)$.

Unfortunately, we do not have a complete classification of all universal graphs to present. For some cases, we were compelled to use coloring heuristics to get estimates and bounds for the chromatic number. More precisely:

Definition 34

A graph G is *heuristically k-chromatic* if a given graph coloring heuristic $\mathcal{H}$ finds a k-coloring, but no $k-1$ coloring. We will denote this number by $\chi_{\mathcal{H}}(G)$, or simply $\chi_{\mathcal{H}}$ if the graph is implied from the context.

Clearly, $\chi \leq \chi_{\mathcal{H}}$. For our purposes we use the graph coloring heuristic Foo-Partialcol for $\mathcal{H}$ which will be presented in Chapter 3.

Note that $\chi_{\mathcal{H}}$ can sometimes be used to obtain exact results. Consider the following case: Suppose that, for a given graph G, we have a proof for $\chi \geq k$ and we find $\chi_{\mathcal{H}} = k$. This gives a proof that, in fact, $\chi = k$.

We are also interested in the *criticality* of these graphs. A graph G is *critical* if the removal of any vertex v reduces the chromatic number. For some graphs, we will present a formal proof, for others, we have only the result of a coloring heuristic.

Definition 35

A graph G is *heuristically critical* if, for every vertex v, we have $\chi_{\mathcal{H}}(G-v) = \chi_{\mathcal{H}}(G)-1$. $G - v$ denotes the graph G with the vertex v removed.

Note that, because universal graphs are vertex-transitive, we only need to determine $\chi_{\mathcal{H}}(G - v)$ for one single vertex v to study their criticality.

1.5.1 The Chromatic Number of penta$K_{3,3}$

Property 36

$$\chi(\text{penta}K_{3,3}) = 4$$

Proof

Recall that penta$K_{3,3}$ is the universal graph $U(5, 3)$. From Theorem 27 with $\gamma = 3$ and $p = 1$, it follows that $P(5, 3, 3)$ is true. This implies (by Property 26) that $\chi(U(5, 3)) \geq 4$. Property 25 with $\gamma = 4$ gives us that $P(5, 4, 3)$ is false, which implies that $\chi(U(5, 3)) \leq 4$. h.x.

An alternative proof is given in [BdW04], where it is shown that the maximum stable set of penta$K_{3,3}$ is of size 10. Recall that the graph has 30 vertices. We show that there is no 3-coloring of penta$K_{3,3}$, because the maximum stable sets are of a particular form, such that three stable sets of cardinality 10 necessarily intersect.

1.5.2 Chromatic Number of $U(k, 3)$

For $k \geq 5$, all graphs $U(k, 3)$ contain penta$K_{3,3}$ as a subgraph and, therefore,

$$\chi(U(k, 3)) \geq 4.$$

For $k \leq 12$, the Foo-Partialcol coloring heuristic can easily determine a 4-coloring, and, therefore, $\chi(U(k, 3)) = 4$ for $k = 5, \ldots, 12$.
For $U(13, 3)$, we have found that $\chi_{\mathcal{H}} = 5$ and that it is heuristically critical.

1.5.3 More General Results

Theorem 37

For any $k \geq 5$ and $\lfloor \frac{k}{2} \rfloor + 1 \leq \lambda < k - 1$, we have

$$\chi(U(k, \lambda)) = k - 1$$

Proof

By Property 26, we have an equivalence between $P(k, \gamma, \lambda)$ and $\chi(U(k, \lambda)) > \gamma$. For $\lambda < k - 1$ every graph $U(k, \lambda)$ has a $(k-1)$-coloring obtained from the natural (k, λ)-coloring by recoloring every vertex v of color k with a color not seen by v. Theorem 27 with $p = 1$ affirms that

$$P(\gamma + 2, \gamma, \lfloor \gamma/2 \rfloor + 2) \text{ is true,}$$

or, equivalently, (setting $k = \gamma + 2$)

$$P(k, k - 2, \lfloor (k - 2)/2 \rfloor + 2) \text{ is true,}$$

which is the same as

$$P(k, k - 2, \lfloor k/2 \rfloor + 1) \text{ is true,}$$

which proves the theorem. h.x.

Property 38

Suppose

$$P(k, \gamma - 1, \lambda) \text{ is true,}$$

which is to say that there exists a graph with $\chi \geq \gamma$ and a (k, λ)-coloring and, at the same time,

$$P(k, \gamma, \lambda) \text{ is false,}$$

which is to say that there is no graph with $\chi > \gamma$ and a (k, λ)-coloring. It then follows that

$$\chi(U(k, \lambda)) = \gamma.$$

This follows straightforward from the equivalence of $P(k, \gamma, \lambda)$ and $\chi(U(k, \lambda)) > \gamma$.

The above Property 38 can be used to determine χ of a certain number of graphs $U(k, \lambda)$. For this, we generate positive statements $P(k, \gamma, \lambda)$ using Theorem 27. If $k = \gamma+2$, we conclude directly that $\chi(U(k, \lambda)) = \gamma+1$ because a (k, λ)-colorable graph is $(k - 1)$-colorable for $k > \lambda$.

Additionally, we seek to obtain a negative statement $\neg P(k, \gamma + 1, \lambda)$ from Property 31 to conclude that $\chi(U(k, \lambda)) = \gamma + 1$ by Property 38.

The results obtained by the method described above have been compiled into Table 1.1.

1.5.4 Critical Oligomatic Graphs

As penta$K_{3,3}$ is critical, we have applied a coloring heuristic to $U(k, \lambda)$ for small values of k, λ and proposed the following conjecture, which was settled in [BBd05].

Theorem 39

For every $n \geq 2$, the graph $U(2n+1, n+1)$ is critical with $\chi = 2n$.

Proof

Let G denote $U(2n+1, n+1)$. By Theorem 37, we get immediately that $\chi(G) = 2n$. We must now show that the chromatic number of $G - v$ is $2n-1$ for some vertex v, which we can freely choose, due to the vertex transitivity of G.

Let c be the natural $(2n+1, n+1)$-coloring. In other words, $c(v) = c_v$ with $v = (c_v, \mathcal{C}_v)$. Let $[m]$ denote the set $\{1, \ldots, m\}$. Let c' be the following improper coloring of G

$$c'(v_{c,\mathcal{C}}) = \begin{cases} c & \text{if } 1 \leq c \leq 2n-1 \\ \min([2n-1] \setminus (\mathcal{C} \cup \{c\})) & \text{if } c = 2n \\ \max([2n-1] \setminus (\mathcal{C} \cup \{c\})) & \text{if } c = 2n+1 \end{cases}$$

We will now show that c' results in only one monochromatic edge. Between vertices $v = (c_v, \mathcal{C}_v)$ with $c_v \leq 2n-1$, there are no conflicts by construction of G.

Between vertices $v = (c_v, \mathcal{C}_v)$ and $u = (c_u, \mathcal{C}_u)$ with $c_v \leq 2n-1$ and $c_u \geq 2n$, there are no conflicts by construction of c', since $c'(u) \notin \mathcal{K}_1(u)$, but $c'(v) = c(v) \in \mathcal{K}_1(u)$. Note that $\mathcal{K}_1(u)$ is with respect to the coloring c.

All that remains is to check for conflicts between vertices $v = (2n, \mathcal{C}_v)$ and $u = (2n+1, \mathcal{C}_u)$. Suppose there is an edge $\{u, v\}$ and, therefore, $2n+1 \in \mathcal{C}_v$ and $2n \in \mathcal{C}_u$. If there is a conflict between u and v, then we must have

$$\min([2n-1] \setminus (\mathcal{C}_v \cup \{c_v\})) = \max([2n-1] \setminus (\mathcal{C}_u \cup \{c_u\}))$$

The only possibility for this to happen is

$$\mathcal{C}_v = \{1, \ldots, n-1, 2n+1\} \quad \text{and} \quad \mathcal{C}_u = \{n+1, \ldots, 2n\}$$

with $c'(v) = c'(u) = n$ (note that $|\mathcal{C}_v| = |\mathcal{C}_u| = n$). Now, removing v from G results in a proper $2n-1$ coloring of $G - v$. h.x.

1.5.5 Properties of $U(k, \lambda)$ Determined Using a Heuristic

With the help of coloring heuristics, we were able to determine exactly the chromatic numbers of all $U(k, \lambda)$ with $k \leq 10$. Table 1.2 gives all graphs not covered by Theorem 37 and not yet contained in Table 1.1.

For values of λ between 3 and $\lceil \frac{k-1}{2} \rceil$, we can use the fact that $U(k, \lambda)$ is a subgraph of $U(k', \lambda')$ for $k' \geq k$ and $\lambda' \geq \lambda$. Therefore, if we know $\chi(U(k, \lambda))$ and we find the same value for $\chi_{\mathcal{H}}(U(k', \lambda'))$, we can conclude that $\chi = \chi_{\mathcal{H}}$ for $U(k', \lambda')$.

This argument has been used for all graphs $U(k, 3)$ for $k = 6, \ldots, 12$ which contain $U(5, 3)$ as a 4-chromatic subgraph. Since $\chi_{\mathcal{H}}(U(k, 3)) = 4$ for $k = 6, \ldots, 12$, we conclude that

these graphs are indeed 4-chromatic.

For $U(9,4)$ we have found $\chi_{\mathcal{H}} = 6$. As $U(8,4)$ is provably 6-chromatic (see Table 1.1), we conclude that $U(9,4)$ is 6-chromatic as well. The same argument can be applied to $U(10,4)$.

For $U(11,4)$, we have found $\chi_{\mathcal{H}} = 7$, so the subgraph argument cannot be applied any more. Because $\chi_{\mathcal{H}} = 7$ for $U(11,4) - v$ as well, this graph is not heuristically critical.

For $U(12,4)$, we can easily find a 7-coloring; and, since $U(11,4)$ is a subgraph with $\chi_{\mathcal{H}} = 7$, we conclude that $U(12,4)$ is also not heuristically critical.

An intriguing case is $U(13,3)$, which is heuristically critical with $\chi_{\mathcal{H}} = 5$.

We will finish the discussion of universal graphs with a final conjecture, which we hope will inspire new techniques of proof and, possibly, new and better bounds:

Conjecture 40

The universal graph $U(13,3)$ is 5-chromatic and critical.

1.6 Generalized Oligomatic Graphs and Larger Neighborhoods

Up to this point, we have been concerned uniquely with the number of different colors a vertex v can see in its immediate neighborhood. What happens if we allow larger neighborhoods? In order to approach this question, we can further generalize the construction of universal graphs by extending it to larger neighborhoods $N_q[v]$, $q \geq 1$.

Definition 41

The *q-universal graph* $U^q(k,\lambda)$ is defined by all possible vertices of the form

$$v = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$$

where the $\mathcal{C}_i \subseteq \{1,\ldots,k\}$, $i = 0,\ldots,q$ satisfy

1. $\mathcal{C}_{i-1} \subseteq \mathcal{C}_i$ for $i = 1,\ldots,q$
2. $|\mathcal{C}_0| = 1$
3. $|\mathcal{C}_1| \geq 2$
4. $|\mathcal{C}_q| = \lambda$.

Two vertices $(\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$ and $(\mathcal{C}'_0, \mathcal{C}'_1, \ldots, \mathcal{C}'_q)$ are joined if $\mathcal{C}_0 \neq \mathcal{C}'_0$, and, if for all $i = 1,\ldots q$, we have $\mathcal{C}_{i-1} \subseteq \mathcal{C}'_i$ and $\mathcal{C}'_{i-1} \subseteq \mathcal{C}_i$

Note that the definition of $U^1(k,\lambda)$ results in a graph isomorphic to $U(k,\lambda)$, but the vertices are represented slightly differently. A vertex v of $U(k,\lambda)$ is represented by $(c,\mathcal{C})$,

where c is its natural color and $\mathcal{C}$ is the set of other colors (not containing c) seen by the vertex v. Note that the set $\mathcal{C}$ is of cardinality $\lambda - 1$.

In the generalized definition of $U^1(k, \lambda)$, a vertex v is represented by $(\{c\}, \mathcal{C}_1)$, where c is again its natural color but $\mathcal{C}_1$ now contains c and corresponds exactly to $\mathcal{K}_1(v)$.

As for the graphs $U(k, \lambda)$, the graphs $U^q(k, \lambda)$ have a natural k-coloring such that, for every vertex $v = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$, the number of colors seen at distance q is λ, in other words, $|\mathcal{K}_q(v)| = \lambda$ for all $v \in V$. More generally, for a given vertex $v = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_q)$, we have $\mathcal{C}_i = \mathcal{K}_i(v)$ with respect to the natural coloring of $U^q(k, \lambda)$.

In [EFH+86], several properties of infinite q-universal graphs are given. A part from shift graphs, no results are given for finite q-universal graphs. Below, we will state a theorem for finite graphs.

We may also generalize Definition 23 for larger neighborhoods:

Definition 42

$P^q(k, \gamma, \lambda)$ abbreviates the following statement: there exists a graph G with $\chi > \gamma$ and a k-coloring such $|\mathcal{K}_q(v)| \leq \lambda$ for every $v \in V$.

Upon commencement of research on the topic of oligomatic graphs, we were first interested in the following question:

"What is the smallest integer q such that, in any $(\chi + p)$-coloring of any graph G, there exists a vertex v such that $N_q(v)$ contains at least χ different colors?"

If $p = 0$, then $q = 1$ is sufficient, according to Property 8. Furthermore, for $q \geq \lfloor \frac{\chi}{2} \rfloor$, there is a vertex v such that $|K_q(v)| \geq \chi$. This can be seen easily, if we notice that, in any coloring, there exists a chain of χ vertices where no two vertices have the same color [dWH05]. For the central vertex v of such a chain, we clearly have $|K_q(v)| \geq \chi$.

Theorem 43

Let G be a graph with chromatic number χ, and let p be an non-negative integer. Then, for any $(\chi + p)$-coloring, there exists a vertex v such that $N_{\lceil \frac{p}{2} \rceil + 1}[v]$ contains at least χ different colors.

In the P^q notation, the theorem is equivalent to the statement

$$P^{\left(\lceil \frac{p}{2} \rceil + 1\right)}(\gamma + p + 1, \gamma, \gamma) \text{ is false for } p \geq 0. \tag{1.5}$$

In other words, there is no graph with chromatic number larger than γ with a $(\gamma + p)$-coloring such that $|\mathcal{K}_{\lceil \frac{p}{2} \rceil + 1}(v)| \leq \gamma$ for every vertex v.

To prove the equivalence of the Theorem 43 and (1.5), let us assume the contrary that there is a graph G with $\chi = \gamma + n$ for some $n \geq 1$, and a $(\gamma + p + 1)$-coloring. This leads us to $\gamma + p + 1 = \chi + p'$ with $p' = p - n + 1 \leq p$. Theorem 43 tells us that every vertex v of G sees at least $\chi > \gamma$ different colors at distance $\lceil p'/2 \rceil + 1$. However, since $p' \leq p$, also at distance $\left\lceil \frac{p}{2} \right\rceil + 1$, we have reached the desired contradiction.

Proof of Theorem 43

Let $\mathcal{C} = \{1, \ldots, \chi + p\}$ be the set of colors, $\mathcal{P} = \{1, \ldots, p + 1\}$; and, for $\mathcal{X} \subseteq \mathcal{C}$, let $\overline{\mathcal{X}} := \mathcal{C} \setminus \mathcal{X}$ so that $|\overline{\mathcal{P}}| = \chi - 1$. Additionally, let $V_{\mathcal{P}}$ the set of vertices with colors in $\mathcal{P}$. Let us assume the contrary: there exists a $(\chi + p)$-coloring $c : V \to \mathcal{C}$ such that

$$|\mathcal{K}_{\lceil \frac{p}{2} \rceil + 1}(v)| \leq \chi - 1 \quad \forall v \in V. \tag{1.6}$$

We will exhibit a coloring $c' : V \to \overline{\mathcal{P}}$ using at most $\chi - 1$ colors by recoloring each vertex in $V_{\mathcal{P}}$ with a color in $\overline{\mathcal{P}}$, and this will contradict the definition of χ.

To demonstrate this, we will require additional notation. For $v \in V_{\mathcal{P}}$ and $q \geq 1$, let $\mathcal{Q}_q(v) := \overline{\mathcal{K}_q(v)} \cap \overline{\mathcal{P}}$. Note that, for $q \leq \lceil \frac{p}{2} \rceil + 1$, (1.6) gives $|\mathcal{K}_q(v)| \leq \chi - 1$, so that

$$|\overline{\mathcal{K}_q(v)} \cap \overline{\mathcal{P}}| + |\overline{\mathcal{K}_q(v)} \cap \mathcal{P}| = |\overline{\mathcal{K}_q(v)}| \geq p + 1$$

and

$$|\mathcal{Q}_q(v)| = |\overline{\mathcal{K}_q(v)} \cap \overline{\mathcal{P}}| \geq p + 1 - |\overline{\mathcal{K}_q(v)} \cap \mathcal{P}| = |\mathcal{K}_q(v) \cap \mathcal{P}|. \tag{1.7}$$

Let $G_{\mathcal{P}}$ be the subgraph of G induced by the vertices in $V_{\mathcal{P}}$. We form an oriented graph $\vec{G}_{\mathcal{P}}$ by orienting each edge of $G_{\mathcal{P}}$ from the vertex of smaller color to the vertex of larger color. For each $v \in V_{\mathcal{P}}$, let $r(v)$ ($a(v)$, respectively) be the rank (anti-rank, respectively) of v in $\vec{G}_{\mathcal{P}}$, (in other words, the number of vertices in a longest path ending (starting, respectively) at v). We set $r(v) = 1$ ($a(v) = 1$, respectively) if there is no arc ending (starting, respectively) at v. Note that there is a path with $r(v) + a(v) - 1$ vertices in $\vec{G}_{\mathcal{P}}$, but no path in $\vec{G}_{\mathcal{P}}$ has more than $|\mathcal{P}| = p + 1$ vertices. Therefore, if $q_v = \min\{r(v), a(v)\}$, then

$$q_v \leq \left\lceil \frac{p+2}{2} \right\rceil = \left\lceil \frac{p}{2} \right\rceil + 1, \tag{1.8}$$

such that (1.7) holds for $q = q_v$. The existence of the paths of $r(v)$ and $a(v)$ vertices ensures that $|\mathcal{K}_{q_v}(v) \cap \mathcal{P}| \geq 2q_v - 1 \geq q_v$, so (1.7) gives

$$|\mathcal{Q}_{q_v}(v)| \geq q_v. \tag{1.9}$$

We can now define the coloring $c' : V \to \overline{\mathcal{P}}$. For $v \in V$, let

$$c'(v) = \begin{cases} \min^{r(v)} \mathcal{Q}_{r(v)}(v) & \text{if } v \in V^r = \{v \in V_{\mathcal{P}} \mid r(v) \leq a(v)\} \\ \max^{a(v)} \mathcal{Q}_{a(v)}(v) & \text{if } v \in V^a = \{v \in V_{\mathcal{P}} \mid a(v) < r(v)\} \\ c(v) & \text{if } v \in V^0 = V \setminus V_{\mathcal{P}} \end{cases},$$

where $\min^q \mathcal{X}$ (respectively $\max^q \mathcal{X}$) denotes the qth smallest (qth largest, respectively) element of $\mathcal{X}$; note that these are well defined, by (1.9).

It remains to prove that c' is a coloring. That is, for each edge $\{u, v\} \in E$, $c'(u) \neq c'(v)$. Without loss of generality, we may assume that $c(u) < c(v)$. Note that, if $0 < s < s'$ then, because u, v are adjacent, $N_s[u] \subseteq N_{s'}[v]$ and $N_s[v] \subseteq N_{s'}[u]$. It follows that $\mathcal{K}_s(u) \subseteq \mathcal{K}_{s'}(v)$ and $\mathcal{K}_s(v) \subseteq \mathcal{K}_{s'}(u)$. As a consequence, if $u, v \in V_{\mathcal{P}}$, then

$$\mathcal{Q}_{s'}(v) = \overline{\mathcal{K}_{s'}(v)} \cap \overline{\mathcal{P}} \subseteq \overline{\mathcal{K}_s(u)} \cap \overline{\mathcal{P}} = \mathcal{Q}_s(u), \tag{1.10}$$

and similarly

$$\mathcal{Q}_{s'}(u) \subseteq \mathcal{Q}_s(v). \tag{1.11}$$

We now distinguish between three unique cases.

Case 1 $\boxed{v \in V^0}$

If $u \in V^0$, then $c'(u) = c(u) \neq c(v) = c'(v)$. If $u \in V_{\mathcal{P}}$, then $c'(u) \in \mathcal{Q}_q(u) \subseteq \overline{\mathcal{K}_q(u)}$ where $q = \min\{r(u), a(u)\}$, and $c(v) \in \mathcal{K}_1(u) \subseteq \mathcal{K}_q(u)$, such that $c'(u) \neq c'(v) = c(v)$.

Case 2 $\boxed{u \in V^a \text{ and } v \in V_{\mathcal{P}}}$

Since $c(u) < c(v)$, it follows that $r(u) < r(v)$ and $a(u) > a(v)$, so, since $u \in V^a$,

$$a(v) < a(u) < r(u) < r(v).$$

Thus $v \in V^a$, and

$$c'(v) = \max{}^{a(v)}\mathcal{Q}_{a(v)}(v) > \max{}^{a(u)}\mathcal{Q}_{a(v)}(v)$$

because $a(v) < a(u)$, and

$$\max{}^{a(u)}\mathcal{Q}_{a(v)}(v) \geq \max{}^{a(u)}\mathcal{Q}_{a(u)}(u) = c'(u)$$

because $\mathcal{Q}_{a(u)}(u) \subseteq \mathcal{Q}_{a(v)}(v)$ by (1.11).

Case 3 $\boxed{u \in V^r \text{ and } v \in V_{\mathcal{P}}}$

Since $c(u) < c(v)$, it follows that $r(u) < r(v)$ and $a(u) > a(v)$. Suppose first that $v \in V^r$. Then

$$c'(v) = \min{}^{r(v)}\mathcal{Q}_{r(v)}(v) > \min{}^{r(u)}\mathcal{Q}_{r(v)}(v)$$

because $r(u) < r(v)$, and

$$\min{}^{r(u)}\mathcal{Q}_{r(v)}(v) \geq \min{}^{r(u)}\mathcal{Q}_{r(u)}(u) = c'(u)$$

because $\mathcal{Q}_{r(v)}(v) \subseteq \mathcal{Q}_{r(u)}(u)$ by (1.10).

We may, therefore, assume from now on that $v \in V^a$. We will prove that

$$|\mathcal{Q}_{a(v)}(v) \cap \mathcal{Q}_{r(u)}(u)| \geq a(v) + r(u), \tag{1.12}$$

from which it will follow immediately that

$$c'(v) = \max{}^{a(v)}\mathcal{Q}_{a(v)}(v) \neq \min{}^{r(u)}\mathcal{Q}_{r(u)}(u) = c'(u), \tag{1.13}$$

as required.

Note that, since $u \in V^r$ and $v \in V^a$, it follows from (1.8) that $r(u), a(v) \leq \left\lceil \frac{p}{2} \right\rceil + 1$. Note also that there is a path P of $a(v) + r(u)$ vertices in $\vec{G}_{\mathcal{P}}$, passing along the arc (u, v).

If $r(u) < a(v)$, then $|\mathcal{Q}_{a(v)}(v)| \geq |\mathcal{K}_{a(v)}(v) \cap \mathcal{P}| \geq a(v) + r(u)$ by (1.7) and the existence of the path P, and $\mathcal{Q}_{a(v)}(v) \subseteq \mathcal{Q}_{r(u)}(u)$ by (1.10), and this implies (1.12) and, therefore, (1.13).

If $r(u) > a(v)$, then $|\mathcal{Q}_{r(u)}(u)| \geq |\mathcal{K}_{r(u)}(u) \cap \mathcal{P}| \geq a(v) + r(u)$ by (1.7) and the existence of the path P, and $\mathcal{Q}_{r(u)}(u) \subseteq \mathcal{Q}_{a(v)}(v)$ by (1.11), and this implies (1.12) and, therefore, (1.13).

Finally, if $r(u) = a(v)$ then $a(v) \leq \frac{p+1}{2}$ because the path P has at most $|\mathcal{P}| = p + 1$ vertices, so $a(v) \leq \lceil \frac{p}{2} \rceil$, and $a(v) + 1 \leq \lceil \frac{p}{2} \rceil + 1$. $|\mathcal{Q}_{a(v)+1}(v)| \geq |\mathcal{K}_{a(v)+1} \cap \mathcal{P}| \geq r(u) + a(v)$ by (1.7) and the existence of the path P, and $\mathcal{Q}_{a(v)+1}(v) \subseteq \mathcal{Q}_{a(v)}(v)$ (clearly) and $\mathcal{Q}_{a(v)+1}(v) \subseteq \mathcal{Q}_{r(u)}(u)$ by (1.10), which implies (1.12) and, finally, (1.13).

h.x.

1.7 Special Classes of Graphs

This section will follow the investigations of the existence and properties of oligomatic graphs among a number of special graph classes.

We have already seen that perfect graphs are not oligomatic.

1.7.1 Planar Graphs

As oligomatic graphs are at least 4-chromatic, planar graphs seem to be an interesting class because their chromatic number is less than or equal to 4 [AH77, AHK77].

The question of whether or not there exists an oligomatic planar graph is still open. However, for some special subclasses of planar graphs, the question has been settled as negative.

Definition 44

A *Halin graph* is a planar graph consisting of a tree with a cycle connecting the leaves of the tree and no vertex of degree two.

In [BBd05], an explicit 3-coloring has been given for Halin graphs, except when the graph is an odd wheel, which is 4-chromatic. It is clear that odd wheels are not oligomatic because the center vertex always sees at least 4 colors. For the other cases, we have shown that there are no oligomatic graphs with chromatic number 3 or less.

Property 45

There is no oligomatic Halin graph.

We will now describe some properties an oligomatic planar graph G should have. First, G must be 4-chromatic and admit a $(k, 3)$-coloring. This implies that the open neighborhood of every vertex induces a bipartite graph because there is a coloring such that every vertex sees only two other colors apart from his own.

Clearly, penta$K_{3,3}$ is not planar, since it contains a $K_{3,3}$, a complete bipartite graph on 3 and 3 vertices. In [BBd05], it has also been shown that penta$K_{3,3}$ contains a K_5 minor. From this, it follows that, for every k, the graph $U(k, 3)$ is not planar either, as it contains $U(5, 3)$ (penta$K_{3,3}$) as a subgraph.

1.7.2 Claw-free Graphs

A claw is the graph $K_{1,3}$:

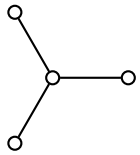

FIGURE 4 A claw.

A graph is called *claw-free* if it contains no claw as an induced subgraph. Claw-free graphs have some interesting properties:

Minty [Min80] showed that the maximum independent set problem, which is in general $\mathcal{NP}$-complete on general graphs, can be solved in polynomial time on a claw-free graph. His proof used the algorithm of Edmonds [Edm65] to find the maximum weighted matching. However, Nakamura and Tamura [NT01] showed that this algorithm fails in some special cases and provided modifications to correct this error.

The question about the existence of oligomatic claw-free graphs is open. However, we have the following negative result:

Theorem 46

No claw-free graph admits a $(\chi + 1, \chi - 1)$-coloring.

In order to prove the above theorem, we need the notion of list-coloring, along with an important property.

Definition 47

Let $G = (V, E)$ be a graph. Let $\mathcal{C}$ be a set of available colors and, for every vertex $v \in V$, let $\ell_v \subseteq \mathcal{C}$ be a set (for historical reasons; also called a *list*) of possible colors for v. A *list-coloring* is a coloring c such that $c(v) \in \ell_v$ for every vertex v.

Property 48 [Viz76, ERT79]

A connected graph G is list-colorable if the cardinality of every list is at least $\Delta(G)$ unless G is a complete graph or G is an odd cycle.

Using Property 48 we can now prove Theorem 46.

Proof of Theorem 46

Let G be a claw-free graph and Δ its maximal degree.

We will suppose the opposite that c is a $(\chi+1, \chi-1)$-coloring of the vertices of G.

Consider the induced subgraph H of G formed by all vertices colored with colors χ and $\chi+1$. Because every vertex sees at most $\chi-1$ different colors, at least two other colors are available (in other words not seen). For isolated vertices of H, we can assign a new color in $\{1, \ldots, \chi-1\}$ without creating conflicts. We may now remove those vertices from H to obtain H'. H' consists of vertices with colors χ and $\chi+1$ such that every vertex of color χ is adjacent to a vertex of color $\chi+1$ and vice versa. As the vertices of H' are 2-colored, H' does not contain odd cycles, and its maximal degree $\Delta(H')$ is at most 2. Otherwise, it would contain a claw.

For every vertex v of H', we have a list ℓ_v of at least two available colors strictly less than χ because every vertex in H' sees both colors χ and $\chi+1$.

Consider the list-coloring problem on H' with the lists ℓ_v. We can apply the Property 48 to every connected component of H' to finally obtain a coloring of H' with colors smaller than χ. Moreover, the recolored vertices of H' do not conflict with the coloring of the other vertices of G. Therefore, we have found a coloring of the vertices of G with $\chi-1$ colors, which is impossible. h.x.

1.7.3 Line Graphs

Line graphs are an important subclass of claw-free graphs.

Definition 49

A *line graph* $L(G) = (V_L, E_L)$ is obtained from a graph $G = (V, E)$ by setting $V_L := E$ and joining two vertices of $L(G)$ if the corresponding edges in G are adjacent.

Definition 50

An *edge coloring* c of a graph G is a function $c : E \to \mathbb{N}$ such that

$$c(e) \neq c(f) \quad e, f \in E,\ e \text{ and } f \text{ adjacent.}$$

A *k-edge coloring* is an edge coloring using no more than k different colors.
The smallest k for which a k-edge coloring exists is called the *chromatic index* and is denoted by

$$\chi'(G).$$

It is trivially apparent that the edge coloring problem on G is equivalent to the vertex coloring problem on $L(G)$. It is clear, as well, that $\Delta(G) \leq \omega(L(G))$, where Δ is the maximum degree of a vertex and ω is the size of a maximum clique.

By Vizing's theorem [Viz64], every graph is $(\Delta + 1)$-edge colorable. It is clear that at least Δ colors are needed. If the edges of a graph can be Δ-colored it is called a *class 1 graph*; otherwise, it is called a *class 2 graph*.

The question of whether or not oligomatic line graphs exist is open. However, we provide two negative results:

Theorem 51

No line graph $L(G)$ of a class 1 graph G is oligomatic.

Proof

Since G is class 1, $\chi'(G) = \Delta(G)$. Since we have $\chi(L(G)) = \chi'(G)$ and $\omega(L(G)) \geq \Delta(G)$, we have $\chi(L(G)) \leq \omega(L(G))$, which shows that a vertex of a maximum clique always sees at least $\chi(L(G))$ colors. h.x.

The only candidates left for oligomatic line graphs are those obtained from class 2 graphs. Recall that line graphs are also claw-free graphs; and, therefore, Theorem 46 applies.

For line graphs in general, we provide a more general result than Theorem 46.

Theorem 52

No line graph admits a $(k, \chi - 2)$-coloring.

Proof

Let G be a graph and $H = L(G)$ its line graph. Now, $\chi(H) = \chi'(G) \leq \Delta(G) + 1$ and $\omega(H) \geq \Delta(G)$. Therefore,

$$\omega(H) \geq \chi(H) - 1.$$

Consider a vertex v of a maximum clique. Clearly, it sees at least $\omega(H)$ different colors,

and, therefore, a $(k, \chi(H) - 2)$-coloring cannot exist. h.x.

To conclude this chapter, we suggest investigating the possibility of constructing line graphs in a way analogous to the construction of universal graphs, to either find an example of an oligomatic line graph or to stimulate new ideas which may prove their nonexistence.

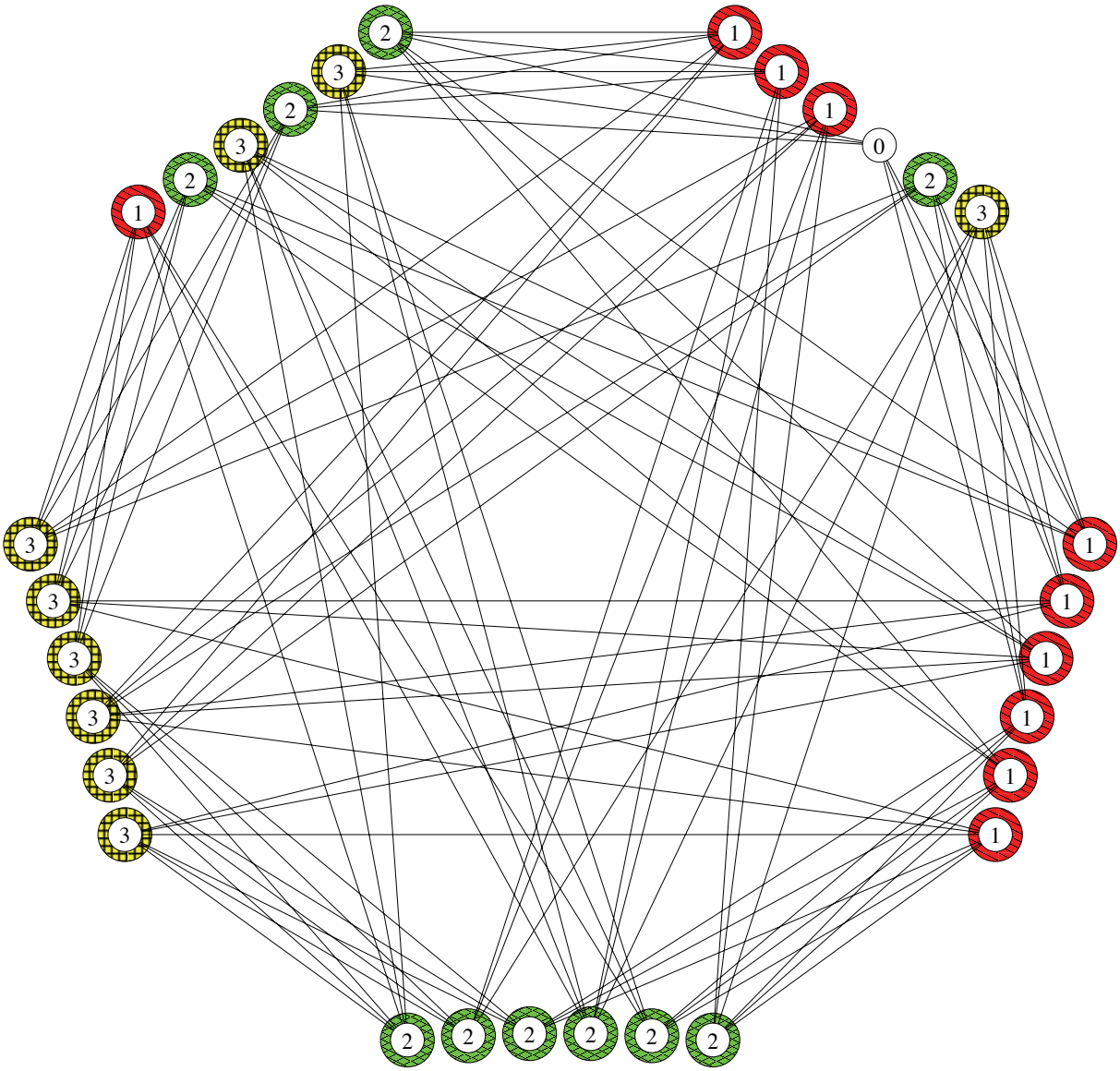

Figure 1.1: The graph penta$K_{3,3}$ with a partial 3-coloring with one uncolored vertex (color 0).

γ'	γ	p'	p	$P(k', \gamma', \lambda')$	upper bound	$\chi(U(k', \lambda'))$
3	-	1	-	$P(5,3,3)$	4-coloring	$\chi(U(5,3)) = 4$
4	-	1	-	$P(6,4,4)$	5-coloring	$\chi(U(6,4)) = 5$
5	6	1	2	$P(7,5,4)$	$\neg P(9,6,4)$	$\chi(U(7,4)) = 6$
5	6	2	2	$P(8,5,4)$	$\neg P(9,6,4)$	$\chi(U(8,4)) = 6$
6	-	1	-	$P(8,6,5)$	7-coloring	$\chi(U(8,5)) = 7$
7	-	1	-	$P(9,7,5)$	8-coloring	$\chi(U(9,5)) = 8$
8	-	1	-	$P(10,8,6)$	9-coloring	$\chi(U(10,6)) = 9$
8	9	2	2	$P(11,8,5)$	$\neg P(12,9,5)$	$\chi(U(11,5)) = 9$
9	-	1	-	$P(11,9,6)$	10-coloring	$\chi(U(11,6)) = 10$
10	-	1	-	$P(12,10,7)$	11-coloring	$\chi(U(12,7)) = 11$
11	-	1	-	$P(13,11,7)$	12-coloring	$\chi(U(13,7)) = 12$
11	12	2	2	$P(14,11,6)$	$\neg P(15,12,6)$	$\chi(U(14,6)) = 12$
11	12	3	3	$P(15,11,6)$	$\neg P(16,12,6)$	$\chi(U(15,6)) = 12$
12	-	1	-	$P(14,12,8)$	13-coloring	$\chi(U(14,8)) = 13$
13	-	1	-	$P(15,13,8)$	14-coloring	$\chi(U(15,8)) = 14$
14	-	1	-	$P(16,14,9)$	15-coloring	$\chi(U(16,9)) = 15$
14	15	2	2	$P(17,14,7)$	$\neg P(18,15,7)$	$\chi(U(17,7)) = 15$
15	-	1	-	$P(17,15,9)$	16-coloring	$\chi(U(17,9)) = 16$
15	16	3	3	$P(19,15,7)$	$\neg P(20,16,7)$	$\chi(U(19,7)) = 16$
16	-	1	-	$P(18,16,10)$	17-coloring	$\chi(U(18,10)) = 17$
17	-	1	-	$P(19,17,10)$	18-coloring	$\chi(U(19,10)) = 18$
17	18	2	2	$P(20,17,8)$	$\neg P(21,18,8)$	$\chi(U(20,8)) = 18$
18	-	1	-	$P(20,18,11)$	19-coloring	$\chi(U(20,11)) = 19$
19	-	1	-	$P(21,19,11)$	20-coloring	$\chi(U(21,11)) = 20$
19	20	3	3	$P(23,19,8)$	$\neg P(24,20,8)$	$\chi(U(23,8)) = 20$
19	20	4	4	$P(24,19,8)$	$\neg P(25,20,8)$	$\chi(U(24,8)) = 20$

Table 1.1: Chromatic numbers of some universal graphs, obtained by combining Theorem 27 and Property 31. The values γ' and p' are associated with the positive statement $P(k', \gamma', \lambda')$, where $k' = \gamma' + p' + 1$ and $\lambda' = \lfloor \gamma'/(p'+1) \rfloor + p' + 1$. Where given, the values γ and p (with $\gamma \geq p(p+1)$) are associated with the negative statement $\neg P(k, \gamma, \lambda)$ where $k = \gamma + p + 1$ and $\lambda = \lfloor \gamma/(p+1) \rfloor + p$. Every $U(k, \lambda)$ with $k \geq \lambda + 2$ has a trivial $k-1$ coloring.

k	λ	$\chi(U(k,\lambda))$	critical	$P(k,\gamma,\lambda)$	contains
6	3	**4**	**no**	$P(6,3,3)$	$U(5,3)$
7	3	**4**	**no**	$P(7,3,3)$	$U(5,3)$
8	3	**4**	**no**	$P(8,3,3)$	$U(5,3)$
9	3	**4**	**no**	$P(9,3,3)$	$U(5,3)$
9	4	**6**	**no**	$P(9,5,4)$	$U(8,4)$
10	3	**4**	**no**	$P(10,3,3)$	$U(5,3)$
10	4	**6**	**no**	$P(10,5,4)$	$U(8,4)$
10	5	**8**	**no**	$P(10,7,5)$	$U(9,5)$
11	3	**4**	(no)	$P(11,3,3)$	$U(5,3)$
11	4	(7)	(no)	$(P(11,5,4))$	not applicable
12	3	**4**	**no**	$P(12,3,3)$	$U(5,3)$
12	4	(7)	(no)	$(P(12,5,4))$	$(U(11,4))$
13	3	(5)	(yes)	$(P(13,4,3))$	not applicable

Table 1.2: Chromatic numbers and criticality, determined using the Foo-Partialcol coloring heuristic. Bold values are exact (lower theoretical and heuristical upper bounds). Values in parenthesis are conjectured, based on the same coloring heuristic. The fifth column shows the equivalent γ-critical statements. For unproven chromatic numbers, the corresponding conjectured P-statement is given in parenthesis as well.

Chapter 2

Improving Tabu Search

The aim of this chapter is to prepare the field for the next two chapters, where we will present heuristics for the graph coloring problem and for a weighted version of the graph coloring problem.

In this chapter, we will present several methods to improve a powerful heuristic called *tabu search*. The goal is to provide simple tools to adjust automatically the *tabu tenure*, the most important parameter of *tabu search*.

First, we will give a definition of a *generic tabu search*. Then, we will give a classification for methods to adjust the *tabu tenure*. Next, we will narrow our focus to graph coloring problems and present ways of measuring similarity and distances between different colorings of a graph. Finally, we present several methods to adjust the *tabu tenure*. These methods have been applied to different heuristics described in the following chapters.

2.1 Tabu Search

Tabu search is a local search algorithm which was originally proposed by Glover [Glo89, Glo90] and Hansen [Han86]. Its basic version can be described as follows:

Define the *solution space* S to be the set of configurations (called *solutions*) to be considered for a combinatorial optimization problem. Depending on the problem, the term *solution* might be misleading because the search space may contain many configurations which are not feasible for the initial problem. For example, for the k-coloring problem, the TABUCOL [HdW87] algorithm considers all possible k-partitions of the vertex set (and, therefore, also improper colorings).

Let f be an objective function which must be minimized over S.

With each solution $s \in S$, a set $N(s)$ is associated and is called the *neighborhood* of s. The solutions in $N(s)$, also called *neighbors* of s, are obtained from s by performing small changes on s called *moves*.

The tabu search algorithm needs an initial solution $s_0 \in S$ as input. Then, the algorithm generates a sequence of solutions $s_1, s_2, \ldots$ in the search space S such that $s_{i+1} \in N(s_i)$. When a move is performed from s_i to s_{i+1}, the inverse of that move is stored in a *tabu list* L. For the following t iterations, where t is the *tabu tenure* (also called *tabu list length*), a move stays *tabu* and cannot be used (with some exceptions) to generate a neighbor

solution. The solution s_{i+1} is computed as

$$s_{i+1} = \arg \min_{s \in N'(s_i)} f(s),$$

where $N'(s)$ is a subset of $N(s)$ containing all solutions s' which can be obtained from s either by performing a move that is not in L (i.e. not tabu) or such that $f(s') < f(s^\star)$, where $s^\star$ is the best solution encountered along the search so far. The process is stopped when a fixed number of iterations without improving $s^\star$ have been performed.

This *generic tabu search* is summarized in Algorithm 1, where we suppose that we know the value of an optimal solution. A move which allows us to generate s' from s is denoted by $(s \hookrightarrow s')$. The algorithm stops as soon as $i_{\max}$ iterations without improvement of $s^\star$ have been performed or as soon as an optimal solution has been found.

In practice, one rarely implements a real list of tabu moves. One rather stores a vector which for each possible move contains the iteration number up to which a move stays tabu. If a move is executed the corresponding entry in the vector is set to the current number of iterations plus the tabu tenure t. A move is tabu, if the current number of iterations is smaller than the corresponding entry in the vector. This is not only easier to implement but is also faster in execution because to test whether or not a move is tabu takes constant time. Testing the presence of an element in the tabu list takes $O(t)$ time, where t is the length of the list. Moreover, there will be some additional time required for adding and deleting elements from the list.

Interpretation of the Tabu List

A tabu search without a tabu list (or with list length zero) is simply a deepest descent method.

The basic idea behind introducing a tabu list is to enable the search process to escape local minima.

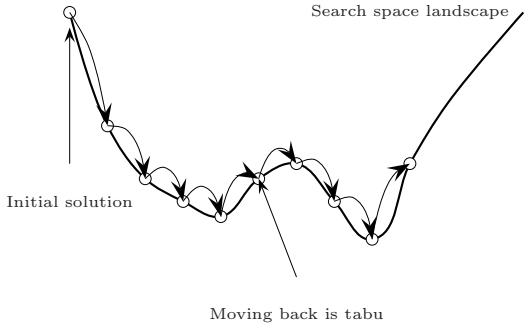

FIGURE 5 A tabu search behaves like a steepest descent, until a local minimum is reached. The tabu list then forces the search to choose solutions with inferior quality, which allows an escape to local minima.

Ideally, one should store complete solutions in the tabu list and forbid visiting these solutions again. The problem with this implementation is that a solution may well be

a complex object whose storage may require a large amount of memory. Storing a large number of solutions may not be possible. Moreover, it can be very time consuming to check whether a given solution is already in the tabu list or not, especially in the case that there are several representations for equivalent solutions. This is the case for graph coloring, where a permutation of the colors results in an equivalent solution.

However, it is possible to approximate an explicit check of repetitions of solutions using hashing techniques. In [GL97] a hash function is presented which is particularly well suited for a tabu search algorithm. Suppose a solution s can be represented by a vector of l integer numbers $(s_1, \ldots, s_l)$. The hash value of the solution s would be

$$H(s) = \sum_{i=1}^{l} z_i s_i \bmod h$$

where z_i are random numbers to be initialized at the beginning of the search process and where h is the number of buckets for the hash. This hash function H may not have the best hashing properties; however, if a move changes only a constant number of components of s, the difference between the previous hash value and the new hash value can be computed in constant time.

2.2 Managing the Tabu Tenure

Many variants and extensions of the basic tabu search can be found in [GL97]. Most of these extensions deal with the organization of the neighborhood of the search and how to choose the next solution in the neighborhood.

We will focus on methods to dynamically adjust the tabu tenure (length of the tabu list) along the search, according to different criteria. We will first define a classification of such methods and give some known examples.

We will define three groups of methods to adjust the tabu tenure: *static*, *dynamic* and *reactive* schemes. Each method may be used in either a *deterministic* or a *randomized* way.

After defining two measures of distance between colorings, we will propose several schemes of automatic adjustment of the tabu tenure. We have implemented these schemes for at least one of the heuristics presented in the next chapters.

2.2.1 Static Tabu Tenure

We define a *static tabu tenure* as a tenure which does not change during the search process.

In the original version of tabu search, the tabu tenure is chosen once and for all and is kept fixed for all instances. This leads to results that compare poorly to other methods of choosing the tabu tenure.

Letting the user supply the tabu tenure is not desirable, because the user should not be part of a heuristic.

An improvement can be made by choosing the tabu tenure as a function of the instance size or, even better, as a function of the mean size of the neighborhood of a solution.

The idea behind this is that the larger the neighborhood of a solution s is, the more possibilities there are to reach this solution. Therefore, the tabu tenure should be larger in order to avoid revisiting s.

2.2.2 Dynamic Tabu Tenure

We define a *dynamic tabu tenure* as a tenure which depends on the current solution and on the move which has been executed to obtain the current solution. No information about previous visited solutions is required.

The Tenure as a Function of the Quality of a Solution

For the graph coloring heuristic TABUCOL, which uses improper k-colorings, the tenure αn_c [FF96, GH99] where α is a constant (typically 0.6) and n_c is the number of vertices involved in a conflict in the current improper coloring, has proven to be very efficient over a wide range of instances.

The above method can be generalized to other problems, provided that the value of the optimal solution is known. This is the case for feasibility problems where the tenure will be determined to be proportional to the number of constraint violations.

If the value of an optimum solution is not known, the above approach might not be a good idea. Consider an instance of an optimization problem where the costs in Swiss francs have to be minimized. Suppose that there is dynamic tabu scheme which adjust the tabu tenure using a function of the costs and that it is well tuned to produce good results when measuring the costs in Swiss francs. But if the costs are measured in English pounds, the performance might suffer, because value of the objective function is much lower, even though the underlying problem has not changed.

One could compare the quality of the current solution with the quality of the best solution found so far, but this does not fit our definition of a *dynamic tabu tenure*, since it involves information about previously visited solutions.

The Tenure as a Function of the Last Move

Suppose we have a method to measure the *impact* of a move. A move with a big *impact* will stay tabu for a longer time than a move with little *impact*. The impact of a move can be defined in different ways, depending on the problem. It should express how much the solution is internally changed. A good measure for this would be the number of constraints or contributions to the objective function involved [VH01]. To illustrate this method, consider the k-coloring problem where the search space is the set of all k-partitions. A move consists in changing the color of a vertex v from color i to color j. For such a move, one can compute the number of resolved conflicts in which the vertex v was involved with neighbors in color i plus the number of newly created conflicts in color j that involve vertex v.

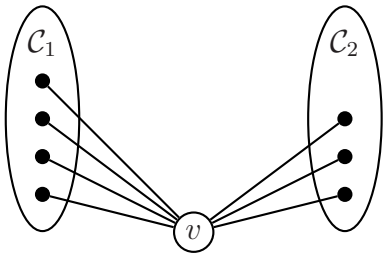

FIGURE 6 A vertex v is moving from color 1 to color 2. v had four neighbors in $\mathcal{C}_1$, so v's removal from C_1 resolves four conflicts. However, v will have three neighbors in C_2, so three conflicts will be added. Therefore, the difference in the objective function is 3-4 = -1, while the impact of the move is 3+4 = 7.

The difference of the objective function caused by a move is not a good indicator of a move's impact, because the difference is most likely to be close to zero once the search has reached a first local minimum. This is because the algorithm always chooses the most profitable non-tabu move unless a new best solution is discovered.

However, for the TABUCOL algorithm, using a dynamic tenure based on the impact measure defined as above leads to poor results, because the TABUCOL algorithm tends to isolate monochromatic edges. In other words, once the search has settled, most moves have an impact of 2 and almost all moves have an impact between 1 and 3 inclusive.

2.2.3 Reactive Tabu Tenure

We define a *reactive tabu tenure* as a tenure determined on the basis of (possibly) the entire search history. We use this more general definition for our purposes.

The term *reactive tabu search* was first introduced by Battiti and Tecchiolli in [BT94].

The Reactive Tabu Scheme by Battiti and Tecchiolli

They propose modifying the size of the tabu tenure when a *cycle* occurs, i.e. when a solution is visited twice by the search process. In their algorithm, an explicit check for the repetition of solutions is added to the basic scheme of tabu search. The appropriate tabu tenure is learned in an automated fashion by reacting to the occurrence of cycles. Each time a cycle is detected, the tabu tenure is increased because we must assume that it was too small to prevent the detected cycle. The tabu tenure is decreased if no cycle is detected during a given number of iterations.

In addition, if the search appears to be repeating an excessive number of solutions too often, then the search is diversified by making a number of random moves proportional to the average of the cycle length in order to escape a local attractor.

2.2.4 Randomizing the Tabu Tenure

Very often, one can improve the performance of a tabu search by randomizing the tabu tenure.

Suppose that we use some arbitrary scheme to determine the tabu tenure t. Then, instead of using t as the tabu tenure, we determine a randomized tenure t_r according to some distribution which normally depends on t.

A typical distribution will be a uniform distribution over an interval $[a(t), b(t)]$:

$$t_r := \text{UNIFORM}(a(t), b(t)).$$

Typically a and b are chosen more or less proportional to t. For the TABUCOL heuristic, a very good performance has been reported for $a(t) = t$ and $b(t) = t + 10$ [FF96, GH99].

2.3 Distances and Similarity for Graph Colorings

To adjust the tabu tenure during the search process, some schemes require that the current solution is compared to previously visited solutions in order to adjust the tabu tenure and evaluate the progress of the search process.

These schemes often need to quantify, for two given solutions how similar the two solutions are to each other or how far apart they are from each other. We will present two measures which have been used in our heuristics.

2.3.1 Hamming Distance

The easiest and most simple distance of a graph coloring problem is the Hamming distance [Ham50]. This distance simply counts the number of differences between the components of two vectors. The Hamming distance between the vectors $(1, 1, 3)$ and $(1, 2, 3)$ would simply be 1 because they differ only in the second component. The main advantages of the Hamming distance are that it is very simple to compute and that it can be applied to any problem which encodes solutions component-wise, which is the case for virtually any combinatorial problem. If a move $(s \hookrightarrow s')$ changes only a constant number of components, the Hamming distance of s' to a reference solution s_{ref} can be recomputed in constant time, based on the distance of s to s_{ref}.

2.3.2 Hamming Distance for Colorings

Given a k-coloring $c : V \to \{1, \ldots, k\}$ and a k'-coloring $c' : V \to \{1, \ldots, k'\}$, we define

$$d(c, c') = |\{v \in V \mid c(v) \neq c'(v)\}|.$$

The disadvantage of this distance comes from the fact that $d(c, c')$ can be large if c' is obtained from c by simply permuting some colors, even if the two colorings are equivalent and represent the same partition of the vertices into stable sets.

Despite this obvious disadvantage, we have obtained good results for the weighted coloring problem.

2.3.3 Topological Distance in a Search Space

For every solution s in the search space S, a neighborhood $N(s)$ is defined. This defines a topology on S. Using this topology, a distance measure can be defined as follows: The distance between two solutions is the minimum number of moves required to transform the first solution into the second. However, depending on the topology of the search space, actually calculating this number might be very complicated and, in the worst case, as complicated as the optimization problem to solve.

For the particular neighborhood structure where the neighbors of a solution s are all solutions which differ in exactly one component of their encodings then the topological distance is equivalent to the Hamming distance.

2.3.4 Similarity of Colorings

We present another attempt to define a distance between two colorings which does not suffer from the disadvantage from which the Hamming distance suffers. We consider colorings represented as partitions of the vertex set V into k sets. Let c be a k-coloring represented by the partition $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ and c' be a k'-coloring represented by the partition $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_{k'}\}$. First, we define a measure of the similarity of two color classes $\mathcal{C}_i$ and $\mathcal{C}'_j$:

$$Sim(\mathcal{C}_i, \mathcal{C}'_j) = \begin{cases} 1 & \text{if } \mathcal{C}_i = \mathcal{C}'_j = \emptyset \\ \frac{|\mathcal{C}_i \cap \mathcal{C}'_j|}{|\mathcal{C}_i \cup \mathcal{C}'_j|} & \text{otherwise} \end{cases}.$$

This measure equals 1 if and only if $\mathcal{C}_i = \mathcal{C}'_j$ and 0 if $\mathcal{C}_i \cap \mathcal{C}'_j = \emptyset$ (with at least one of the sets non-empty). Moreover, we have $Sim(\mathcal{C}_i, \mathcal{C}'_j) = Sim(\mathcal{C}'_j, \mathcal{C}_i)$, which is natural to expect from any such measure. This measure of similarity is called *Jaccard's coefficient* and is used in information retrieval. We have chosen this measure because it is intuitive. There are some other measures with similar properties, such as, for instance, *Dice's coefficient* $\frac{2|X \cap Y|}{|X|+|Y|}$ or the *overlap coefficient* $\frac{|X \cap Y|}{\min(|X|,|Y|)}$ [Hub82], but we have not investigated them.
We can now, based on the definition of *Sim*, define a measure of similarity of two colorings:

$$Sim(c, c') = \frac{2}{k + k'} \sum_{\substack{1 \le i \le k \\ 1 \le j \le k'}} Sim(\mathcal{C}_i, \mathcal{C}'_j).$$

Here again, if $c = c'$, then $Sim(c, c') = 1$ because all terms $Sim(\mathcal{C}_i, \mathcal{C}'_j)$ will be zero, except the k terms, where the sets $\mathcal{C}_i$ and $\mathcal{C}'_j$ are equal. Note that $Sim(\cdot, \cdot) > 0$. To see this, consider a vertex $v \in \mathcal{C}_i$. It is also contained in $\mathcal{C}'_j$ for $j = c'(v)$ and, therefore, $|\mathcal{C}_i \cap \mathcal{C}'_j| \ge 1$.
Note that the definition does not depend on the order of the sets $\mathcal{C}_i$.

Finally we can define a distance d based on the measure of similitude as follows:

$$d(c, c') = 1 - Sim(c, c').$$

This distance has the properties that if $c = c'$ then $d(c, c') = 0$ and, if $c \neq c'$, then $d(c, c') > 0$.

Average Similarity Between two Colorings

In order to gain a better understanding of this measure of similarity, we will estimate the similarity between two random k-colorings of a graph without edges. We will perform both an empirical and an analytical estimation.

Estimation by simulation For different values of n (the number of vertices) and k (the number of colors), we have generated random assignments of the colors $\{1, \ldots, k\}$ to every vertex. Then, for two such random colorings, the similarity has been computed. See Table 2.1 for the results.
The mean is close to $\frac{k}{2k-1}$. However, this is not the theoretical value but probably the asymptotic limit when n grows, as numerical tests suggest.

Theoretical analysis We were not able to provide an exact value for the average similarity of two random k-colorings but we provide a first order approximation thereof.
Let V be a set of n vertices, and c and c' two random k-colorings in the sense that the probability for a vertex v having color i is simply $\frac{1}{k}$.
First, we compute the expectation of the intersection

$$E\left(|\mathcal{C}_i \cap \mathcal{C}'_j|\right) = \sum_{v \in V} 1 \cdot P(c(v) = i, c'(v) = j) = \frac{n}{k^2}$$

and the expectation of the union

$$\begin{aligned} E\left(|\mathcal{C}_i \cup \mathcal{C}'_j|\right) &= E\left(|\mathcal{C}_i| + |\mathcal{C}'_j| - |\mathcal{C}_i \cap \mathcal{C}'_j|\right) \\ &= 2\sum_{v \in V} 1 \cdot P(c(v) = i) - \frac{n}{k^2} \\ &= \frac{2n}{k} - \frac{n}{k^2}\,. \end{aligned}$$

Of course, in general for two random variables X and Y, $E(X/Y) \neq E(X)/E(Y)$. But $E(X)/E(Y)$ is a first order approximation for $E(X/Y)$. Using the above results we obtain

$$E\left(Sim(\mathcal{C}_i, \mathcal{C}'_j)\right) = E\left(\frac{|\mathcal{C}_i \cap \mathcal{C}'_j|}{|\mathcal{C}_i \cup \mathcal{C}'_j|}\right) \approx \frac{E\left(|\mathcal{C}_i \cap \mathcal{C}'_j|\right)}{E\left(|\mathcal{C}_i \cup \mathcal{C}'_j|\right)} = \frac{\frac{n}{k^2}}{\frac{2n}{k} - \frac{n}{k^2}} = \frac{1}{2k-1};$$

and, using the definition

$$E(Sim(c, c')) = \frac{2}{k+k} \sum_{\substack{1 \leq i \leq k \\ 1 \leq j \leq k}} E\left(Sim(\mathcal{C}_i, \mathcal{C}'_j)\right) = \frac{1}{k} k^2 E\left(Sim(\mathcal{C}_1, \mathcal{C}'_1)\right),$$

we can give a first order approximation

$$E(Sim(c, c')) \approx \frac{k}{2k-1},$$

which corresponds within less than one standard deviation to the numerical values obtained by the simulation.

This analysis shows that two k-colorings with a similarity of 0.5 are in fact "very different" in the sense that they are less similar than two random colorings in average.

Computational Complexity of the Similarity Measure

The proposed similarity measure has useful properties, but the computation of the double sum is computationlly intensive.

Suppose we have two k-colorings c and c' encoded as a function $V \to \{1, \ldots, k\}$. In order to compute $\mathcal{C}_i \cap \mathcal{C}'_j$, we must consider all vertices $v \in V$ and test whether $c(v) = i$ and $c'(v) = j$. The same is true for $\mathcal{C}_i \cup \mathcal{C}'_j$. This indicates a complexity of $O(n)$ for each pair of colors i, j. There are k^2 pairs; and, therefore, the complexity of the computation of $Sim(c, c')$ is $O(nk^2)$. In the following paragraph, we will see how to reduce the complexity to $O(n)$.

Note that the set $\mathcal{C}_i \cap \mathcal{C}'_j$ is non-empty only for pairs of colors i, j for which there exists a vertex $v \in V$ such that $i = c(v)$ and $j = c'(v)$. We can use this fact to reduce the complexity of the computation.

We begin by computing $\gamma_i = |\mathcal{C}_i|$ and $\gamma'_i = |\mathcal{C}'_i|$ for all $i = 1 \ldots k$. This can be done in $O(n)$ by first initializing all γ_i to zero and then increasing $\gamma_{c(v)}$ and $\gamma'_{c'(v)}$ by one for each vertex $v \in V$.

Then, compute all non-zero $\Gamma_{i,j} = |\mathcal{C}_i \cap \mathcal{C}'_j|$ as follows. For every $v \in V$, set $\Gamma_{c(v),c'(v)}$ to zero. Then, for each vertex $v \in V$, increase $\Gamma_{c(v),c'(v)}$ by one.

Note that the $\Gamma_{i,j}$ variables take an amount of memory space of $O(k^2)$ if we wish to ensure that we can access each variable in constant time. For the following, we assume that enough memory can be allocated.

We use the following equality:

$$Sim(\mathcal{C}_i, \mathcal{C}'_j) = \frac{\Gamma_{i,j}}{\gamma_i + \gamma'_j - \Gamma_{i,j}} = \sum_{v \in \mathcal{C}_i \cap C'_j} \frac{1}{\gamma_i + \gamma'_j - \Gamma_{i,j}}.$$

Initialize $\sigma = 0$. For each vertex $v \in V$, let $i = c(v)$ and $j = c'(v)$ and set

$$\sigma \leftarrow \sigma + 1/(\gamma_i + \gamma_j - \Gamma_{i,j}).$$

Finally, multiply σ by $\frac{1}{k}$ to find the similarity between c and c'. See Algorithm 2 for a pseudocode of this algorithm.

Similarity of two k-colorings Differing on one Vertex

How similar are two k-colorings that differ only on one vertex? Assume that there is a graph with n vertices and a balanced k-coloring c, i.e. every color class is of size γ. As a consequence, we have $n = \gamma k$. Now, a new coloring c' is obtained from c by recoloring a vertex v of color i with color j. What is the value of $Sim(c, c')$? Note that $Sim(\mathcal{C}_r, \mathcal{C}'_s) = \delta_{r,s}$ for $r, s \notin \{i, j\}$, where $\delta_{r,s}$ is the Kronecker symbol. It equals 1 if $r = s$ and 0 otherwise. From this, we obtain

$$\begin{aligned} Sim(c, c') &= \frac{1}{k} \sum_{\substack{1 \le i \le k \\ 1 \le j \le k}} Sim(\mathcal{C}_i, \mathcal{C}'_j) \\ &= \frac{1}{k} \left(k - 2 + Sim(\mathcal{C}_i, \mathcal{C}'_i) + Sim(\mathcal{C}_i, \mathcal{C}'_j) + Sim(\mathcal{C}_j, \mathcal{C}'_i) + Sim(\mathcal{C}_j, \mathcal{C}'_j) \right) \\ &= \frac{1}{k} \left(k - 2 + \frac{\gamma - 1}{\gamma} + \frac{1}{2\gamma} + \frac{0}{2\gamma} + \frac{\gamma}{\gamma + 1} \right) \\ &= 1 - \frac{1}{k} \cdot \frac{1}{2\gamma} - \frac{1}{k} \cdot \frac{1}{\gamma + 1}; \end{aligned}$$

and, using the fact that $n = \gamma k$, we find

$$\begin{aligned} Sim(c, c') &= 1 - \frac{1}{2n} - \frac{1}{n + k} \\ &\le 1 - \frac{1}{n}. \end{aligned}$$

The last inequality relies on $k \le n$ which is obvious.

Computing the Similarity to a Reference Coloring

In a local search algorithm, it can be useful to compare the current solution to a reference solution at each iteration. Therefore, we need a fast method for computing the similarity between a new solution and the reference solution, based on the similarity between the previous solution and the reference solution.

Suppose we have a reference k-coloring c^r, a k-coloring c^p from the previous iteration, and a neighbor k-coloring c' which differs in exactly one vertex from c^p. Moreover, suppose that we are given $s = Sim(c^r, c^p)$ and the size of all color classes $\gamma_i^r = |\mathcal{C}_i^r|$ and $\gamma_i^p = |\mathcal{C}_i^p|$ for $i = 1 \dots k$. Further, we suppose that we know $\Gamma_{i,j} = |\mathcal{C}_i^r \cap \mathcal{C}_j^p|$ for all $i, j = 1 \dots k$. Using these variables we can compute

$$\frac{|\mathcal{C}_i^r \cap \mathcal{C}_j^p|}{|\mathcal{C}_i^r \cup \mathcal{C}_j^p|} = \frac{\Gamma_{i,j}}{\gamma_i^r + \gamma_j^p - \Gamma_{i,j}}.$$

Let v' be the vertex that changes color, i.e. the unique $v' \in V$ such that $c^p(v') \neq c'(v')$. Let $\alpha = c^p(v')$ be its color in the previous coloring and $\beta = c'(v')$ its color in the new

coloring and $\theta = c^r(v')$ its color in the reference solution.
We are interested in

$$k(Sim(c^r, c') - Sim(c^r, c^p)) = \sum_{\substack{i=1\ldots k \\ j=1\ldots k}} \left(\frac{|\mathcal{C}_i^r \cap \mathcal{C}_j'|}{|\mathcal{C}_i^r \cup \mathcal{C}_j'|} - \frac{|\mathcal{C}_i^r \cap \mathcal{C}_j^p|}{|\mathcal{C}_i^r \cup \mathcal{C}_j^p|} \right).$$

If $j \notin \{\alpha, \beta\}$, then the terms cancel out because the sets $\mathcal{C}_j^p$ and $\mathcal{C}_j'$ are the same. However, all other terms remain:

$$k(Sim(c^r, c') - Sim(c^r, c^p)) = \sum_{\substack{i=1\ldots k \\ j=\alpha,\beta}} \frac{|\mathcal{C}_i^r \cap \mathcal{C}_j'|}{|\mathcal{C}_i^r \cup \mathcal{C}_j'|} - \frac{|\mathcal{C}_i^r \cap \mathcal{C}_j^p|}{|\mathcal{C}_i^r \cup \mathcal{C}_j^p|}.$$

This formula gives an $O(k)$ time algorithm to update the similarity measure, provided that we can update all values γ and Γ in $O(k)$. In fact, it is possible to update these values in constant time.
The values γ^r do not change. The values γ' for the new solution verify that $\gamma'_j = \gamma_j$ for $j \neq \alpha, \beta$, $\gamma'_\alpha = \gamma_\alpha - 1$ and $\gamma'_\beta = \gamma_\beta + 1$.
The values $\Gamma_{i,j}$ change only for $j = \alpha, \beta$ and $i = \theta$. For other values of i and j the vertex changing color v' is not in one of the intersections $\mathcal{C}_i^r \cap \mathcal{C}_j^p$ or $\mathcal{C}_i^r \cap \mathcal{C}_j'$. Therefore, $\Gamma_{\theta,\alpha}$ needs to be decremented by one, and $\Gamma_{\theta,\beta}$ needs to be incremented by one. See Algorithm 3 for a pseudocode of this algorithm.

2.4 The Foo-scheme

The Foo-*scheme* provides a reactive tabu tenure based on the ***f****luctuation* ***o****f the* ***o****bjective* function. The idea behind this scheme is to observe how the objective function evolves along the search process. If the objective function does not change its value (or changes its value very little) during a long period, we have reason to believe that the search process is trapped in an uninteresting region of the search space. As a reaction, we increase the tabu tenure in an attempt to escape the region where the search process has been trapped.

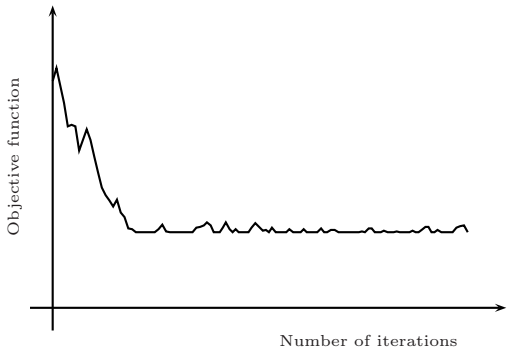

FIGURE 7 When the values of the objective function only cover a small interval over a long period, we must assume that the search process is trapped and diversification is needed.

As a counterweight for increasing the tenure, we make the tenure evaporate slowly along the search process. This makes the search process alternate between intensification (when the tenure is low) and diversification (when the tenure is high).

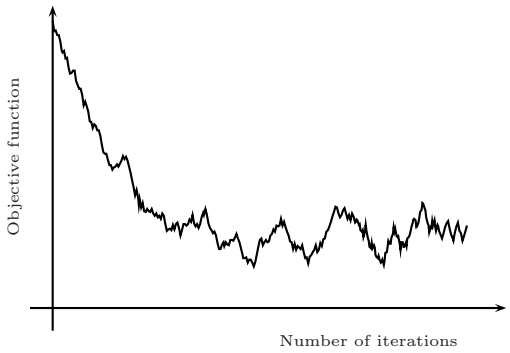

FIGURE 8 If the objective function fluctuates "enough," we slowly decrease the tabu tenure to make the search process more focused.

2.4.1 Implementation

We define three parameters for the FOO-scheme: φ, η and b. Every φ iterations, we will determine Δ, the difference between the maximum and minimum values that the objective function has taken during the last φ iterations. We will refer to φ as a *frequency*, even if this is technically incorrect. (A frequency is the number of events per time, and not the time per event.) If Δ is less than the *threshold* b, we conclude that the search process is probably trapped and increase the tabu tenure t by the *increment* η. Otherwise, if Δ is larger than the threshold b, we decrease t by one. See Algorithm 4 for pseudocode of the FOO-scheme.

FIGURE 9 Illustration of the FOO-scheme with the interval of φ iterations to determine the range Δ of values the objective function has taken over the last φ iterations.

2.4.2 Parameter Tuning

The parameters φ, η and b of the FOO-scheme must be tuned for a specific problem. We will refer to the process of determining good parameter combinations as the *tuning phase*. There are two extreme scenarios to avoid. In the first scenario, the tabu tenure grows beyond all limits. This can happen because the threshold b or the increment η are too large or because φ is too small. In the second scenario, the tabu tenure always stays close to zero and the search process is quickly blocked in an uninteresting region of the search space. This can happen because the threshold b or the increment η are too small or because φ is too large.

2.4.3 Randomizing the FOO-scheme

Instead of fixing the values of φ, η and b, they can be chosen randomly every φ iterations from a uniform distribution over some intervals to be determined. The intervals should be chosen such that they include parameter combinations determined in the tuning phase to be good. Randomizing the parameters can avoid over-training in the tuning phase and can make the scheme more robust.

2.5 The ACD-scheme

We present another reactive scheme, called the ***a**pproximated **c**ycle-**d**etection* scheme. The underlying idea is to detect cycles without storing the previously visited solutions. Instead, one only needs one *reference solution* and a distance or a similarity measure.

After every iteration of the tabu search, the reference solution is compared to the current solution. If they are equal (i.e. at distance zero), the tabu tenure is increased. Otherwise, the tabu tenure slowly evaporates. The main difficulty with this approach is determining how and when to reset the reference solution.

Updating the Reference Solution

Suppose the search process is trapped in some region of the search space. Due to the tabu list, the sequence of visited solutions is erratic. For example, a situation may arise where, even if a solution s is visited several times along the search process, the solutions visited right before s and after s may differ every time we pass through s. The challenge is to find a mechanism that chooses a reference solution which is most likely to be revisited by the search process.

The first observation is that the search process prioritizes solutions with a small objective value (assuming that we are minimizing). Under the assumption that the search process is trapped in a region of the search space, we assume that it is more likely to revisit local minima than other solutions. Therefore, we reset the reference solution to the current solution every time the current solution has a better objective value than the reference solution.

The second observation is that, if a reference solution has not been revisited for a long time, we assume that the search process will never revisit the reference solution again because either the search process has moved to another region or it is trapped in a smaller region that does not contain the reference solution. Therefore, after a number of iterations to be determined, we set the reference solution to the current solution.

The third observation is that once the current solution is very far (in terms of a distance function) from the reference solution, we can assume that the search process has definitely left the region and will not revisit the reference solution again.

2.6 The ETB-scheme

The ETB-scheme's main goal is to provide a mechanism to enforce diversification by means of manipulating the tabu tenure. It requires a reference solution s_{ref} and a distance measure. The initial tabu tenure is set to a small value. The following is repeated.

The tabu tenure slowly increases until the current solution s ***e****scapes* ***t****he* ***b****all* defined by the set of solutions which are at a distance less than or equal to a radius r to be defined. The reference solution s_{ref} is set to the current solution and the tabu tenure is set back to the initial value.

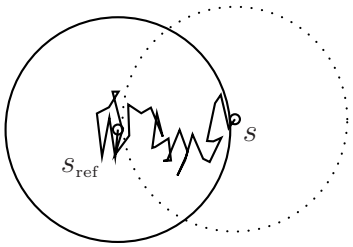

FIGURE 10 An illustration of the ETB-scheme. The tabu tenure grows until the current solution s escapes the ball around s_{ref}. At that moment, the reference solution s_{ref} is initialized to the current solution s and the tabu tenure is set back to a small value.

Preliminary tests with the ETB-scheme for the graph coloring problem have not led to promising results. Nevertheless, when it is used together with other schemes, it provides an interesting diversification procedure with a guarantee on the degree of diversification (measured by the radius r of the ball).

Algorithm 1 Generic tabu search

Input: Maximum number of iterations $i_{\max}$ without improvement, tabu tenure t, objective function f.

Output: The best solution encountered $s^\star$.

generate an initial solution s
$s^\star \leftarrow s$, $f^\star \leftarrow f(s)$
$L \leftarrow \emptyset$, $i \leftarrow 0$ /* *L is the tabu list and i the iteration counter* */
/* *Depending on the problem, the second condition does not make sense* */
while $i \leq i_{\max}$ **and** s is not optimal **do**
  $N'(s) \leftarrow \{s' \in N(s) \mid \text{ move } (s \hookrightarrow s') \notin L \vee f(s') < f^\star\}$
  $s' \leftarrow \arg \min_{s'' \in N'(s)} f(s'')$ /* *Ties are broken randomly* */
  if $f(s') < f^\star$ **then**
    $f^\star \leftarrow f(s')$, $s^\star \leftarrow s'$ /* *update the current best solution and its value* */
    $i \leftarrow 0$
  end if
  $L \leftarrow L \cup (s' \hookrightarrow s)$ /* *update the tabu list L* */
  remove moves from L which have been added more than t iterations ago
  $s \leftarrow s'$ /* *update the current solution s* */
  $i \leftarrow i + 1$
end while
return $s^\star$ of value $f^\star$

n	k	mean	$\frac{k}{2k-1}$	std dev.	min	max
50	5	0.5518	0.5556	0.014	0.51	0.59
50	10	0.5217	0.5263	0.024	0.45	0.57
100	5	0.5552	0.5556	0.007	0.53	0.57
100	10	0.5272	0.5263	0.007	0.50	0.55
100	20	0.5151	0.5128	0.016	0.47	0.55
200	5	0.5554	0.5556	0.004	0.54	0.56
200	10	0.5273	0.5263	0.004	0.51	0.54
200	20	0.5120	0.5128	0.007	0.49	0.53

Table 2.1: Similarities between random colorings. For each set of parameters n (number of vertices) and k (number of colors), 300 pairs of random colorings have been generated, and the similarities between the two have been computed. The statistics over the 300 runs are given. Additionally, we give the first order approximation $\frac{k}{2k-1}$ for the expectation of the similarity.

Algorithm 2 An $O(n)$ algorithm to compute the similarity between a k-coloring c and a k'-coloring c'. $Sim(c, c') = \frac{2}{k+k'} \sum_{\substack{i=1,\dots \\ j=1\dots k'}} \frac{|\mathcal{C}_i \cap \mathcal{C}'_j|}{|\mathcal{C}_i \cup \mathcal{C}'_j|}$

Input: Vertex set V, k-coloring $c : V \to \{1, \dots, k\}$, k'-coloring $c' : V \to \{1, \dots, k'\}$.
Output: $Sim(c, c')$.

1: /* *Variables to compute* $|\mathcal{C}_i|$ *and* $|\mathcal{C}'_j|$ */
2: $\gamma_i \leftarrow 0$ for all $i = 1 \dots k$
3: $\gamma'_j \leftarrow 0$ for all $j = 1 \dots k'$
4: /* *Compute* $|\mathcal{C}_i|$ *and* $|\mathcal{C}'_j|$ */
5: **for all** $v \in V$ **do**
6: $\gamma_{c(v)} \leftarrow \gamma_{c(v)} + 1$
7: $\gamma'_{c'(v)} \leftarrow \gamma'_{c'(v)} + 1$
8: **end for**
9: **for all** $v \in V$ **do**
10: $\Gamma_{c(v),c'(v)} \leftarrow 0$
11: **end for**
12: /* *Compute* $|\mathcal{C}_i \cap \mathcal{C}'_j|$ */
13: **for all** $v \in V$ **do**
14: $\Gamma_{c(v),c'(v)} \leftarrow \Gamma_{c(v),c'(v)} + 1$
15: **end for**
16: $\sigma \leftarrow 0$
17: **for all** $v \in V$ **do**
18: $i \leftarrow c(v)$, $j \leftarrow c'(v)$
19: $\sigma \leftarrow \sigma + \frac{1}{\gamma_i + \gamma_j - \Gamma_{i,j}}$
20: **end for**
21: $\sigma \leftarrow \sigma \frac{2}{k+k'}$
22: **return** σ /* *return* $Sim(c, c')$ */

Algorithm 3 An $O(k)$ algorithm to recompute the similarity between a reference coloring c^r and a coloring c to be updated by recoloring a given vertex.

Input: Vertex set V, two k-colorings $c^r = \{\mathcal{C}_i^r\}_{i=1,\ldots,k}$ (reference coloring) and $c = \{\mathcal{C}_j\}_{j=1,\ldots,k}$ (coloring to be updated), a vertex v' to be recolored with color β, $\gamma_i^r = |\mathcal{C}_i^r|$, $i = 1,\ldots,k$ and $\gamma_j = |\mathcal{C}_j|$, $j = 1,\ldots,k$, and $\Gamma_{i,j} = |\mathcal{C}_i^r \cap \mathcal{C}_j|$, $i,j = 1,\ldots,k$, $s = \mathit{Sim}(c^r, c)$.

Output: Modified coloring c', updated values for γ_j and $\Gamma_{i,j}$ (for colorings c^r and c'), $s = \mathit{Sim}(c^r, c')$.

1: $\alpha \leftarrow c(v')$
2: $\theta \leftarrow c^r(v')$
3: **for all** $i = 1,\ldots,k$ **do**
4: $\quad s \leftarrow s - \frac{\Gamma_{i,\alpha}}{\gamma_i^r+\gamma_\alpha-\Gamma_{i,\alpha}} - \frac{\Gamma_{i,\beta}}{\gamma_i^r+\gamma_\beta-\Gamma_{i,\beta}}$
5: **end for**
6: $\Gamma_{\theta,\alpha} \leftarrow \Gamma_{\theta,\alpha} - 1$
7: $\Gamma_{\theta,\beta} \leftarrow \Gamma_{\theta,\beta} + 1$
8: $\gamma_\alpha \leftarrow \gamma_\alpha - 1$
9: $\gamma_\beta \leftarrow \gamma_\beta + 1$
10: $c' \leftarrow c$
11: $c'(v') = \beta$
12: **for all** $i = 1,\ldots,k$ **do**
13: $\quad s \leftarrow s + \frac{\Gamma_{i,\alpha}}{\gamma_i^r+\gamma_\alpha-\Gamma_{i,\alpha}} + \frac{\Gamma_{i,\beta}}{\gamma_i^r+\gamma_\beta-\Gamma_{i,\beta}}$
14: **end for**
15: **return** s /* return $\mathit{Sim}(c^r, c')$ */

Algorithm 4 The Foo-scheme to adjust in a reactive manner the tabu tenure.

Input: Tabu tenure t, parameters η, b, difference Δ of the maximum and the minimum values of the objective function over the last φ iterations.

Output: An adjusted tabu tenure t.

1: **if** $\Delta \leq b$ **then**
2: $\quad t \leftarrow t + \eta$
3: **else**
4: $\quad$ **if** $t > 0$ **then**
5: $\quad\quad t \leftarrow t - 1$
6: $\quad$ **end if**
7: **end if**
8: **return** t

Algorithm 5 The ACD-scheme to adjust the tabu tenure in a reactive manner.

Input: Real valued tabu tenure t, current solution s_{cur} and reference solution s_{ref}, objective function f, maximal allowed age $t_{\max}$ of s_{ref}, distance function d on the search space, distance threshold $d_{\max}$, rate of evaporation ν.

Output: An adjusted tabu tenure t and an updated s_{ref}.

1: **if** $d(s_{\text{cur}}, s_{\text{ref}}) = 0$ **then**
2: $\quad t \leftarrow t + \eta$
3: **end if**
4: $t \leftarrow \nu t$
5: **if** $f(s_{\text{cur}}) < f(s_{\text{ref}})$ **or** s_{ref} is older than $t_{\max}$ **or** $d(s_{\text{cur}}, s_{\text{ref}}) > d_{\max}$ **then**
6: $\quad s_{\text{ref}} \leftarrow s_{\text{cur}}$
7: **end if**

Chapter 3

The PARTIALCOL Heuristic

3.1 Introduction

The graph coloring problem has many practical applications such as the creation of time tables, frequency assignments, and stock management [Cha82, GR92, Lei79, Zuf02]. As a result of its importance and simplicity, many methods have been developed to solve or to approach the graph coloring problem. Elaborated exact methods are able to find optimal colorings for graphs with up to about 100 vertices [Her03, JAMS91]. This limit is, of course, a rule of thumb. There are intractable instances that have fewer than 100 vertices and graphs that can be solved optimally that have more than 150 vertices.

For larger graphs, we must use heuristics. Strategies range from very simple greedy approaches to elaborate hybrid evolutive heuristics. A set of well known benchmark graphs can be found on the web [dim]. We have tested our approach on a sample of these benchmark graphs, and we have compared our results to similar methods, as well as to the two genetic hybrid algorithms GH [GH99] and MMT [MMT05], which rank among the most efficient graph coloring algorithms.

3.2 Various Approaches to Vertex Coloring

There are two widely explored approaches to vertex coloring. The first consists of producing k-colorings and attempting to decrease k while maintaining a (valid) coloring. Most constructive and greedy heuristics make use of this approach. These heuristics are generally very fast but yield solutions which are far from optimal.

The second approach consists of fixing k and starting with an *improper* k-coloring. An *improper* coloring may contain *conflicts*. If two adjacent vertices x and y have the same color, we say that there is a *conflict* between vertices x and y, respectively vertices x and y are *conflicting vertices*, and the edge $\{x, y\}$ is a *conflicting edge*. One may try then to reduce the number of conflicts until, in the best case, a k-coloring is found. If a k-coloring is found, the method is restarted in order to search for a $(k-1)$-coloring, and so on. If no k-coloring is found, we restart the method to search for a $(k+1)$-coloring, and so on. The process is stopped as soon as a fixed k is considered twice by the method. The output solution is a k-coloring with the smallest k. Most state-of-the-art heuristics are derived

from this approach.

A third and barely explored approach uses *partial k-colorings* with a fixed k and a strategy for adapting k derived from the second approach described above. A *partial k-coloring* consists of k mutually disjoint stable sets $S_1, \ldots, S_k$ and a set $\mathcal{O}$ of non-colored vertices such that $\mathcal{O} = V \setminus (S_1 \cup \ldots \cup S_k)$. Note that a partial k-coloring has no conflict. The goal of this third approach is to increase the size of the partial solution, which is equivalent to reducing the size of $\mathcal{O}$. Our method, PARTIALCOL, is based on this third approach. Morgenstern proposed a complex algorithm based on partial solutions in [Mor96]. More details on his algorithm, henceforth called MOR, will be given in Section 3.3.3.

Some practical frequency assignment problems [AvHK+03], for which we know the set of all the interference constraints, could be formulated as graph coloring problems. In this case, a frequency (instead of a color) must be assigned to each antenna (instead of a vertex) while respecting a subset of the set of the interference constraints. For example, there could be an interference between antennae x and y if they transmit on the same frequency. In the *minimum interference frequency assignment problem*, the goal is to minimize the sum of the interferences. In this case, all the antennae must receive a frequency, and any interferences are penalized. To solve this, the approach using improper colorings could be appropriate. However, if the goal is to assign a frequency to as many antennae as possible while respecting all the interference constraints (and not only a subset), then the partial solution approach is useful. This problem is known as the *maximum service frequency assignment problem*.

3.2.1 Constructive Heuristics

Constructive heuristics build a solution in a step-by-step fashion. In each step, a vertex is chosen and a color is assigned to it such that no conflict is generated. There exist numerous strategies for determining the order in which the vertices are chosen. The most simple approach is the *greedy algorithm*, which chooses vertices randomly and assigns the smallest possible color (supposing the colors are numbered) to the examined vertex. A more successful method called DSATUR [Bré79] consists of always choosing the vertex with the highest number of differently colored adjacent vertices; the vertex that has the highest number of incident edges is used in the event of a tie. Other published methods of this type are RLF [Lei79], KP [Zuf02], and the ITERATED GREEDY coloring algorithm [Cul92]. These methods find solutions quickly, but these solutions are often far from optimal. Note that the ITERATED GREEDY algorithm is often combined with local search procedures in order to obtain better results.

3.2.2 Local Search Heuristics

Local search heuristics operate in a *search space* S, also called a *solution space*. The elements of this space are called *solutions* even if not all elements are solutions to the initial problem. For every solution $s \in S$, a neighborhood $N(s) \subset S$ is defined. A local search method starts at an initial solution and then moves from a solution to a neighbor solution and tries to find "better" solutions, measured by an appropriate objective function.

The passage from one solution to the next is called a *move*. Most graph coloring heuristics use one of the two following search spaces: (1) a space of all colorings, where the heuristic simply tries to reduce the number of colors used, or (2) a space with improper colorings but a fixed number of colors, where the heuristic attempts to reduce the number of conflicts to zero.

The most-studied search strategies are simulated annealing [CHdW87, JAMS91], tabu search (called TABUCOL) [HdW87], and variable neighborhood search [AHZ03, Zuf02]. Today, one of the most efficient variants of TABUCOL can be briefly described as follows [GH99]: the search space is the set of k-partitions; and the objective function f, which is to be minimized, is the total number of conflicting edges. A neighbor solution is obtained by modifying the color of a conflicting vertex. When the color of a vertex x is modified from i to j, color i is declared tabu for vertex x for a certain number t of iterations, and all solutions where x has color i are called *tabu solutions*. The parameter t is called *tabu tenure* or, historically, *tabu list length*.

At each iteration, TABUCOL determines the best neighbor s' of the current solution s (ties between solutions of equal quality are broken randomly) such that either s' is a non-tabu solution or $f(s') < f(s^\star)$, where $s^\star$ is the best solution found so far. The tabu tenure is set equal to $\text{UNIFORM}(0, 9) + \alpha n_c$, where $\text{UNIFORM}(a, b)$ returns an integer randomly chosen in $\{a, \ldots, b\}$, $\alpha = 0.6$, and n_c represents the number of conflicting vertices in the current solution s. If the tabu tenure depends on the current solution, we say that it is *dynamic*. More details and results on these methods can be found in [GH99, HGZ05, Zuf02]. Note that Dorne and Hao proposed a comparable tabu search heuristic for graph coloring in [DH98]. In their method, they use a candidate list strategy, an aspiration criterion, incremental evaluation, and dynamic tabu tenure as described above, but with $\alpha \in \{2, 4\}$. Such a method does not outperform the one proposed by Galinier and Hao in [GH99] and, consequently, we will not compare our algorithms to this method.

3.2.3 Evolutive Hybrid Heuristics

Evolutive hybrid heuristics work with a population P of several solutions (or pieces of solutions) at a time and attempt to produce better solutions by recombining solutions (or pieces of solutions) of P. When new solutions are produced, the algorithm applies a local search procedure in an effort to improve these solutions. The solutions obtained are finally used to update P. In such methods, almost all CPU time is consumed by the local search procedure. The most successful coloring methods are of this type and usually use a variant of TABUCOL as a local search procedure. The choice of TABUCOL is often preferred to other local search methods because it is the most simple, rapid, and efficient algorithm among the best-known local search coloring procedures.

We would like to draw attention to several genetic hybrid algorithms [DH98, FF96, GH99, MMT05], a scatter search method [HH02a], and an adaptive memory heuristic [HGZ05, Zuf02]. The last method is the most simple method among the methods of this type. It is close to the one proposed in [GH99]. In both of these cases, an appropriate recombination operator and an efficient and quick local search procedure (TABUCOL) are responsible for the excellent performance exhibited by those methods. The very recent MMT algorithm [MMT05] shows an excellent performance and has improved the best

known coloring on several graphs. It is based on the same neighborhood structure as the PARTIALCOL algorithm to be presented in this chapter. It is worth noting that the recombination operator and the variant of local search method used internally must be adapted specifically to the given problem for optimum results.

3.3 The PARTIALCOL Algorithm

In this section, we describe an alternative tabu search method for the graph coloring problem. We first describe the solution space, its associated neighborhood structure, and alternative neighborhood structures. We then present a method of choosing a neighbor solution. Finally, we propose three ways of managing the tabu tenure.

PARTIALCOL is a local search method as described in Section 3.2.2. The method takes as input a graph $G = (V, E)$ with vertex set V and edge set E, along with the desired number k of colors to find a k-coloring.

3.3.1 Definition of the Search Space

The search space consists of *partial k-colorings*. A *partial k-coloring* consists of k disjoint stable sets (mutually non-adjacent vertices) $S_1, \ldots, S_k$ and a set $\mathcal{O} = V \setminus \bigcup_{i=1}^{k} S_i$. We note $s = (S_1, \ldots, S_k; \mathcal{O})$. Each vertex $v \in V$ is in exactly one of the sets $S_1, \ldots, S_k, \mathcal{O}$. If $v \in S_i$, then v has color i. If $v \in \mathcal{O}$, then v is not colored.

We can see that there is a major difference between the structure of a solution in TABUCOL and the one used for PARTIALCOL. In the approach chosen for PARTIALCOL, all the conflicts are in the set $\mathcal{O}$; however, in TABUCOL, the conflicts could be in any of the k color classes. Consequently, the solution spaces induced by these two solution structures are very different.

Note that the idea to deal with partial assignment is not new. For example, Morgenstern proposed a complex method based on partial k-colorings in [Mor96] for the graph coloring problem (see Section 3.3.3 for more details). In addition, Vasquez proposed in [Vas02] a tabu search for the frequency assignment problem with polarizations. He worked with partial assignments too. Because these two algorithms both perform well, one of our motivations is to propose a method based on partial assignments, but which is much simpler than the one proposed by Morgenstern.

3.3.2 Strategy and Generation of an Initial Solution

To begin we have to determine the first tested k for which we will try to find a k-coloring. In our algorithms, we chose $k = \chi(G)$ if $\chi(G)$ is known, and $k = k^\star$ otherwise, where $k^\star$ is the smallest k for which a k-coloring has been found by a known heuristic. Two possible cases arise: (1) a k-coloring is found or (2) no k-coloring is found. If a k-coloring has been found, we decrease k by one, start over, and continue until no k-coloring can be found. Otherwise, if no k-coloring could be found, we increase k by one and start over, continuing until a k-coloring has been found.

In order to generate an initial partial k-coloring $s = (S_1, \ldots, S_k; \mathcal{O})$, we use the following greedy algorithm. At each step we randomly choose a non considered vertex v and insert it to S_i with the smallest possible i, such that S_i is still a stable set. If no such set S_i exists, we add v to $\mathcal{O}$.

In order to generate an initial k-partition (or improper k-coloring) $s = (\mathcal{C}_1, \ldots, \mathcal{C}_k)$, we also use a greedy algorithm which is very close to the previous one. At each step, we randomly choose a non-considered vertex v and put it into $\mathcal{C}_i$ with the smallest possible i such that we avoid creating any conflict. If it is impossible to do so without creating a conflict, we put v in a randomly selected class $\mathcal{C}_j$.

Note that the two above greedy algorithms do not consume more than a fraction of second to generate an initial solution.

3.3.3 Neighborhood of a Solution

Let $A(v)$ be the subset of V containing all the vertices adjacent to v. A neighbor solution $s' = (S'_1, \ldots, S'_k; \mathcal{O}')$ can be obtained from a solution $s = (S_1, \ldots, S_k; \mathcal{O})$ by coloring an uncolored vertex $u \in \mathcal{O}$ with a color c. We call this a *move* $(u \hookrightarrow S_c)$. To execute this move, we first set $\mathcal{O}' = \mathcal{O} \setminus \{u\}$ and $S'_c = S_c \cup \{u\}$. It is now possible that S'_c is no longer stable, so we remove vertices adjacent to u in order to make S'_c a stable set. The sets for the new solution s' will be

$$\begin{array}{lcll} \mathcal{O}' & = & (\mathcal{O} \setminus \{u\}) \cup (A(u) \cap S_c) & \\ S'_c & = & (S_c \cup \{u\}) \setminus (A(u) \cap S_c) & \\ S'_j & = & S_j & 1 \leq j \leq k, \quad j \neq c. \end{array}$$

This kind of neighborhood was first proposed by Morgenstern in [Mor96], but his method is much more complicated than the one proposed here. In Morgenstern's method, a neighbor solution is chosen as it is in a simulated annealing search, i.e. a temperature parameter T is required, but this parameter is difficult to tune. Further, Morgenstern's method uses a second neighborhood structure when the first one cannot improve the best solution found after a fixed number of iterations. This other neighborhood structure is complex and deals with *s-chain interchanges* (refer to [Mor96] for a definition). Moreover, Morgenstern's method is based on a pool of good partial k-colorings and not only on a single partial k-coloring. Finally, Morgenstern uses a specific method called XRLF (presented in [JAMS91]) to generate the initial pool of partial solutions.

Note that, with such a neighborhood and such an evaluation function (the number of uncolored vertices), many neighbor solutions have the same value. This property makes the search more random and less guided. It might be interesting to cast a new objective function which discriminates more effectively between different neighbor solutions. However, the danger with such an objective function would be that it might constrict the search and remove too much randomness. As a consequence, the algorithm with this new objective function may perform very well on some graphs with a special structure but poorly on others.

An idea to more effectively discriminate between the neighbor solutions is to use another objective function, as proposed in [Mor96]. Let $d(x)$ be the vertex degree of vertex x and

the objective function be $\sum_{x \in \mathcal{O}} d(x)$ instead of $\sum_{x \in \mathcal{O}} 1 = |\mathcal{O}|$.

Preliminary tests with this objective function have not been conclusive, except for the DIMACS benchmark graphs le450_25c and le450_25d [dim], where colorings with 26 colors were found within seconds. Apparently, the objective function is very well suited for the structure of these two graphs but not for others. Interestingly, with this objective function, the algorithm is no longer capable of finding colorings with 15 colors for the graphs le450_15c and le450_15d but only colorings with 16 colors. With the original objective function, a coloring with 15 colors is found within seconds.

In order to discriminate between the choices of a neighbor solution, we will propose several ways to automatically adjust the tabu tenure (see below).

3.3.4 Alternative Neighborhoods

Our research also involved the investigation of more complex neighborhoods. Instead of simply moving all vertices from $A(u) \cap S_c$ to $\mathcal{O}'$, it may be possible to move some vertices of $A(u) \cap S_c$ to other color classes without creating conflicts.

Our research went further to determine whether, once the vertices of $A(u) \cap S_c$ have been removed from S_c, other vertices from $\mathcal{O}$ could go into S_c without creating conflicts. This neighborhood has been extensively studied in [BGH01] and has proven to be competitive with TABUCOL for graphs of up to 500 vertices.

Both neighborhoods have been tested in our framework, but they are computationally much more expensive than the original neighborhood. The CPU time needed to obtain a neighbor solution grows much faster with the size of the input and graphs with more than 500 vertices become intractable with those neighborhoods.

3.3.5 Choosing the Best Neighbor Solution

PARTIALCOL examines every possible neighbor solution. For each combination of $v \in \mathcal{O}$ and $c \in \{1, \ldots, k\}$, our method evaluates the cardinality $|\mathcal{O}'|$ and chooses the move $(v \hookrightarrow S_c)$ that leads to the smallest $|\mathcal{O}'|$, provided either that the move is not tabu or, if it is tabu, that it leads to a value $|\mathcal{O}'|$ which is smaller than the best value already encountered (denoted $|\mathcal{O}^\star|$). If several moves lead to the same value of $|\mathcal{O}'|$, then one of them is randomly chosen.

Due to the amount of time devoted proportionally to the evaluation of the neighborhood in local search methods, it is imperative that these evaluations can be performed with maximum efficiency, even at the expense of efficiency in other parts of the method.

In our case, we must evaluate, for every vertex $v \in \mathcal{O}$ and every color $c \in \{1, \ldots, k\}$, the number $\gamma_{v,c}$ of vertices adjacent to v with color c. In order to do this efficiently, we store all values $\gamma_{v,c}$ in a matrix Γ of size $n \times k$ (n is the number of vertices). We do this for every vertex and not only vertices in $\mathcal{O}$. Once Γ is computed, the evaluation of a move takes constant time. The evaluation of the complete neighborhood takes a time proportional to $k \cdot |\mathcal{O}|$, where k is the number of colors and $|\mathcal{O}|$ the number of uncolored vertices.

After a move $(v \hookrightarrow S_c)$ has been selected and executed, Γ must be updated as follows.

- For every adjacent vertex u of v, we increase $\gamma_{u,c}$ by one. This update can be done in a time proportional to the number of adjacent vertices of v by storing a list of adjacent vertices for each vertex.

- For every vertex w which is moved back from S_c to $\mathcal{O}$, we decrease the value $\gamma_{u_w,c}$ by one for every neighbor u_w of w. The complexity of this update is small in practice and comparable to the complexity of the first update, which is due to the fact that only very few vertices w will need be removed in general. This characteristic is a result of the fact that our algorithm precisely chooses moves minimizing the number of such vertices w.

3.4 Reactive Tabu Tenure Schemes

For our experiments, we have considered three schemes and used them to govern adjustments to the tabu tenure. The first scheme is a dynamic scheme called the DYN-scheme. The two other are reactive schemes and are named the FOO-scheme and the ACD-scheme (described in Chapter 2). In this section, we discuss the details of how these schemes were adapted to PARTIALCOL and TABUCOL.

3.4.1 The DYN-scheme

We are taking advantage of the fact that finding a k-coloring is really a feasibility problem. This fact allows us to know the optimal value of the objective function, provided that a k-coloring exists. The tabu tenure will be set to a value proportional to the objective function.

Application to TABUCOL

For the TABUCOL algorithm, we have used the following tabu tenure:

$$t = 0.6 \cdot n_c + \text{UNIFORM}(0, 9)$$

where n_c is the number of vertices involved in a conflict and $\text{UNIFORM}(0, 9)$ returns a uniform random integer between 0 and 9 inclusive. This dynamic tenure has been successfully applied to graph coloring in [FF96, GH99]. The algorithm using this scheme will be denoted by DYN-TABUCOL or DYN-TC.

Application to PARTIALCOL

For the PARTIALCOL algorithm, preliminary tests have shown that the same dynamic tabu tenure as for TABUCOL leads to good results. We have replaced n_c (the number of vertices involved in a conflict) with the number of uncolored nodes $|\mathcal{O}|$:

$$t = 0.6 \cdot |\mathcal{O}| + \text{UNIFORM}(0, 9)$$

where UNIFORM$(0, 9)$ returns a uniform random integer between 0 and 9 inclusive. The algorithm using this scheme will be denoted by DYN-PARTIALCOL or DYN-PC.

3.4.2 The FOO-scheme

See Section 2.4 for a detailed description of this reactive scheme. For the FOO-scheme, we have used the same, randomized parameter set for both the PARTIALCOL and TABUCOL algorithms:

Parameter	Range
Frequency ϕ	[500,5000]
Increment η	[5,30]
Threshold b	[1,2]

Every ϕ iterations, the three parameters are drawn uniformly from each range. The algorithms using the FOO-schemes will be denoted by FOO-PARTIALCOL (FOO-PC) and FOO-TABUCOL(FOO-TC).

3.4.3 The ACD-scheme

We have considerably extended the ACD-scheme described in Section 2.5. For both PARTIALCOL and TABUCOL, we have applied the same scheme with the same parameter settings. The corresponding algorithms are denoted by ACD-PARTIALCOL (ACD-PC) and ACD-TABUCOL (ACD-TC).

Implementation

We also use a parameter η, which is the increment used to increase the tabu tenure in case a cycle is detected. η is initialized to the value of 5 and it changes along the search in the following way. When a cycle is detected (i.e. the current solution s_{cur} has a similarity of 1 with respect to the reference solution s_{ref}), η is incremented by 5. Every 15'000 iterations of the tabu search, η is decremented by 1 if it is larger than 5.

Every time a cycle is detected, the tabu tenure t is incremented by η (after η itself has been incremented). The evaporation of t is implemented as follows. Every 5000 iterations, t is decremented by $1 + \lfloor t/20 \rfloor$.

In case the tabu tenure is decremented to zero, it is set equal to a positive value. For TABUCOL the tenure is set equal to the number of conflicts. For PARTIALCOL the tenure is set equal to half the number of uncolored vertices. The motivation behind is this: if no cycle has been detected despite the fact that the tabu tenure is very small, we assume that the search is trapped in a large plateau and that the tenure must be increased.

The reference solution is updated every time at least one of the following conditions holds.

- The current solution s_{cur} has a lower objective function than s_{ref}.
- The similarity $Sim(s_{\text{ref}}, s_{\text{cur}})$ is lower than 0.6 (i.e. the solutions are far apart).

- The current solution s_{cur} has the same objective function as the reference solution s_{ref} and more than 1000 iterations have been performed since the last update of s_{ref}.
- The last update of s_{ref} has been performed more than 5000 iterations ago.

3.5 Numerical Results

In this section, we present numerical results of six algorithms for comparison. We have chosen a set of well known, difficult instances, as described below. A first series of exhaustive tests were made with a limit of 10 minutes of CPU-time per run (on a 2GHz Pentium 4 with 512MB of RAM).

Compared to other graph coloring experiments by other researchers, a limit of 10 minutes of CPU-time is very short. However, the imposition of this limit allowed us to run 50 tests for every graph and every tested number of colors k. All tests together resulted in over 23'000 runs. In order to test so many runs, we had no option but were compelled to execute the different tests in parallel on 30 identical machines.

Based on the results of the first series of tests, four of the six algorithms were selected to be run for 60 minutes CPU-time, again with 50 runs for each tested k. We had also slightly reduced the set of tested instances by removing graphs which were optimally colored by all six algorithms.

3.5.1 Instances

We have chosen a sample of 23 graphs, of which most of them are well known as they were given at the COLORING02 workshop [dim]. These graphs encompass all large and difficult graphs used for the famous DIMACS benchmarks [JT96]. Other graphs in this collection of benchmark instances are very easy to solve, in the sense that a sophisticated greedy algorithm or a short application of a very basic tabu search finds a solution in a very short time. See [HGZ05] for details.

We have added a graph called "U_13_3-v", which is the universal graph $U(13, 3)$ with one vertex removed. This graph is involved in Conjecture 40 stating that $U(13, 3)$ is 5-chromatic and critical. The chosen set of instances contains graphs of different types.

- There are *random graphs* [JAMS91], named "DSJC$n.d$", where n is the number of vertices and d is 10 times the density. They are generated in such a way that the probability of an edge being present between two given vertices equals the density.
- There are *geometric random graphs* [JAMS91], named "DSJR$n.r$" and "R$n.r$". They are generated by choosing randomly n points in the unit square, which will be the vertices of the graph, and by joining two vertices by an edge, if the two corresponding points are at a distance less than r from each other. Instances with a letter "c" appended to their name are simply the complement of an instance without the "c".
- There are *flat graphs* [Cul92] named "flatn_k_δ" where n is the number of vertices, k is the chromatic number and δ is a *flatness parameter* giving the maximal allowed difference between the degrees of two vertices.

- The *Leighton graphs* are generated to have a given chromatic number using a sophisticated randomized algorithm [Lei79]. The instances we use have been generated by C. Morgenstern and they are named "le$n$_$k$$x$" where n is the number of vertices, k the chromatic number and x is a lower case letter to distinguish different graphs with similar parameter settings.

The graphs that we have chosen for our sample are known to be difficult to color, and most of them have been studied by other researchers [Mor96, FF96, DH98, GH99, HH02a, Zuf02, HGZ05].

3.5.2 Tests with a CPU Time Limit of 10 Minutes

For the first series, we allotted 10 minutes of CPU-time to each of the following six algorithms: DYN-PARTIALCOL, FOO-PARTIALCOL, ACD-PARTIALCOL, DYN-TABUCOL, FOO-TABUCOL and ACD-TABUCOL.

For each graph, each algorithm and for each tested k, we executed 50 runs with a different random seed.

For each instance tested, we report the number $|V|$ of vertices, the chromatic number $\chi(G)$ when known, and the value $k^\star$ which is the smallest k for which a k-coloring has been found by an algorithm (at time of publication). Further, we report the number of colors k for which a k-coloring was found, the number of successful runs out of the 50, the average number of iterations (in thousands) needed to obtain such k-colorings, and the average CPU-time in seconds needed to find the k-colorings.

The results can be found in Tables 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6. A summary and a comparison of the results for all six algorithms is presented in Table 3.7.

If more than one k was tested, we report (at most) the four smallest values of k for which a k-coloring could be found.

3.5.3 Tests with a CPU Time Limit of one Hour

For the second series of tests, we have selected four out of the original six algorithms to be run on 21 out of the original 23 graphs. We have set the CPU-time limit to 60 minutes and, as for the first series, performed 50 runs with different random seeds for every tested algorithm and graph, and every tested k.

From the results of the first series (see Table 3.7), we can see that, for all graphs, except the geometric random graph R250.5, the ACD-scheme is dominated by the FOO-scheme or the DYN-scheme, in the sense that, for every graph (except R250.5), there is at least one algorithm which finds a coloring with as many or fewer colors.

Also, for the two flat graphs, flat300_20_0 and flat300_26_0, an optimal coloring could be found by every algorithm in the first series of tests. Therefore, we elected to remove those graphs from consideration by the second series of tests.

The detailed results for DYN-PARTIALCOL, FOO-PARTIALCOL, DYN-TABUCOL and FOO-TABUCOL are reported in Tables 3.8, 3.9, 3.10, and 3.11. A summary for these results is reported in Table 3.12. Additionally, we report the results of the algorithms GH, MOR, and MMT. However, because the CPU-time required by GH, MOR, and MMT has not

been given, it is not possible to ensure that conditions were equivalent, so any comparison between these algorithms those presented here must be made with care.

We give results of tests made with GH, MOR, and MMT so that the reader may have a baseline by which he may evaluate DYN-PARTIALCOL, FOO-PARTIALCOL, DYN-TABUCOL, and FOO-TABUCOL. GH, MOR, and MMT are widely considered to be among the most efficient graph coloring heuristics available.

3.5.4 Interpretation of the Results

Our results permit the comparison primarily of two issues: (1) the different neighborhoods and (2) the performance characteristics of dynamic and reactive tabu tenures. The tests demonstrate that it is useful to combine the partial solution approach with a reactive way of adjusting the tabu tenure, especially on the "flat" graphs.

PARTIALCOL **Versus** TABUCOL

TABUCOL uses improper colorings (but all vertices are colored) and tries to eliminate all conflicts. Its objective function is the number of conflicting edges. PARTIALCOL, on the other hand, uses partial colorings and a set of uncolored vertices. Its objective is to minimize the number of uncolored vertices.

On the random graphs (the DSJC's), TABUCOL performs better than PARTIALCOL. On the "flat" and "Leighton's" graphs, on the other hand, the results are much less predictable. While one algorithm finds a k-coloring very quickly, the other algorithm might even be challenged to find a $(k+1)$-coloring.

On the "Leighton's" graphs le450_25c and le450_25d, TABUCOL finds colorings with 26 colors quite easily, while PARTIALCOL is only able to find 27-colorings. Surprisingly, this situation is reversed for the graphs le450_15c and le450_15d, where PARTIALCOL finds optimal colorings very quickly, but TABUCOL finds optimal colorings only rarely or not at all. It seems that this is mainly due to the objective function. Preliminary tests have shown that if we use the sum of the degrees of all vertices in $\mathcal{O}$ as the objective function for PARTIALCOL it easily finds a 26-coloring for the le450_25c/d graphs but no 15-coloring for the le450_15c/d graphs. However, such a discriminating objective function does not generally lead to better results on the other graphs. Consequently, it may be an interesting avenue of research to test intermediate objective functions within our approach, such as $\sum x \in O \left\lceil \frac{d(x)}{\beta} \right\rceil$ for some $\beta \geq 1$, where $d(x)$ is the degree of the vertex x. The advantage of this formula is that we can continuously shift β from 1 to the maximal degree Δ of the graph G to get the two extremes. For $\beta = \Delta$, we obtain simply $|\mathcal{O}|$; and, for $\beta = 1$, we obtain the sum of the degrees of the vertices in $\mathcal{O}$.

On the "flat" graphs, the partial solution neighborhood seems to be more appropriate. First, the graphs flat300_20_0 and flat300_26_0 are colored optimally much more quickly by PARTIALCOL than by TABUCOL. For the graph flat300_28_0, we even find an optimal coloring using 28 colors! At the time of publication, the best coloring ever discovered by other algorithms required 31 colors.

Impact of the Foo-scheme in the First Series of Tests

In this section, we discuss the impact of the Foo-scheme in the first series of tests (where each test was limited to 10 minutes of cpu-time). For almost all graphs, either the Dyn-scheme or the Foo-scheme outperforms the Acd-scheme. We will, therefore, focus our attention on comparing the Dyn-scheme with the Foo-scheme.

For Partialcol, the Foo-scheme improves the number of colors over the Dyn-scheme on three graphs: DSJR500.1c, R1000.1c and U_13_3-v. On 15 graphs[1], the number of colors for the best coloring found is the same; and, on five graphs, Dyn-Partialcol finds a better coloring, namely on DSJC1000.9, DSJC500.1, DSJR500.5, R1000.5 and flat1000_76_0. On the graphs where both algorithms found the same minimum number of colors, Foo-Partialcol had a higher success rate (the number of successful runs out of the 50 runs) on five graphs[2], and a smaller rate on three graphs[3].

For Tabucol, we can first observe that the Foo-scheme improves the number of colors on three graphs: DSJR500.1c, R250.1c and le450_15c. For 14 graphs[4], the number of colors is the same; and Dyn-Tabucol outperformed Foo-Tabucol on the following six graphs: DSJC1000.1, DSJC1000.9, DSJR500.5, R1000.5, flat1000_50_0 and flat1000_60_0. On the graphs where both algorithms found the same minimum number of colors, Foo-Tabucol had a higher success rate on four graphs[5], and a lower one on six graphs[6]. These tests show that Dyn-Tabucol has a slightly better performance than Foo-Tabucol.

The fact that the results are varied suggests to implement both schemes with a mechanism to alternate between them.

Our Foo-scheme has two big advantages. First, it can improve the performance of a tabu search. Second, the implementation of such a reactive tenure is very simple because it only depends on the value of the objective function (and not on any specific knowledge of the problem). It should be possible to add it to existing tabu search heuristics that are applied to different problems without major modification of existing source code.

Results of Tests Run Under 60 Minutes cpu-time Limit

The conclusions for the first series of tests comparing Partialcol and Tabucol with the Foo-scheme and the Dyn-scheme are more or less confirmed by the second series of tests:

For Partialcol, the Foo-scheme results in better colorings than the Dyn-scheme for two graphs (R1000.1c and U_13_3-v) but needs more colors for six graphs (DSJC1000.1, DSJC500.5, DSJC500.9, DSJR500.5, R1000.5 and R250.5).

For Tabucol, the Foo-scheme leads to better results on four out of 21 test cases (R250.1c, flat1000_76_0, flat300_28_0 and le450_15c). The results for the graph flat300_28_0 are

[1] DSJC1000.1, DSJC1000.5, DSJC500.5, DSJC500.9, R250.1c, R250.5, flat1000_50_0, flat1000_60_0, flat300_20_0, flat300_26_0, flat300_28_0, le450_15c, le450_15d, le450_25c, le450_25d

[2] DSJC1000.5, R250.1c, R250.5, flat300_28_0, le450_15d

[3] DSJC500.5, DSJC500.9, flat1000_60_0

[4] DSJC1000.5, DSJC500.1, DSJC500.5, DSJC500.9, R1000.1c, R250.5, U_13_3-v, flat1000_76_0, flat300_20_0, flat300_26_0, flat300_28_0, le450_15d, le450_25c, le450_25d

[5] R1000.1c, flat300_26_0, le450_15d, le450_25c

[6] DSJC500.1, DSJC500.5, DSJC500.9, R250.5, flat1000_76_0, flat300_28_0

particularly remarkable as its best known coloring is reduced from 31 to 29 colors. This coloring is still not optimal, but the improvement clearly shows the impact of the FOO-scheme.

On the other hand the FOO-scheme is outperformed by the DYN-scheme on six graphs (DSJC1000.1, DSJR500.5, R1000.5, U_13_3-v, flat1000_50_0 and flat1000_60_0).

Comparison with Other Algorithms

In Table 3.12, we have included the best colorings found by the GH, the MOR and the MMT algorithm, if available, for comparison.

It is interesting to compare PARTIALCOL with MOR [Mor96], which uses the same solution structure, i.e. partial k-colorings. Note that the MOR-algorithm is a simulated annealing algorithm with sophisticated management of the temperature and alternative neighborhoods to escape local minima. Moreover, MOR uses a pool of solutions instead of only one. Nevertheless, we can observe that the results are similar. While the two methods find the same results on nine graphs[7], MOR is better on seven graphs[8] and the PARTIALCOL algorithms finds better colorings for four graphs[9].

It is interesting to compare PARTIALCOL with GH [GH99] and MMT [MMT05] which are arguably among the most efficient coloring methods. GH is based on TABUCOL and MMT is based on PARTIALCOL. Unfortunately, not all benchmarks are given given by [GH99] in their presentation of GH.

Table 3.12 shows that the only graph on which PARTIALCOL outperforms every other method is the graph flat300_28_0. Of the 14 graphs for which we have results from GH, PARTIALCOL find colorings with the same number of colors in six cases and a smaller number of colors in seven cases. When we consider the graphs for which we have data concerning MOR's performance, we see that the algorithms perform equally in nine cases, while PARTIALCOL outperforms MOR in four cases, and MOR outperforms PARTIALCOL in seven cases. PARTIALCOL's performance equals that of MMT in ten cases, but MMT outperforms PARTIALCOL in nine cases.

However, one should consider that PARTIALCOL is only a simple local search method and that the three methods GH, MOR, and MMT are much more complex. Further, for large and difficult graphs, GH, MOR, and MMT required more CPU-time than the 60 minutes to which PARTIALCOL was limited for our experiments, so the comparison has to be done with care. To conclude this section, we put forth that PARTIALCOL should be considered one of the best local search coloring heuristics developed to date.

[7] DSJC500.1, DSJC500.5, DSJR500.1c, R1000.1c, R250.1c, flat1000_50_0, flat1000_60_0, le450_15c, le450_15d

[8] DSJC1000.5, DSJC1000.9, DSJR500.5, R1000.5, R250.5, le450_25c, le450_25d

[9] DSJC1000.1, DSJC500.9, flat1000_76_0, flat300_28_0

3.6 Supporting a Conjecture

In practice, tabu-type algorithms have been found extremely efficient when applied to the problem of coloring graphs with fewer than 200 vertices. We believe that, for such graphs, an algorithm like Foo-Partialcol very often finds a k-coloring within a few seconds if it exists. It can, therefore, be used to estimate the chromatic number of a graph very quickly with high accuracy [HH02b].

This is especially useful if one needs to verify the existence of a k-coloring where k is given and one strongly believes that it exists.

3.6.1 Problem Definition

The conjecture we will support concerns edge coloring. To discuss this conjecture, we will use the notion of line graphs and edge colorings, as defined in Definitions 49 and 50 in Section 1.7.3.

We recall that an edge coloring c_e of G translates straight forward into a vertex coloring of $L(G)$; and, in the same way, a vertex coloring c of $L(G)$ translates into an edge coloring of G.

Therefore, the edge coloring problem on G is equivalent to the vertex coloring problem on $L(G)$, and one can use Foo-Partialcol to find an edge coloring of G by applying it to $L(G)$.

It is clear that, to color the edges of a graph G, one requires at least as many colors as there are edges incident to any vertex. The number of edges incident to a vertex is called its *degree*. The maximum degree of a graph G is denoted by $\Delta(G)$. So, clearly, $\chi'(G) \geq \Delta(G)$.

The following property is a special case of Vizing's theorem formulated in 1964.

Property 53

The edges of a graph $G = (V, E)$ can be colored with at most $\Delta(G) + 1$ colors.

Definition 54

A graph G is *regular* (or *k-regular*), if all vertices have the same degree (k).

We now have the ingredients to state the conjecture originally put forth by Chetwynd and Hilton [CH89]

Conjecture 55

Let G be a regular graph of (even) order $2n$ and degree

$$\Delta(G) \geq 2 \left\lfloor \frac{n+1}{2} \right\rfloor - 1 = \begin{cases} n & \text{if } n \text{ is odd} \\ n-1 & \text{if } n \text{ is even} \end{cases}$$

then $\chi'(G)$ equals $\Delta(G)$.

This conjecture has been proven for $n \leq 5$. For greater values of n, the best theoretical bounds are larger than that given in the conjecture. See [Car04] for more details.

3.6.2 Verification

In order to generate all k-regular graphs on n vertices, we used the very efficient program *genreg* [Mer96, Mer99]. For each generated graph, we computed its line graph and stored it in a file that the FOO-PARTIALCOL algorithm could read. We attempted to find a k-coloring of the vertices of the line graph. If successful, we have a proof that the tested graph is of class 1. Otherwise, we might have found a counterexample to the conjecture and further investigation is needed, such as either a formal proof or an exact algorithm confirming the nonexistence of a k-coloring.

As *genreg* is an enumeration algorithm, we can expect that graphs generated consecutively share similar structures. We might therefore be able to reuse a coloring obtained as initial solution for the following graph. Experiments show that fewer iterations are required to find a solution when reusing the coloring of the previous graph (with conflicting vertices removed) as opposed to generating a randomized greedy coloring for an initial solution.

It happens often enough to merit mentioning that the partial coloring from the previous coloring can be greedily completed by the routine generating an initial coloring such that no local search iterations are necessary.

3.6.3 Results

We verified the Conjecture 55 for $n = 6$ and $n = 7$. That is to say that we found a 5-coloring for the line graphs of all 5-regular graphs on 12 vertices and a 7-coloring for all 7-regular graphs on 14 vertices.

In Table 3.13, we report the number of graphs, the total CPU time consumed for the verification, the number of graphs tested per second, and the mean and the standard deviation of the number of iterations needed by the FOO-PARTIALCOL algorithm to determine a k-coloring.

In Table 3.14, we report the same data again but for the variant of the method whereby we reuse the previously obtained coloring. There is a notable improvement in performance of roughly 12%.

The coloring of these graphs is easy for FOO-PARTIALCOL, but it is not trivial, as shown by the experiments made on the 5-regular graph on 12 vertices using the DSATUR algorithm [Bré79], one of the most efficient greedy type algorithms. Only 386 of the total 7848

graphs could be colored with an optimal number of 5 colors, while 4917 graphs required 6 colors, 2506 graphs required 7 colors, and 39 graphs required 8 colors.

3.7 Conclusion and Further Work

In this chapter, we have presented Foo-Partialcol, a local search approach to the graph coloring problem. We have shown that this heuristic obtains competitive results (in comparison with other local search coloring methods) on a large sample of benchmark graphs which are generally agreed to be difficult to color. In contrast to several coloring algorithms, Partialcol is very simple and efficient.

Contemporary state-of-the-art coloring methods are hybrid evolutive algorithms using Tabucol as the internal local search procedure. Partialcol now offers a powerful alternative to Tabucol and its variations. Recent advances [MMT05] show the potential of this approach.

The Foo-scheme can improve the performances of both Tabucol and Partialcol. The reactive tabu tenure adjusts itself depending not only on the graph but also on the state of the search. Moreover, the proposed reactive tabu tenure is very easy to implement and, in contrast to other methods, is does not need to perform an explicit check for the repetition of configurations. Determination of the tabu tenure requires only the variation of the objective function. Hence, in contrast to the dynamic tabu tenure strategies proposed by Galinier and Hao in [GH99] and Dorne and Hao in [DH98], Foo-scheme's nature does not rely on the specificity of the problem. (Recall that we need the number of conflicting vertices in the above dynamic tabu tenure strategy.) Consequently, the use of this new and simple way of adjusting the tabu tenure could easily be tested on other combinatorial problems where a tabu method has been applied. It should improve the results of the method (especially if the tabu status is static) and simplify its use, once the parameters have been tuned.

Finally, we would like to mention that the paradigm of partial solutions is applicable to other combinatorial problems such as timetabling and frequency assignments.

Algorithm 6 PARTIALCOL: search for a k-coloring for graph G

Input: Graph G, initial solution $s = \{S_1, \ldots, S_k; \mathcal{O}\}$, integers $k \geq 1$, $i_{\max} \geq 0$, a scheme to adjust the tabu tenure.

Output: A k-coloring of G or a message that no k-coloring has been found.

$i \leftarrow 0$ /* *iteration counter* */
initialize the tabu tenure t according to the chosen scheme
$s^\star \leftarrow s$ /* *best solution encountered* */
$|\mathcal{O}^\star| \leftarrow |\mathcal{O}|$
while $i \leq i_{\max}$ **and** $\mathcal{O} \neq \emptyset$ **do**
  compute all moves $M \leftarrow \{(v \hookrightarrow S_q) \mid v \in \mathcal{O}, 1 \leq q \leq k\}$
  remove tabu moves from M which do not lead to a new best solution
  choose a move $(u \hookrightarrow S_q) \in M$ minimizing $|\mathcal{O}'|$, break ties randomly
  apply move $(u \hookrightarrow S_q)$ to s
  determine the tenure t according to the chosen scheme
  set the move $(w \hookrightarrow S_q)$ tabu for the next t iterations
  $i \leftarrow i + 1$
  if $|\mathcal{O}| < |\mathcal{O}^\star|$ **then**
    $s^\star \leftarrow s,\ |\mathcal{O}^\star| \leftarrow |\mathcal{O}|$
    $i = 0$
  end if
end while
if $\mathcal{O} = \emptyset$ **then**
  return k-coloring s
end if
return no k-coloring is found

Results for DYN-PARTIALCOL with a CPU time limit of 10 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	2	277.7
DSJC1000.5	1000	?,83	90	13 of 50	396	7284.4
			91	49 of 50	211	4253.5
DSJC1000.9	1000	?,224	229	1 of 50	537	3104.9
			230	9 of 50	523	2862.7
DSJC500.1	500	?,12	12	39 of 50	122	24812.3
			13	50 of 50	0	16.3
DSJC500.5	500	?,48	50	43 of 50	203	15667.5
			51	50 of 50	16	1281.8
DSJC500.9	500	?,126	128	43 of 50	281	7849.9
			129	50 of 50	42	1194
DSJR500.1c	500	?,85	86	10 of 50	158	9377
			87	14 of 50	209	13420.1
DSJR500.5	500	?,122	127	30 of 50	204	10960.2
			128	45 of 50	122	6461.9
			129	49 of 50	38	1987
R1000.1c	1000	?,98	99	4 of 50	86	1263.6
R1000.5	1000	?,234	251	4 of 50	471	5281.8
			252	23 of 50	402	4729.3
			253	37 of 50	320	3564
R250.1c	250	?,64	64	3 of 50	10	1560.9
			65	9 of 50	26	3452.7
			66	15 of 50	39	5414
R250.5	250	?,65	67	36 of 50	81	10355.4
			68	48 of 50	21	2900.2
U_13_3-v	857	4,4	5	50 of 50	0	1.7
flat1000_50_0	1000	50,50	50	50 of 50	27	107.9
			51	50 of 50	26	105.2
			52	50 of 50	23	102.9
			53	50 of 50	22	101.4
flat1000_60_0	1000	60,60	60	50 of 50	98	390.4
			61	50 of 50	101	401.4
			62	50 of 50	92	410.4
			63	50 of 50	93	416.7
flat1000_76_0	1000	76,83	88	1 of 50	574	9525.3
			89	14 of 50	395	7611.7
			90	48 of 50	187	3833.1
flat300_20_0	300	20,20	20	50 of 50	0	1.4
flat300_26_0	300	26,26	26	50 of 50	0	43.5
flat300_28_0	300	28,31	28	2 of 50	224	19253.8
			29	8 of 50	313	29188.1
			30	16 of 50	334	35438.8
			31	26 of 50	261	31482.9
le450_15c	450	15,15	15	50 of 50	3	615.7
			16	50 of 50	7	1617.4
le450_15d	450	15,15	15	49 of 50	13	2793.1
			16	50 of 50	7	1713.6
le450_25c	450	25,25	27	50 of 50	10	1583.3
le450_25d	450	25,25	27	50 of 50	7	1151.4

Table 3.1: Results for DYN-PARTIALCOL with a CPU time limit of 10 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for FOO-PARTIALCOL with a CPU time limit of 10 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	4	583.6
DSJC1000.5	1000	?,83	90	18 of 50	380	6971.4
			91	50 of 50	181	3534.6
DSJC1000.9	1000	?,224	230	5 of 50	529	3034.5
DSJC500.1	500	?,12	13	50 of 50	0	23.8
DSJC500.5	500	?,48	50	29 of 50	268	20426.6
			51	50 of 50	15	1222.1
DSJC500.9	500	?,126	128	10 of 50	310	7697.8
			129	50 of 50	74	2008.9
DSJR500.1c	500	?,85	85	35 of 50	194	9522.1
			86	50 of 50	57	2880.6
			87	50 of 50	16	865.1
DSJR500.5	500	?,122	128	4 of 50	394	12270
			129	49 of 50	174	5758.9
R1000.1c	1000	?,98	98	1 of 50	97	1354.9
			99	6 of 50	337	5035.6
R1000.5	1000	?,234	253	5 of 50	409	3491
R250.1c	250	?,64	64	50 of 50	4	453.9
			65	50 of 50	0	20.4
			66	50 of 50	0	8
R250.5	250	?,65	67	13 of 50	254	24346.1
			68	50 of 50	24	2500.7
U_13_3-v	857	4,4	4	5 of 50	152	62897.4
			5	50 of 50	0	4.3
flat1000_50_0	1000	50,50	50	50 of 50	109	528.6
			51	50 of 50	118	573.6
			52	50 of 50	101	553.4
			53	50 of 50	101	604.5
flat1000_60_0	1000	60,60	60	33 of 50	443	2505.6
			61	33 of 50	463	2566.3
			62	26 of 50	468	2704.7
			63	36 of 50	472	2787.7
flat1000_76_0	1000	76,83	89	18 of 50	370	6916.5
			90	49 of 50	168	3335.6
flat300_20_0	300	20,20	20	50 of 50	0	5.8
flat300_26_0	300	26,26	26	50 of 50	1	91.4
flat300_28_0	300	28,31	28	11 of 50	355	32865.6
			29	4 of 50	302	30955.2
			30	3 of 50	350	39054.8
			31	8 of 50	211	26003.7
le450_15c	450	15,15	15	50 of 50	1	180.3
			16	50 of 50	1	137.5
le450_15d	450	15,15	15	50 of 50	3	509.1
			16	50 of 50	1	179.2
le450_25c	450	25,25	27	50 of 50	4	707.5
le450_25d	450	25,25	27	50 of 50	4	583.6

Table 3.2: Results for FOO-PARTIALCOL with a CPU time limit of 10 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for ACD-PARTIALCOL with a CPU time limit of 10 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	28	870.1
DSJC1000.5	1000	?,83	90	1 of 50	454	4283.2
			91	21 of 50	404	4095.8
DSJC1000.9	1000	?,224	232	1 of 50	506	1549.4
DSJC500.1	500	?,12	13	50 of 50	0	26
DSJC500.5	500	?,48	50	7 of 50	419	14293.6
			51	50 of 50	72	2487.4
DSJC500.9	500	?,126	128	2 of 50	412	6050.7
			129	44 of 50	232	3511
DSJR500.1c	500	?,85	85	4 of 50	407	9382.9
			86	44 of 50	195	4064.3
			87	50 of 50	44	858.4
DSJR500.5	500	?,122	127	2 of 50	282	5946
			128	7 of 50	140	2940.3
			129	18 of 50	134	2858
R1000.1c	1000	?,98	99	1 of 50	166	1394.4
R1000.5	1000	?,234	252	3 of 50	480	3082.5
			253	12 of 50	450	2862.6
R250.1c	250	?,64	64	50 of 50	17	421.5
			65	50 of 50	5	128.3
			66	50 of 50	1	36.7
R250.5	250	?,65	66	1 of 50	42	2625.8
			67	2 of 50	437	26550.4
			68	14 of 50	64	3824.4
U_13_3-v	857	4,4	4	48 of 50	32	1706.1
			5	50 of 50	0	1.5
flat1000_50_0	1000	50,50	50	50 of 50	54	258
			51	50 of 50	56	240.5
			52	50 of 50	58	267.1
			53	50 of 50	49	234.1
flat1000_60_0	1000	60,60	60	48 of 50	352	1672.6
			61	42 of 50	401	1758.5
			62	42 of 50	396	1859.8
			63	41 of 50	442	2106
flat1000_76_0	1000	76,83	90	26 of 50	389	4041.8
flat300_20_0	300	20,20	20	50 of 50	0	13.7
flat300_26_0	300	26,26	26	50 of 50	4	204
flat300_28_0	300	28,31	28	2 of 50	306	16313.3
			29	4 of 50	221	12044.4
			30	7 of 50	382	21162.3
			31	4 of 50	350	17321.6
le450_15c	450	15,15	15	50 of 50	3	137.2
			16	50 of 50	2	121
le450_15d	450	15,15	15	50 of 50	6	339.4
			16	50 of 50	3	172.6
le450_25c	450	25,25	27	50 of 50	58	2837.4
le450_25d	450	25,25	27	50 of 50	42	2240.7

Table 3.3: Results for ACD-PARTIALCOL with a CPU time limit of 10 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for DYN-TABUCOL with a CPU time limit of 10 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	20	1 of 50	264	31798.8
			21	50 of 50	1	161.8
DSJC1000.5	1000	?,83	89	10 of 50	388	5355.9
			90	49 of 50	133	2391.6
			91	50 of 50	40	761.9
DSJC1000.9	1000	?,224	227	2 of 50	470	1441.3
			228	14 of 50	472	1814.4
			229	46 of 50	407	1883.9
			230	50 of 50	317	1472.7
DSJC500.1	500	?,12	12	50 of 50	48	8878.8
			13	50 of 50	0	7.7
DSJC500.5	500	?,48	49	4 of 50	444	17838.6
			50	50 of 50	25	1567.4
			51	50 of 50	3	195
DSJC500.9	500	?,126	127	45 of 50	251	5552.8
			128	50 of 50	31	718.7
			129	50 of 50	11	245.9
DSJR500.1c	500	?,85	87	4 of 50	247	22126
DSJR500.5	500	?,122	127	12 of 50	153	10387
			128	16 of 50	98	7827
			129	29 of 50	36	2573
R1000.1c	1000	?,98	98	27 of 50	239	7086.4
			99	37 of 50	155	5580
R1000.5	1000	?,234	249	14 of 50	408	8700.1
			250	20 of 50	315	6681.3
			251	42 of 50	201	4231.4
			252	45 of 50	148	3105.5
R250.1c	250	?,64	66	2 of 50	0	0.1
R250.5	250	?,65	67	8 of 50	76	10788.4
			68	33 of 50	18	2685.7
U_13_3-v	857	4,4	5	50 of 50	0	0.3
flat1000_50_0	1000	50,50	50	46 of 50	399	694.1
			51	42 of 50	392	670.1
			52	48 of 50	362	631
			53	50 of 50	334	598.3
flat1000_60_0	1000	60,60	60	2 of 50	483	702.2
			61	3 of 50	497	781.6
			62	5 of 50	478	709.2
			63	9 of 50	526	780.8
flat1000_76_0	1000	76,83	88	9 of 50	300	4174.4
			89	50 of 50	169	3107.7
			90	50 of 50	38	755.5
flat300_20_0	300	20,20	20	50 of 50	0	2
flat300_26_0	300	26,26	26	39 of 50	208	5671
flat300_28_0	300	28,31	31	40 of 50	235	20237
			32	50 of 50	1	154.2
le450_15c	450	15,15	16	50 of 50	4	847.7
le450_15d	450	15,15	15	1 of 50	12	2246.6
			16	48 of 50	9	2222.7
le450_25c	450	25,25	26	49 of 50	9	954.6
			27	50 of 50	0	14.4
le450_25d	450	25,25	26	50 of 50	12	1313.3
			27	50 of 50	0	15.7

Table 3.4: Results for DYN-TABUCOL with a CPU time limit of 10 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for Foo-Tabucol with a cpu time limit of 10 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	cpu sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	2	188.8
DSJC1000.5	1000	?,83	89	1 of 50	228	2875.8
			90	46 of 50	302	5538.4
			91	50 of 50	73	1354
DSJC1000.9	1000	?,224	228	5 of 50	508	2029.9
			229	35 of 50	407	1579.9
			230	47 of 50	369	1437.5
DSJC500.1	500	?,12	12	40 of 50	154	25628.4
			13	50 of 50	0	16
DSJC500.5	500	?,48	49	1 of 50	169	10257.2
			50	50 of 50	58	3925.4
			51	50 of 50	4	299.5
DSJC500.9	500	?,126	127	19 of 50	345	6642.3
			128	50 of 50	58	1249.8
			129	50 of 50	15	294
DSJR500.1c	500	?,85	85	6 of 50	198	10856
			86	12 of 50	130	7480.3
			87	32 of 50	149	8816
DSJR500.5	500	?,122	128	5 of 50	196	4779.1
			129	35 of 50	194	4762.2
R1000.1c	1000	?,98	98	32 of 50	302	7248.2
			99	49 of 50	118	3230.3
R1000.5	1000	?,234	254	5 of 50	160	1414.2
			255	21 of 50	115	1099.6
R250.1c	250	?,64	64	45 of 50	42	5424.4
			65	50 of 50	6	779.1
			66	50 of 50	0	37.1
R250.5	250	?,65	67	3 of 50	343	20497.5
			68	50 of 50	74	4874.7
U_13_3-v	857	4,4	5	50 of 50	0	0.4
flat1000_50_0	1000	50,50	81	3 of 50	505	3165.7
			82	12 of 50	443	3397.2
flat1000_60_0	1000	60,60	86	4 of 50	460	6455
			87	29 of 50	459	6829
flat1000_76_0	1000	76,83	88	2 of 50	342	4847.6
			89	48 of 50	280	5072.9
			90	50 of 50	66	1255.9
flat300_20_0	300	20,20	20	50 of 50	0	20.4
flat300_26_0	300	26,26	26	50 of 50	23	870.6
flat300_28_0	300	28,31	31	34 of 50	198	19222.9
			32	50 of 50	2	285.2
le450_15c	450	15,15	15	17 of 50	200	28197.5
			16	50 of 50	4	193.4
le450_15d	450	15,15	15	2 of 50	233	36927.6
			16	50 of 50	3	189.7
le450_25c	450	25,25	26	50 of 50	19	2123.1
			27	50 of 50	0	28.2
le450_25d	450	25,25	26	50 of 50	25	2819.7
			27	50 of 50	0	27

Table 3.5: Results for Foo-Tabucol with a cpu time limit of 10 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 cpu and 512MB of ram.

Results for ACD-TABUCOL with a CPU time limit of 10 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	9	231.9
DSJC1000.5	1000	?,83	90	8 of 50	330	3179.6
			91	49 of 50	211	2078.2
DSJC1000.9	1000	?,224	229	1 of 50	458	1035.7
			230	11 of 50	485	1264.2
DSJC500.1	500	?,12	12	17 of 50	323	17175.3
			13	50 of 50	0	19
DSJC500.5	500	?,48	49	1 of 50	56	1714.8
			50	45 of 50	227	6866.5
			51	50 of 50	18	562.7
DSJC500.9	500	?,126	127	4 of 50	407	4964.1
			128	48 of 50	150	1912.6
			129	50 of 50	38	441.7
DSJR500.1c	500	?,85	85	11 of 50	278	4901.1
			86	47 of 50	122	2163.1
			87	50 of 50	22	436.2
DSJR500.5	500	?,122	127	14 of 50	179	4107.9
			128	24 of 50	99	2471.6
			129	36 of 50	57	1374.2
R1000.1c	1000	?,98	98	11 of 50	185	1875.4
			99	49 of 50	138	1509.5
R1000.5	1000	?,234	249	1 of 50	563	4612.5
			250	8 of 50	438	3531.1
			251	29 of 50	365	3139.6
			252	45 of 50	260	2188.3
R250.1c	250	?,64	64	15 of 50	5	317.5
			65	22 of 50	18	1123.7
			66	29 of 50	30	1814.3
R250.5	250	?,65	67	13 of 50	76	4900.1
			68	37 of 50	50	3192.3
U_13_3-v	857	4,4	4	50 of 50	5	257.1
			5	50 of 50	0	0.4
flat1000_50_0	1000	50,50	50	2 of 50	426	724
			51	2 of 50	504	959.1
			52	3 of 50	479	883.8
			53	5 of 50	433	796.4
flat1000_60_0	1000	60,60	87	2 of 50	405	3239.6
flat1000_76_0	1000	76,83	89	9 of 50	285	2790.2
			90	50 of 50	203	2069.2
flat300_20_0	300	20,20	20	50 of 50	0	10.9
flat300_26_0	300	26,26	26	50 of 50	24	685
flat300_28_0	300	28,31	31	10 of 50	327	15308
			32	50 of 50	7	375.3
le450_15c	450	15,15	15	42 of 50	125	7121.1
			16	50 of 50	5	144.6
le450_15d	450	15,15	15	26 of 50	165	9503.3
			16	50 of 50	6	164.3
le450_25c	450	25,25	26	49 of 50	162	6840.7
			27	50 of 50	0	34.2
le450_25d	450	25,25	26	47 of 50	192	8142.6
			27	50 of 50	0	29.9

Table 3.6: Results for ACD-TABUCOL with a CPU time limit of 10 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Summary for all six algorithms with a CPU time limit of 10 minutes								
Graph	$\|V\|$	χ, $k^\star$	PARTIALCOL			TABUCOL		
			FOO	DYN	ACD	FOO	DYN	ACD
DSJC1000.1	1000	?,20	21	21	21	21	**20**	21
DSJC1000.5	1000	?,83	90	90	90	**89**	**89**	90
DSJC1000.9	1000	?,224	230	229	232	228	**227**	229
DSJC500.1	500	?,12	13	**12**	13	**12**	**12**	**12**
DSJC500.5	500	?,48	50	50	50	**49**	**49**	**49**
DSJC500.9	500	?,126	128	128	128	**127**	**127**	**127**
DSJR500.1c	500	?,85	**85**	86	**85**	**85**	87	**85**
DSJR500.5	500	?,122	128	**127**	**127**	128	**127**	**127**
R1000.1c	1000	?,98	**98**	99	99	**98**	**98**	**98**
R1000.5	1000	?,234	253	251	252	254	**249**	**249**
R250.1c	250	?,64	**64**	**64**	**64**	**64**	66	**64**
R250.5	250	?,65	67	67	**66**	67	67	67
U_13_3-v	857	4,4	**4**	5	**4**	5	5	**4**
flat1000_50_0	1000	50,50	**50**	**50**	**50**	81	**50**	**50**
flat1000_60_0	1000	60,60	**60**	**60**	**60**	86	**60**	87
flat1000_76_0	1000	76,83	89	**88**	90	**88**	**88**	89
flat300_20_0	300	20,20	**20**	**20**	**20**	**20**	**20**	**20**
flat300_26_0	300	26,26	**26**	**26**	**26**	**26**	**26**	**26**
flat300_28_0	300	28,31	**28**	**28**	**28**	31	31	31
le450_15c	450	15,15	**15**	**15**	**15**	**15**	16	**15**
le450_15d	450	15,15	**15**	**15**	**15**	**15**	**15**	**15**
le450_25c	450	25,25	27	27	27	**26**	**26**	**26**
le450_25d	450	25,25	27	27	27	**26**	**26**	**26**

Table 3.7: All six algorithms with a CPU-time limit of 10 minutes compared. Only the smallest value of k for which a k-coloring has been found is reported for each algorithm. The lowest values for each graph are in bold face. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for DYN-PARTIALCOL with a CPU time limit of 60 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	20	3 of 50	2301	292947.5
			21	50 of 50	2	277.7
DSJC1000.5	1000	?,83	89	6 of 50	2786	45502.6
DSJC1000.9	1000	?,224	228	30 of 50	2275	14826.7
DSJC500.1	500	?,12	12	50 of 50	193	38819.9
DSJC500.5	500	?,48	49	1 of 50	811	55679.3
			50	50 of 50	291	22684.8
DSJC500.9	500	?,126	127	1 of 50	1680	43409.5
			128	50 of 50	347	9885.4
DSJR500.1c	500	?,85	85	3 of 50	989	56980.8
DSJR500.5	500	?,122	126	28 of 50	1544	79620.2
			127	44 of 50	631	34271.7
			128	47 of 50	147	7867.2
R1000.1c	1000	?,98	99	20 of 50	1895	28967.6
R1000.5	1000	?,234	249	10 of 50	2098	25024.1
			250	30 of 50	2020	24472.9
			251	47 of 50	1515	18487.7
			252	50 of 50	845	10178.7
R250.1c	250	?,64	64	4 of 50	635	89068.3
			65	13 of 50	488	64974.2
			66	16 of 50	225	29462
R250.5	250	?,65	66	6 of 50	2400	296613
			67	45 of 50	241	31809.9
U_13_3-v	857	4,4	5	50 of 50	0	1.7
flat1000_50_0	1000	50,50	50	50 of 50	26	107.9
flat1000_60_0	1000	60,60	60	50 of 50	91	390.4
flat1000_76_0	1000	76,83	88	9 of 50	2376	40543.7
flat300_28_0	300	28,31	28	13 of 50	1878	154261.2
			29	35 of 50	1398	133092.5
			30	46 of 50	1221	131767.5
			31	49 of 50	652	79871.8
le450_15c	450	15,15	15	50 of 50	3	615.7
			16	50 of 50	7	1617.4
le450_15d	450	15,15	15	50 of 50	22	4682.1
			16	50 of 50	7	1713.6
le450_25c	450	25,25	27	50 of 50	10	1583.3
le450_25d	450	25,25	27	50 of 50	7	1151.4

Table 3.8: Results for DYN-PARTIALCOL with a CPU time limit of 60 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for FOO-PARTIALCOL with a CPU time limit of 60 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	3	441.9
DSJC1000.5	1000	?,83	89	5 of 50	1893	32399.6
DSJC1000.9	1000	?,224	228	20 of 50	2296	13373.1
DSJC500.1	500	?,12	12	23 of 50	1811	324770.8
DSJC500.5	500	?,48	50	50 of 50	337	26470.5
DSJC500.9	500	?,126	128	48 of 50	1260	32862
DSJR500.1c	500	?,85	85	50 of 50	438	21243.1
DSJR500.5	500	?,122	128	24 of 50	1808	56146.6
R1000.1c	1000	?,98	98	2 of 50	604	8070.7
			99	30 of 50	1769	26593.4
R1000.5	1000	?,234	252	2 of 50	1967	15678.4
			253	35 of 50	1920	16310.1
R250.1c	250	?,64	64	50 of 50	4	453.9
			65	50 of 50	0	20.4
			66	50 of 50	0	8
R250.5	250	?,65	67	39 of 50	1331	128496.8
U_13_3-v	857	4,4	4	13 of 50	887	307040.7
			5	50 of 50	0	4.3
flat1000_50_0	1000	50,50	50	50 of 50	111	581.4
flat1000_60_0	1000	60,60	60	50 of 50	527	2929.9
flat1000_76_0	1000	76,83	88	10 of 50	2332	40546.5
flat300_28_0	300	28,31	28	35 of 50	1905	179148.1
			29	24 of 50	1587	162908.3
			30	26 of 50	1525	168488.1
			31	34 of 50	1464	184134
le450_15c	450	15,15	15	50 of 50	1	230
			16	50 of 50	1	137.5
le450_15d	450	15,15	15	50 of 50	3	592.9
			16	50 of 50	1	179.2
le450_25c	450	25,25	27	50 of 50	4	707.5
le450_25d	450	25,25	27	50 of 50	3	583.6

Table 3.9: Results for FOO-PARTIALCOL with a CPU time limit of 60 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for DYN-TABUCOL with a CPU time limit of 60 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	CPU sec	10^3 iter
DSJC1000.1	1000	?,20	20	14 of 50	1855	224021.7
			21	50 of 50	1	161.8
DSJC1000.5	1000	?,83	89	48 of 50	1224	17482.6
DSJC1000.9	1000	?,224	227	48 of 50	1520	7198.4
			228	50 of 50	718	3384.8
DSJC500.1	500	?,12	12	50 of 50	48	8878.8
DSJC500.5	500	?,48	49	11 of 50	1550	69803.6
			50	50 of 50	25	1567.4
DSJC500.9	500	?,126	127	50 of 50	328	7198.4
			128	50 of 50	32	718.7
DSJR500.1c	500	?,85	85	1 of 50	685	55458.3
DSJR500.5	500	?,122	126	5 of 50	746	56818.8
			127	12 of 50	154	10387
			128	18 of 50	313	19643.7
R1000.1c	1000	?,98	98	47 of 50	812	28505.3
			99	44 of 50	308	12207.7
R1000.5	1000	?,234	249	41 of 50	1245	29509.4
			250	44 of 50	740	17605.1
			251	50 of 50	381	8517.2
			252	48 of 50	205	4569.9
R250.1c	250	?,64	66	2 of 50	0	0.1
R250.5	250	?,65	67	11 of 50	528	64103.3
U_13_3-v	857	4,4	4	3 of 50	1371	656103
			5	50 of 50	0	0.3
flat1000_50_0	1000	50,50	50	50 of 50	421	732.8
flat1000_60_0	1000	60,60	60	49 of 50	1415	2099.6
flat1000_76_0	1000	76,83	88	46 of 50	1173	16532.9
flat300_28_0	300	28,31	31	50 of 50	378	32521.3
le450_15c	450	15,15	16	50 of 50	4	847.7
le450_15d	450	15,15	15	1 of 50	12	2246.6
			16	49 of 50	14	3572.4
le450_25c	450	25,25	26	49 of 50	9	954.6
			27	50 of 50	0	14.4
le450_25d	450	25,25	26	50 of 50	12	1313.3
			27	50 of 50	0	15.7

Table 3.10: Results for DYN-TABUCOL with a CPU time limit of 60 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Results for Foo-Tabucol with a cpu time limit of 60 minutes						
Graph	$\|V\|$	χ, $k^\star$	k	success	cpu sec	10^3 iter
DSJC1000.1	1000	?,20	21	50 of 50	2	188.8
DSJC1000.5	1000	?,83	89	22 of 50	2001	31773.6
DSJC1000.9	1000	?,224	227	39 of 50	2068	8760
			228	50 of 50	1120	5074.6
DSJC500.1	500	?,12	12	50 of 50	242	40328
DSJC500.5	500	?,48	49	6 of 50	1776	101117.2
			50	50 of 50	59	3925.4
DSJC500.9	500	?,126	127	47 of 50	1037	20129
			128	50 of 50	58	1249.8
DSJR500.1c	500	?,85	85	13 of 50	1347	68509.1
DSJR500.5	500	?,122	128	17 of 50	1494	31236.8
R1000.1c	1000	?,98	98	50 of 50	652	15795.8
			99	50 of 50	140	3801.1
R1000.5	1000	?,234	254	8 of 50	1140	7532.8
			255	31 of 50	645	4909.4
R250.1c	250	?,64	64	49 of 50	144	17726.6
			65	50 of 50	6	779.1
			66	50 of 50	0	37.1
R250.5	250	?,65	67	13 of 50	1834	108584.7
U_13_3-v	857	4,4	5	50 of 50	0	0.4
flat1000_50_0	1000	50,50	73	2 of 50	2987	8246.6
			74	2 of 50	2570	8120.4
			75	6 of 50	2489	8425.5
			76	12 of 50	2367	8377.5
flat1000_60_0	1000	60,60	79	1 of 50	3313	16661.2
			80	2 of 50	2912	16527.6
			81	10 of 50	2943	18253.6
			82	19 of 50	2870	20018.5
flat1000_76_0	1000	76,83	87	1 of 50	3060	36751.4
			88	14 of 50	1962	31241.9
flat300_28_0	300	28,31	29	1 of 50	1208	77181
			30	2 of 50	1007	79914.7
			31	50 of 50	515	50391.7
le450_15c	450	15,15	15	32 of 50	1072	193694.3
			16	50 of 50	4	193.4
le450_15d	450	15,15	15	11 of 50	995	161652.8
			16	50 of 50	3	189.7
le450_25c	450	25,25	26	50 of 50	19	2123.1
			27	50 of 50	0	28.2
le450_25d	450	25,25	26	50 of 50	25	2819.7
			27	50 of 50	0	27

Table 3.11: Results for Foo-Tabucol with a cpu time limit of 60 minutes. Only values of k for which at least one k-coloring could be found are reported. If several k were tested successfully, (at most) the four smallest values are reported. Values of k for which there was no successful run are not reported. $|V|$ is the number of vertices in the graph, χ is the chromatic number (where known), $k^\star$ is the best known coloring achieved (at the time of publication), k is the number of colors a coloring has been found and 10^3 iter is the mean number of iterations required to successfully find a k-coloring (unsuccessful runs are neglected). The tests were executed on Linux systems equipped with a 2GHz Pentium 4 cpu and 512MB of ram.

Graph	$\|V\|$	χ, $k^\star$	PARTIALCOL FOO	PARTIALCOL DYN	TABUCOL FOO	TABUCOL DYN	GH	MOR	MMT
DSJC1000.1	1000	?,20	21	**20**	21	**20**	20	21	20
DSJC1000.5	1000	?,83	**89**	**89**	**89**	**89**	83	88	83
DSJC1000.9	1000	?,224	228	228	**227**	**227**	224	226	226
DSJC500.1	500	?,12	**12**	**12**	**12**	**12**	12	12	12
DSJC500.5	500	?,48	50	**49**	**49**	**49**	48	49	48
DSJC500.9	500	?,126	128	**127**	**127**	**127**	126	128	127
DSJR500.1c	500	?,85	**85**	**85**	**85**	**85**	-	85	85
DSJR500.5	500	?,122	128	**126**	128	**126**	-	123	122
R1000.1c	1000	?,98	**98**	99	**98**	**98**	-	98	98
R1000.5	1000	?,234	252	**249**	254	**249**	-	241	237
R250.1c	250	?,64	**64**	**64**	**64**	66	-	64	64
R250.5	250	?,65	67	**66**	67	67	-	65	65
U_13_3-v	857	4,4	**4**	5	5	**4**	-	-	-
flat1000_50_0	1000	50,50	**50**	**50**	73	**50**	50	50	50
flat1000_60_0	1000	60,60	**60**	**60**	79	**60**	60	60	60
flat1000_76_0	1000	76,83	88	88	**87**	88	83	89	82
flat300_28_0	300	28,31	**28**	**28**	29	31	31	31	31
le450_15c	450	15,15	**15**	**15**	**15**	16	15	15	15
le450_15d	450	15,15	**15**	**15**	**15**	**15**	15	15	15
le450_25c	450	25,25	27	27	**26**	**26**	26	25	25
le450_25d	450	25,25	27	27	**26**	**26**	26	25	25

Table 3.12: Four algorithms with a CPU-time limit of 60 minutes compared. Only the smallest value of k for which a k-coloring has been found is reported for each algorithm. The lowest values of k for each graph among the four algorithms FOO-PARTIALCOL, DYN-PARTIALCOL, FOO-TABUCOL and DYN-TABUCOL are in bold face. The CPU times for the three algorithms GH, MOR and MMT are not reported and are sensibly larger than 60 minutes for some some large and difficult graphs. However, these results are given so that the reader may qualitatively compare our methods with state-of-the-art methods.

$2n$	degree	# graphs	time	graphs/sec	mean iter.	std dev.
12	5	7848	1.25 s	6280	62	75
14	7	21'609'301	2h 13 min	2689	129	193

Table 3.13: Results using a greedy initial solution. The CPU times were measured on a 2.6 GHz Pentium with 1GB of RAM.

$2n$	degree	# graphs	time	graphs/sec	mean iter.	std dev.
12	5	7848	1.1 s	7134	48	68
14	7	21'609'301	1h 58 min	3028	100	183

Table 3.14: Results reusing the previous coloring as an initial solution. The CPU times were measured on a 2.6 GHz Pentium with 1GB of RAM.

Chapter 4

Weighted Vertex Colorings

4.1 Introduction

The graph coloring problem has numerous applications. However, practical problems often require more complex models than simple vertex coloring. In this chapter, we will investigate heuristics used to solve problems involving a version of *weighted colorings*. The *weighted coloring problem* (WCP) can be described as follows: Given a *weighted graph* $G = (V, E, w)$ with a *weight function* $w : V \to \mathbb{R}^+$, a coloring c of the vertices of G with minimal weight must be determined. The weight w_c of a coloring c is defined as the sum of the weights of each color class $\mathcal{C}_i$:

$$w_c = \sum_i w(\mathcal{C}_i).$$

The weight $w(\mathcal{C}_i)$ of a color class $\mathcal{C}_i$ is defined to be

$$w(\mathcal{C}_i) = \max_{v \in C_i} w(v).$$

The WCP is $\mathcal{NP}$-hard [GJ79, Kar72]. This can be seen easily: If, for a given graph G, all weights are set to 1, the problem is equivalent to the problem of finding a coloring using a minimum number of colors for the underlying unweighted graph.

There are various applications of the WCP, such as in telecommunications or batch scheduling [GZ97, FJS04], for instance. As an intuitive example, we describe a batch scheduling problem.

Suppose that there is a set of tasks to be completed and that there is a completion time associated with each task. Suppose that, for each pair of tasks, it is known whether or not the two tasks can be processed simultaneously. Further, suppose that there is a machine which can process many compatible tasks (a batch) at the same time. The time required to process a batch is equal to the maximum completion time among all jobs in the batch. This problem can be modeled with a weighted graph, where the set of tasks corresponds to the vertices and the completion times correspond to the weights on the vertices. Two vertices are joined by an edge if the corresponding tasks are incompatible. A weighted coloring can be interpreted as a schedule, where each color corresponds to a batch of tasks.

The weight of a color class equals the completion time of the corresponding batch and the weight of the coloring corresponds to the completion time for all tasks.

In this chapter, we will focus on the development of several heuristics to solve the WCP. We will present two adaptations of tabu search: one working with feasible solutions and another working with infeasible solutions. We test several reactive tabu tenure schemes with each approach. Further, we devise an adaptive memory approach.

In order to test the developed heuristics, we also study ways of generating benchmark instances with different properties for the WCP.

4.2 Known Properties of Weighted Colorings

In this section, we will present some interesting, known results for the WCP. First, we introduce some additional notation. For a given weighted graph $G = (V, E, w)$, let $w^\star$ denote the minimum weight over all colorings of G. Let χ_w denote the minimum number of colors such that a coloring of weight $w^\star$ exists. One can easily construct examples where $\chi_w > \chi$:

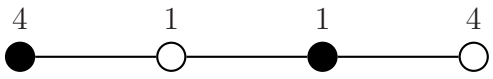

FIGURE 11 A weighted 2-chromatic graph with a 2-coloring of weight $4 + 4 = 8$.

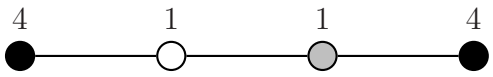

FIGURE 12 The same weighted graph as in Figure 11 with a minimum weight coloring using three colors. Its weight is $4 + 1 + 1 = 6$. For this example, we have $\chi_w > \chi$.

As we will see later in this chapter, for random instances of the WCP, there is generally a gap between the values for the best approximations obtained by heuristics for χ and for χ_w.

It is also trivial to construct an example which admits several minimum weight colorings using a different number of colors:

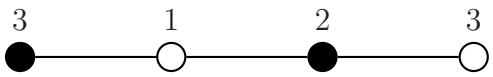

FIGURE 13 A minimum weight coloring using 2 colors with weight $3 + 3 = 6 = w^\star$.

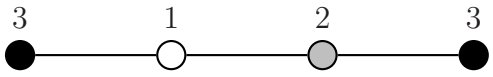

FIGURE 14 A minium weight coloring of the same graph as in Figure 13 using 3 colors (instead of $\chi_w = 2$) and weight $3 + 2 + 1 = 6 = w^\star$.

Clearly, χ is a lower bound for χ_w. The maximum degree of a graph plus one is an upper bound [DdWMP01].

Property 56

For any weighted graph $G = (V, E, w)$,

$$\chi \leq \chi_w \leq \Delta + 1.$$

Several properties of weighted colorings using integer weights and approximation algorithms have been derived in [DdWMP01, DdWMP02, MPdW$^+$04, DdWMP05].

4.2.1 Cases Solvable in Polynomial Time

In [FJS04], the authors present an $O(n^3)$ dynamic programming algorithm that solves the WCP on complements of *interval graphs*. (In fact, they solve the *weighted clique partitioning* problem, which is equivalent to the WCP on the complement of the graph.) An interval graph is defined as follows: The vertices correspond to intervals on the real line. Two vertices are joined by an edge if the two corresponding intervals intersect. Interval graphs can be recognized in linear time [BL76].

4.3 Generating Instances

To evaluate a heuristic, it is necessary to have a set of instances to benchmark the heuristic and to evaluate its solution quality and CPU time requirements.

In the ideal case, one uses instances for which the optimal solution is known. This allows one to evaluate the quality of the results produced by a heuristic. However, non-trivial instances of this type are difficult to obtain. Generally, large instances cannot be solved

exactly due to their size, and instances constructed to have a known optimum are generally easy to solve with a heuristic.

If the optimum of an instance is not known, the only possibility is to compare the results of different heuristics with each other. Ideally, one uses instances from other researchers and compares the obtained results with theirs. Unfortunately, we are not aware of any other heuristics for the WCP. Therefore, we have generated a large pool of benchmark instances of different types. We distinguish at least three categories of generated instances:

- Random instances.
- Constructed instances with special properties.
- Data from real, practical problems.

Random instances are often hard to approach and, except for very small instances, the optimal solution is not known because it is not possible to solve an $\mathcal{NP}$-hard problem to optimality. For randomly generated instances, different heuristics are compared with each other in order to get an idea of both the value of the optimal solution and the performance of the heuristic.

For other test cases, one may seek to construct instances such that certain properties are known. Ideally, one knows the value of the optimal solution or, at least, an upper bound (in the case of a minimization problem). These constructions may be more or less sophisticated and may attempt to "hide" certain features from the heuristic in order to make it more difficult to rediscover the (optimal) solution. Often, such constructed instances are very easy to solve. It is a challenging task in and of itself to devise methods of constructing "difficult" instances.

Having real-world instances is interesting and motivating. Moreover, most of the time, one can compare the solutions obtained with the solution that is actually used in practice. The difficulty of real-world instances is variable and must be analyzed for each case. Unfortunately, we are not aware of such real-world instances for the WCP available to the public.

4.3.1 Random Instances

To generate a random instance $G = (V, E, w)$ for the weighted coloring problem, we use three parameters:

- n, the number of vertices,
- d, the density of the graph,
- r, a random seed.

First, n vertices $v_1, \ldots, v_n$ are generated. Then, for every pair of vertices v_i, v_j with $i < j$, an edge is inserted with probability d. Finally, a random weight uniformly distributed in $[0, 1]$ is assigned to every vertex.

One such instance is named "unif01_n.d_r.col". For example, the instance with $n = 100$, $d = 0.5$ and $r = 43$ will be called "unif01_100.5_43.col".

Random graphs have been extensively studied [Bol01] since their introduction [ER59]. The asymptotic behavior of the chromatic number of a random graph (when the number of vertices n, goes to infinity) has been known since 1988 [Bol88], and it has been shown [AN04] that the chromatic number is concentrated at two possible values almost certainly when n grows.

Also, for "small" random graphs, the concentration of the chromatic number is confirmed by many numerical experiments, and rather precise estimates are available [JM82]. For instance, for random graphs with $n = 100$ vertices and density 0.5, almost all graphs are 15-colorable, and a few are 14-colorable. For the weighted coloring problem, however, we will see that we find that the best solutions require 18 or 19 colors.

4.3.2 Constructed Instances

We will describe two methods to construct instances with a known optimal solution for the WCP. The first method is straightforward. The construction results in a graph with a given chromatic number χ where an optimal solution for the WCP uses $\chi_w = \chi$ colors and its value $w^\star$ is known.

The second method is more sophisticated and constructs a graph where the chromatic number χ and the number χ_w of colors used in an optimal weight coloring differ. This is interesting because random instances show the same behavior.

Single Clique Method

The *single clique method* (SCM) allows one to generate an instance for the WCP where the optimal solution uses $\chi_w = \chi$ colors.

Required parameters for this methods are: the number n of vertices, the number k of colors to be used in an optimal coloring, the density d and a random seed r.

First, a clique on k vertices $v_1, \ldots, v_k$ is generated (to ensure that $\chi \geq k$) and a random weight $w(v_i)$ is assigned to each vertex v_i of the clique (according to some distribution). The first k vertices will be colored with the first k colors such that $c(v_i) = i$.

Then, all the other vertices v_j $(k < j \leq n)$ are colored with a random color $c(v_j)$ between 1 and k (to ensure that $\chi \leq k$). To every vertex v_j with $k < j \leq n$, a random weight between 0 and the weight of the vertex $v_{c(v_j)}$ (the vertex with the same color in the clique) is assigned.

Finally, edges between vertices of different colors are inserted with probability d. See Algorithm 7 for detailed pseudocode for the implementation of the single clique method.

It is obvious that every vertex of the clique $\{v_1, \ldots, v_k\}$ must be colored with a different color and, therefore, the weight of any coloring must be at least the sum of the weights of those vertices. By construction, the heaviest vertex of any color is the vertex in the clique. Therefore, the weight of the constructed coloring is exactly the sum of the weights of the vertices in the clique, which proves its optimality.

We will refer to instances of this type as sc-graphs.

Multiple Clique Method

The *multiple clique method* (MCM) generates test cases for the WCP such that the number of colors in an optimal weighted coloring can be up to almost twice the chromatic number of the instance.

The MCM takes the following parameters as input:

- the density d of the graph to be generated,
- the size ω of the maximum clique,
- the number $\gamma < \omega$ of cliques to generate,
- the number $n \geq \gamma\omega$ of vertices,
- a random seed r.

We will begin with describing in details how the MCM works and then prove that the instances generated with this method have the claimed properties.

First, γ cliques $K_i = \{v_{i,1}, \ldots, v_{i,\omega}\}$ of size ω are generated, with $i = 1, \ldots, \gamma$. Every clique K_i has a distinguished vertex $v_{i,1}$ which will be denoted by x_i. Then, edges are added between every pair of distinguished vertices x_i, $i = 1, \ldots, \gamma$ such that they form a clique of size γ.

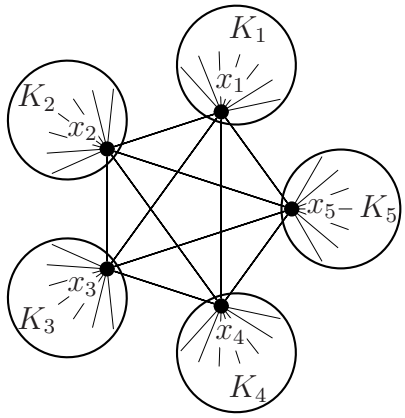

FIGURE 15 The initial cliques of size ω for $\gamma = 5$ for the *multiple cliques method*.

The weights for all undistinguished vertices are chosen according to some distribution. One should choose a distribution such that larger values have higher probability.

Let w_m be the smallest weight of the undistinguished vertices. Now, choose weights for the distinguished vertices such that the sum of the weights of the distinguished vertices is less than w_m.

Next, two colorings are defined. An ω-coloring c to ensure that $\chi \leq \omega$ (and, therefore, $\chi = \omega$, by the existence of a clique of size ω) and a $(\omega + \gamma - 1)$-coloring $c^\star$, which will be an optimal coloring for the WCP.

To construct c, distinct random colors between 1 and γ are assigned to the distinguished vertices x_i, $i = 1, \ldots, \gamma$. Then, for every clique K_i, $i = 1, \ldots, \gamma$, the colors $\{1, \ldots, \omega\} \setminus \{c(x_i)\}$ are randomly distributed among the $\omega - 1$ undistinguished vertices of every K_i.

The construction of $c^\star$ is done in an iterative manner. To start, no vertex is colored. At each step, the heaviest uncolored and undistinguished vertex of every clique K_i is selected and the smallest possible color is assigned to them. This results in a $(\omega-1)$-coloring (using colors 1 to $\omega - 1$) for the undistinguished vertices. Finally, $c^\star$ is completed by coloring the γ distinguished vertices (ordered by decreasing weights) with colors $\omega, \ldots, \omega + \gamma - 1$.

Finally, we add $n - \omega\gamma$ vertices, which will be named u_j, with $j = 1, \ldots, n - \omega\gamma$. For every new vertex u_j, a random color $c(u_j)$ in $\{1, \ldots, \omega\}$ and a random color $c^\star(u_j)$ in $\{1, \ldots, \omega + \gamma - 1\}$ is assigned to u_j.

A random weight $w(u_j)$, which is smaller than the maximal weight among the vertices with color $c^\star(u_j)$, is assigned to u_j.

To complete the instance, edges are added with probability d between vertices u, v with $c(u) \neq c(v)$ and $c^\star(u) \neq c^\star(v)$. To complete the generation of the graph, the weight $w^\star$ of the coloring $c^\star$ is computed, which is the weight of the optimal solution. See Algorithm 8 for a pseudocode implementation of the MCM.

Theorem 57

Let $G = (V, E, w)$ be an instance generated by the MCM with parameters n, ω and γ. The chromatic number of G then equals ω and an optimal weight coloring uses exactly $\omega + \gamma - 1$ colors and has weight $w^\star$, given by the coloring $c^\star$ in the MCM.

Proof

It is immediately clear that $\chi(G) = \omega$ because, by construction, we have an ω-coloring and a clique of size ω.

We can restrict ourselves to the case where $n = \omega\gamma$, i.e. the case where there are no additional vertices. Once we have proved the property for this case, it follows immediately for $n \geq \omega\gamma$, by the method of choosing the weights for the additional vertices, since the coloring will have the same weight. The optimal weight coloring of a graph cannot be smaller than an optimal weight coloring of one of its subgraphs.

From now on, fix $n = \omega\gamma$. Suppose we have a k-coloring c of G with $k \geq \omega$. We will show that we can construct a $(\omega + \gamma - 1)$-coloring $c^\star$ with a smaller weight. Let $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be the color classes of the coloring c, ordered by decreasing weights. Construct the coloring $c^\star$ as done by the MCM. Let $\{\mathcal{C}_1^\star, \ldots, \mathcal{C}_k^\star\}$ be the color classes of the coloring $c^\star$ (which are ordered by decreasing weights by construction). Let further denote u_i ($u_i^\star$ respectively) the heaviest vertex in $\mathcal{C}_i$ ($\mathcal{C}_i^\star$ respectively), such that we have $w(\mathcal{C}_i) = w(u_i)$ ($w(\mathcal{C}_i^\star) = w(u_i^\star)$ respectively).

Now, clearly, $w(\mathcal{C}_1) = w(\mathcal{C}_1^\star)$ (they each contain the heaviest vertex). Then, for any $i = 2, \ldots, \omega - 1$ we claim that $w(\mathcal{C}_i^\star) \leq w(\mathcal{C}_i)$.

By contradiction, suppose that the claim is false and let $j \leq \omega - 1$ be the smallest index

such that $w(\mathcal{C}_j^\star) > w(\mathcal{C}_j)$. This implies $w(u_j^\star) > w(u_j)$. Let α be the index of the clique with $u_j^\star \in K_\alpha$. By construction of $c^\star$, the vertex $u_j^\star$ is the j^{th} heaviest vertex in K_α. Therefore, in the clique K_α, there are j vertices $y_1, \ldots, y_j = u_i^\star$ with a weight greater than or equal to $w(u_i^\star)$. As a consequence, any coloring contains at least j color classes with a weight greater than or equal to $w(u^\star)$, which contradicts the fact that $w(\mathcal{C}_j) < w(u^\star)$, since the classes are ordered by decreasing weights. We conclude that $w(\mathcal{C}_i) \geq w(\mathcal{C}_i^\star)$ for $i = 1, \ldots, \omega - 1$.

The color classes $\mathcal{C}_i^\star$ with $\omega \leq i \leq \omega + \gamma - 1$ all consist of only one (distinguished) vertex. The sum of the weights of all those color classes is simply $\overline{w} = \sum_{i=1}^{\gamma} w(x_i)$, which is, by construction, smaller than the weight of any non-distinguished vertex.

So, if $\mathcal{C}_\omega$ contains any non-distinguished vertex, we can conclude that $w(\mathcal{C}_\omega) \geq \overline{w}$ and we are done. Otherwise, we conclude that all $\gamma \cdot (\omega - 1)$ non-distinguished vertices are contained in the sets $\mathcal{C}_i$ where $i = 1, \ldots, \omega - 1$. As the graph is covered by γ cliques, the maximum size of a stable set (and, therefore, of a color class) is γ. So, we conclude that no distinguished vertex is among the first $\omega - 1$ color classes. This implies that every distinguished vertex has neighbors of colors 1 to $\omega - 1$ and must, therefore, take a color greater than $\omega - 1$. As the distinguished vertices also form a clique, we conclude that, in fact, $k = \omega + \gamma - 1$ and that every distinguished vertex forms a color class of its own, as well. So, the total weight of the last γ classes of c and $c^\star$ is the same.

h.x.

4.3.3 Generated Instances for the Numerical Tests

We have generated three types of instances:

- Random graphs
- Graphs generated with the single clique method
- Graphs generated with the multiple clique method

The generated instances are available for download at `http://rose.epfl.ch/` under "Publications".

Random Graphs

We have chosen the number of vertices $n \in \{100, 200, 300, 500\}$ and for the density we have chosen $d \in \{0.2, 0.5, 0.8\}$. For every possible combination of n and d, we have generated three random graphs with random uniform weights between 0 and 1. The graphs are called "unif01_n.d_r" where $r \in \{41, 42, 43\}$ [Ada79] is the random seed used.

Single Clique Graphs

We have generated graphs of density 0.5 with a known chromatic number, which is also equal to the number of colors used in a minimum weight coloring. We have chosen the

number of vertices $n \in \{100, 200, 300, 500\}$ and the number of colors close to the number of colors found for random graphs with the same number of vertices and density 0.5. The graphs are named "sc$n$_k", where k is the number of colors used in a minimum weight coloring. See Table 4.7 for the list of generated instances.

Multiple Clique Graphs

These instances are called "mcm$n$_ω_γ", where n is the number of vertices, ω is the size of a maximum clique, and γ is the number of cliques of size ω. These graphs are ω-chromatic, and an optimum weight coloring uses $\omega + \gamma - 1$ colors. ω has been chosen to be close to the expected chromatic number [JM82] of a random graph with n vertices, and γ has been chosen such that $n/2 \leq \gamma\omega < n$. See Table 4.7 for the list of generated instances.

In the following sections, we will present our heuristics for the WCP.

4.4 A Tabu Search Using Feasible Solutions

In this section, we present an adaptation of the tabu search algorithm to the WCP. The main characteristic of this adaptation is the fact that it deals only with (feasible) colorings (no conflict is allowed) but with a variable number of colors. This algorithm will be referred to as the *feasible solution search* (FSS).

This approach has the advantage that the number of colors k is not an input parameter, so the FSS algorithm can be applied directly to unknown instances.

In what follows, we specify, for each element of a tabu search, how that element was adapted to the FSS.

Search Space

The search space consists of all possible colorings. The possible colors are in the set $\{1, \ldots, n\}$, where n is the number of vertices of the graph.

To evaluate the quality of a solution c, we simply use its weight.

$$w(c) = \sum_{\mathcal{C}_j \neq \emptyset} \left(\max_{v \in \mathcal{C}_j} w(v) \right),$$

where $\mathcal{C}_j$ is the set of vertices with color j.

Neighborhood

Two colorings are neighbors if they differ in exactly one vertex. A move consists of changing the color of a vertex. Note that two adjacent solutions might use different numbers of colors.

Tabu Status

The tabu status is assigned to couples of colors and vertices. If a vertex v is colored with color j and changes it to j', then it will be tabu to reassign color j to v for t iterations, where t is the current tabu tenure.

Choice of a Neighbor Solution

The neighborhood of the current solution is entirely explored and the weight of every neighbor solution is computed. This can be done efficiently using auxiliary data structures. For each vertex v and for each color j, we store the number of neighbors of v colored with color j. This allows to test whether or not vertex v can be colored with color j in constant time. Moreover, for each color j, we store the weights of the heaviest and second heaviest vertex. This allows to compute the difference of the objective function for a move in constant time.

Among all possible moves, the best (according to the objective function) non-tabu move is chosen. However, if there is a neighbor solution better than the best solution encountered so far, it is chosen even if it is tabu. This mechanism is called *aspiration*.

Stopping Criteria

The search is stopped if the best solution encountered has not been improved during $i_{\max}$ iterations, or if a given amount of CPU time has elapsed.

4.4.1 Implemented Schemes for the Tabu Tenure

We have implemented three reactive schemes to adapt the tabu tenure. The FOO-scheme and two ETB-schemes. See Sections 2.4 and 2.6 for a general definition and motivation of those schemes.

We have implemented an ETB-scheme using the Hamming distance (called ETB-H) and another using the similarity between colorings (called ETB-S).

ETB-H Scheme

After multiple tests on different instances, we have implemented the ETB-H scheme as follows: at the beginning of the algorithm, the reference solution s_{ref} is initialized to the current solution s_{cur}. Every 500 iterations, the Hamming distance between the reference solution s_{ref} and the current solution s_{cur} is computed. If this distance is less than $0.65 \cdot n$ (where n is the number of vertices of the graph), the tenure is incremented by 20. Otherwise, the reference solution s_{ref} is set to the current solution s_{cur}, and the tabu tenure is multiplied by 0.7. Each time a new best solution is encountered, the reference solution s_{ref} is reinitialized to the current solution s_{cur}, and the tabu tenure is multiplied by 0.7.

ETB-S Scheme

For the ETB-S scheme, which makes use of the measure of similarity between colorings, numerous tests were performed in order to determine a good (leading to high quality solutions) set of parameters determining the behavior of the scheme. Based on those tests, the we have implemented the ETB-S scheme as follows: at the beginning of the algorithm, the reference solution s_{ref} is initialized to the current solution s_{cur}. Every 500 iterations, the similarity between the reference solution s_{ref} and the current solution s_{cur} is computed. If the similarity is larger than 0.68 (meaning that the solution is still inside the ball around the reference solution), the tenure is incremented by 20. Otherwise, or when a new best solution has been found, the tabu tenure is multiplied by 0.7 and the reference solution s_{ref} is set to the current solution s_{cur}.

FOO-scheme

For the FOO-scheme, we have three parameters to adjust: the frequency φ, the increment η and the threshold b. After preliminary tests we have fixed $\varphi = 500$ and have adapted the adjustment of the tabu tenure as follows: if t is to be increased, $\eta + \lfloor t/20 \rfloor$ is added. If t is to be decreased, $1 + \lfloor t/20 \rfloor$ is subtracted. To tune the parameters η and b, we have conducted extensive tests on 4 random graphs. (See Table 4.1 for details.) The FOO-scheme shows the best performance for $i = 10$ and $i = 20$ and for $b = 0.4$ and $b = 0.5$. Based on these experiments, we have chosen the following parameter setting as a compromise:

Parameter	Value
Frequency φ	500
Increment η	15
Threshold b	0.45

.

4.5 A Tabu Search Using Infeasible Solutions

For this approach, we will consider any k-partition of the vertices as a solution in the solution space where k is given as an input to the algorithm. The algorithm will then try to render the coloring feasible while minimizing the weight of the coloring. We will refer to this algorithm as *infeasible solution search* (ISS).

The drawback of this approach is that the user of the method must supply the number of colors k, so he must have some forehand knowledge of the instance. Alternatively, one can apply the method repeatedly for a range of values for k and then retain the best solution. The most trivial range would be $[1, n]$ where n is the number of vertices. A more appropriate lower bound for the range would be the chromatic number χ, if known. As an upper bound $\Delta + 1$ is sufficient (by Property 56), where Δ is the maximal degree of the graph.

In practice, it is more efficient in terms of CPU time to start with the largest k in the range. The reason for this is that the algorithm often finds colorings using fewer than k colors if the value of k is considerably larger than the optimal value χ_w. This permits

the search process to skip one or several values of k while scanning the range from top to bottom.

Search Space

The search space is the set of all k-partitions of the set of vertices V.
A k-partition $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ of V can be represented as a function

$$c : V \to \{1, \ldots, k\}$$

such that

$$v \in \mathcal{C}_{c(v)} \quad \forall v \in V.$$

Note that if the algorithm finds a coloring with fewer than k colors, some of the sets $\mathcal{C}_i$ will be empty. Nevertheless, we will consider such configurations to be k-partitions as well.
A k-partition of the vertices of a graph $G = (V, E)$ can also be viewed as an improper k-coloring of the vertices of G or as a coloring of the graph $G' = (V, \emptyset)$.
The weight of a partition c is defined as an extension of the weight of a coloring.

$$w(c) = \sum_{j=1}^{k} \left(\max_{v \in C_j} w(v) \right) + \alpha \cdot |\{(u, v) \in E \mid c(u) = c(v)\}| \cdot \max_{v \in V} w(v) \quad \text{with } \alpha \geq 1. \quad (4.1)$$

The sum in equation (4.1) is simply the same weight function as for colorings. The second part of the function deals with the conflicts where for each conflict α times the maximum weight is added to the weight of the partition. This ensures that we are able to find a coloring with the same weight (for $\alpha = 1$) or less by introducing a new color for each conflict. The weight of each newly created color class is, of course, less than or equal to the maximum weight.
The weight of a partition in equation (4.1) is a generalization of the objective function of the TABUCOL algorithm [HdW87] for plain vertex coloring. To see this, consider the weights of all vertices set to one. We then obtain up to the additive constant k (first part of (4.1)), α times the objective function of TABUCOL, which counts the number of conflicting edges.
To tune α, several tests have shown that, with $\alpha = 1$, the algorithm often fails to find a feasible coloring for large graphs and a value of k close to the optimum χ_w. Based on preliminary tests, we use $\alpha = 2$ in our implementation.

Neighborhood and Moves

Given a partition c, we define the set of neighbors of a partition c to be the set of all partitions c' for which $c(v) \neq c'(v)$ for exactly one vertex v. A move consists of changing the color of exactly one vertex or, equivalently, moving a vertex from one color class to another.
The tabu status, the choice of a neighbor solutions, and the stopping criteria are the same

as for the Fss algorithm (see Section 4.4).

We have implemented the same reactive tabu schemes as we did for the Fss algorithm. Preliminary tests have shown that the same parameter settings lead to good results. Refer to Section 4.4.1 for details on the implementation and parameter settings.

4.6 An Adaptive Memory Search Algorithm

Tabu search is very well suited to find local minima quickly. However, it explores only a small region of the search space. It has been shown for several combinatorial problems that the performance of a tabu search can be improved using an *adaptive memory scheme* [RT95], also called a *genetic hybrid scheme* [Dav91].

An adaptive memory scheme uses a pool of solutions or pieces thereof. At every step (also called *generation*), a new solution is generated by a *recombination operator* which takes two or more solutions from the pool. This new solution is used as the initial solution for a tabu search. The solution returned by the tabu search is inserted into to the pool of solutions or discarded, according to rules that we will define later. To keep the size of the pool constant when a new solution is inserted, one solution is discarded according to carefully defined criteria. Possible criteria (to choose the solutions to be discarded) are the quality (measured by the objective function) where low quality solutions will be discarded [GH99], the age (the solutions which has been in the pool for the longest time will be discarded), a criterion related to the diversity of the pool [HGZ05] (for example, the solution in the pool which is most similar to the new solution is discarded) or a random choice [DH98]. In our implementation, we combined a diversity preserving criterion with a selection based on the quality of the solution.

The main advantage of an adaptive memory scheme is the fact that the tabu search visits very different regions of the search space. Because of this fact, it is crucial to maintain diversity among the solutions in the pool. There are two mechanisms to maintain diversity. First, the recombination operator generates solutions that lie in a potentially new region, provided the solutions used as input for the recombination operator are varied enough. Second, when inserting a new solution into the pool, the solution to be discarded must be chosen carefully. On the one hand, one should discard low quality solutions from the pool in order to make the search progress. On the other hand, one must make sure that diversity is maintained.

In the next section, we present how to design an adaptive memory algorithm for the weighted coloring problem.

4.6.1 Implementation

As we will see later in the Results Section, the Fss and Iss algorithms perform approximately equally. This is rather surprising because, for plain vertex coloring, the Fss approach does not lead to good results (in the sense of finding colorings using few colors) [Kob99, Cul92] compared to the Iss or partial solution approach. Moreover, the number of colors k to be used in our tests for the Iss algorithm has been determined using the Fss algorithm.

Because the FSS is simpler to use, we have chosen to implement an adaptive memory scheme using the FSS algorithm. We refer to our adaptive memory algorithm as AMA.

Pool of Solutions

The pool of solutions contains colorings that use a variable number of colors. The size p of the pool is constant along the search process. A size leading to good results (i.e. colorings with a low weight) has to be determined by running experiments. Useful values for p range from 10 to 50.

If the pool is too small, the diversity among the solutions is lost before very good solutions are discovered. As a consequence, the search is confined to a limited region of the search space, and the advantage of an adaptive memory algorithm is lost.

If the pool is too large, it takes a very long time to find good solutions because the average quality of the solutions in the pool is improved very slowly.

Initializing the Pool

The pool of solutions is initialized using a simple greedy algorithm to color the vertices. For every generated solution, the order in which the greedy algorithm considers the vertices is changed. For the first solution, the vertices are colored in the order of decreasing weights. The first half of the pool (except the first solution, which is already generated) is initialized with a greedy algorithm using a *slightly perturbed order* of decreasing weight. By "slightly perturbed order", we mean an ordering of the vertices such that, for most vertices, their position is close to the position they would have if ordered by decreasing weights. (See Algorithm 9 for more details.) We have chosen $\rho = 4$ in the perturbation algorithm. The second half of the solutions is initialized with greedy colorings on a completely random order. Finally, we apply a short tabu search to improve each initial solution individually.

Recombination of Colorings

At every generation, two colorings c_1 and c_2 are chosen randomly from the pool of solutions. The two colorings are combined into a new coloring c using a recombination operator inspired by the GPX operator [GH99]. The main difference between the two operators is the fact that ours combines two colorings to a new coloring while the GPX operator combines two k-partitions into a new k-partition. (See Algorithm 10 for details.) The basic idea is to build a new coloring c stepwise from the color classes of two colorings c_1 and c_2. First, the coloring c is empty, or, in other words, no vertex is colored by c. At each step, one of the two colorings c_1 and c_2 is chosen randomly. In the chosen coloring, the color class which contains the most uncolored vertices (with respect to c) is chosen, c is completed with this class, and, if possible, other uncolored vertices (with respect to c) are added greedily to complete the class.

Improving the Recombined Coloring

The coloring c that results from the recombination is used as an initial solution for a short tabu search, which returns a improved coloring c' (hopefully). "Short" means that

the number of iterations without improvement is typically set to a number between 500 and 10,000 [GH99, MMT05, HGZ05]. To compare, when applying tabu search alone to this problem, the number of iterations without improvement is typically set between 100,000 and several millions of iterations.

Updating the Pool

Two principles are applied when considering replacing a solution in the pool with a new solution. The first principle is to replace a solution in the pool which is similar to the new solution. This principle helps to keep a high diversity among the solutions in the pool. The second principle is to replace a solution which is of inferior quality (according to the objective function). This principle helps to increase the mean quality of the solutions in the pool.

Let s' be the new solution to be introduced into the pool. Let i be the number of solutions in the pool with an inferior or equal quality compared to s'. If i is smaller than a fraction b of the pool size p, i is set equal to bp. Then, among the i worst solutions in the pool, we choose the solution which is most similar to s' and replace it with s'. This approach has the nice feature that the best solution in the pool can only be replaced by a new solution which is at least as good, provided that $i < p$, which is the case for fractions $b < 1$.

Experiments have shown that a value of b close to 0.5 leads to best results in terms of the quality of the solutions found. For larger values of b, the solutions in the pool have a tendency to become more and more similar to each other, and the search process gets blocked in a region of the search space. This is due to the fact that the main criterion to eliminate a solution from the pool is the quality of the solution.

On the other hand, if b is small, the mean quality of the solutions in the pool improves only slowly.

4.6.2 Tuning of the Parameters for the AMA Algorithm

There are mainly three parameters to tune: the size of the pool p, the number of iterations without improvements in the stopping criterion of the tabu search, and the tabu tenure.

Over a large range of instances, we have tested values of p ranging from 10 to 50 and have obtained the best results (in terms of the weight of the colorings found) for $p = 20$. A small value of p leads to faster convergence, but the risk of loosing diversity is considerably larger. For a large value of p the convergence is slow, but the diversity of the pool is preserved much better. If a large amount of CPU time is available, a larger value of p is preferred as it leads to better solutions.

Tuning of the Tabu Tenure

A reactive scheme is not well suited for an adaptive memory method because the number of iterations for the tabu search procedure is small, which leaves too little time for reactive schemes to adjust the tabu tenure properly. Instead, we have chosen to use a static tenure, which depends on the number of vertices. To determine the best value, we have tested the algorithm on random graphs with 100, 200, 300, and 500 vertices. With preliminary tests,

we have selected ranges of tabu tenures leading to colorings with the lowest weight. For five values of the tabu tenure t within the determined range, we have conducted systematic tests.

For graphs with 100 vertices, the best solutions found are all identical, but the distribution of the values found (represented as histograms) are different. The histograms give an indication of the robustness of each algorithm. A narrow histogram indicates that the algorithm is robust because it has found similar values for all runs. A flat histogram indicates that the algorithm is not robust and that good results have been found by chance after several runs. Choosing $t = 50$ for $n = 100$ vertices seems to be the best choice because for this value of t the histograms are all very narrow. (See Table 4.2 for details.)

For $n = 200$ vertices, we have selected $t = 90$ as the ideal tenure because, for this value of t, the largest number (four out of 9 graphs) of best solutions has been found. (See Table 4.3 for details.)

For $n = 300$, we have selected $t = 120$ as the best tenure due to the low mean of the solutions found and the high number (three out of nine) of best solutions among different tabu tenures. (See Table 4.4 for more details.)

Finally, for $n = 500$, we have found $t = 170$ to be the best tenure. For this value, we obtain the highest number (four out of nine) of best solutions and the lowest mean of best solutions. (See Table 4.5 for details.)

Based on these results, we have chosen to set the tenure to

$$t = \lfloor 10\sqrt{n} - 50 \rfloor.$$

Note that this formula makes no sense for n smaller than 25 and should not be applied for $n \leq 50$. This formula produces values very similar to those we obtained from our experiments.

n	experimental t	$\lfloor 10\sqrt{n} - 50 \rfloor$
100	50	50
200	90	91
300	120	123
500	170	173

4.7 Numerical Results

Various tests have been performed in order to tune and compare various methods that we have proposed.

In a first series of tests, the FSS approach was tested with four algorithms: FOO-FSS, ETB-H-FSS, ETB-S-FSS and AMA. For the first series, for each graph, and for each algorithm, 50 runs were made, and each one was limited to 5 minutes of CPU time.

In a second series of tests, we allowed each run to use more CPU time. The infeasible solution approach (ISS) has been tested with the number of colors found in the first series. We also tested the FSS approach with more CPU time allowed (between 10 and 50 minutes).

In general, we report the name of the graph, the optimum value of a weighted coloring when known, and the best value obtained among all tests combined. Then, for each algorithm, we report the value of the best solution found (over several runs), the number of colors used in the best solution, and a histogram that shows the distribution of the weights of all obtained colorings. This gives a good indication of the robustness of the algorithm.

In a last series of tests, we tested the AMA algorithm with a large CPU time limit. Again, we used the results obtained (i.e. the number of colors used in the best coloring for each graph) to test the ISS approach with a large CPU time limit.

4.7.1 Preliminary Tests with FSS

To start our series of tests, we have performed 50 runs limited to 5 minutes CPU time on each random instance for the four algorithms FOO-FSS, ETB-H-FSS, ETB-S-FSS and AMA. The results are reported in Table 4.6.

For each graph, $k^\star$ was determined to be the number of colors for which the best coloring among all four algorithms was found. This $k^\star$ was used for the next series of tests as input parameter k for the ISS algorithms.

Note also that the number of colors used in the best coloring found (which is an estimation for χ_w) is considerably larger than the estimation for χ. In the following table, the values are given for random graphs of density 0.5.

number of vertices	χ (estimated)	χ_w (estimated)
100	15	18 - 19
200	23 - 24	29 - 30
500	48 - 49	62 - 65

Estimations of χ and χ_w for random graphs with density 0.5 and uniformly distributed real weights in the interval $[0, 1]$.

4.7.2 Short Tests With all Algorithms

The preliminary results with the FSS algorithms found a best coloring using $k^\star$ colors. (Note that this $k^\star$ may be different for each graph.) This $k^\star$ was used as the input parameter for the ISS algorithms. The CPU time limit depends on the size of each graph and is given here:

number of vertices	CPU time limit in minutes
100	5
200	10
300	30
500	50

For the FSS approach, each run has been repeated 30 times with various random seeds. For the ISS approach, we have chosen to test three values of k: $k^\star$, $k^\star + 2$ and $k^\star + 5$. For each value of k, 10 runs with different random seeds have been executed. Note that the

number of colors actually used by a coloring obtained by an ISS-algorithm can be lower than the supplied parameter k.

Results for Constructed Graphs

As expected, the instances constructed with the single clique method (called sc$n$_k) were easy to solve in the sense that each algorithm was able to solve almost all test cases to optimality. Moreover, the value of the random seed has little bearing on the algorithm's ability to find the optimum solution.

The multiple clique methods provides test cases which require more CPU time to be solved, if they can be solved at all. All instances but one were solved to optimality by at least one algorithm.

Results for the FSS Algorithms on Constructed Graphs

All single clique instances were solved to optimality by each FSS algorithm. Moreover, the optimum was discovered for all random seeds except for some random seeds on the graph sc500_60 which the FOO-FSS algorithm failed.

The multiple clique graphs were considerably harder to solve (in the sense that the optimum was found less often), and only the adaptive memory algorithm AMA was able to find the optimum for most instances (all but the graph mcm500_49_5). The graphs with 100 vertices were solved to optimality by all algorithms. Among the four algorithms, FOO-FSS obtains the worst solutions in terms of the weight of the colorings found. No mcm-graph with 200 vertices or more has been solved to optimality. The ETB-H-FSS algorithm performs rather poorly as well, but it still outperforms the FOO-scheme in 5 out of 6 instances with 200 vertices or more. The ETB-S scheme, which uses the similarity measure (introduced in Section 2.3.4) instead of the Hamming distance, performs very well and finds the optimum on 4 out of 6 mcm-graphs and finds the best solution among the four algorithms 5 out of 6 times. Nevertheless, the ETB-S-FSS algorithm is less robust than the AMA algorithm, which finds the optimum for any random seed on 4 out of 6 mcm-graphs and an optimal solution for 5 out of 6 mcm-graphs. The AMA algorithm is outperformed only on the graph mcm500_49_5. The detailed results of these tests can be found in Table 4.7, where we report the best coloring found for each algorithm, as well as the number of colors used in this best coloring.

Results for the ISS Algorithms on Constructed Graphs

Again, each algorithm solved to optimality the sc-graphs (instances generated with the single clique method) except sc500_60, where the FOO-ISS algorithm was not able to find the optimum solution. On the MCM-instances (generated with the multiple clique method), FOO-ISS outperforms the other algorithms and finds the best coloring for this tests series for the graph mcm500_49_5. However, as the flat histograms show, these results should be interpreted with care, because it is clear that the ISS algorithms lack robustness. Additionally, for the ISS algorithms, the ETB-S scheme (which uses the similarity between colorings) outperforms the ETB-H scheme (using the Hamming distance) except on 2 out of 8 graphs. For detailed results, see Table 4.8.

Comparing Fss and Iss

The histograms for the Iss algorithms on the mcm-graphs are much more flat than the histograms for the Fss algorithms. This indicates that the Fss algorithms are more robust. However, the lack of robustness of the Iss algorithms is also due to the three different parameters k supplied to the Iss algorithms for each graph.

Even though the performance of the Iss algorithms is slightly better than the performance of the Fss algorithms, the Iss algorithms require the knowledge of a good parameter k (the number of colors to use). If an automated scheme were used to determine k, a large amount of cpu time would be consumed on values of k which lead to colorings which are too heavy.

Results for Random Graphs

Random graphs seem to be difficult instances. Already for 200 vertices, the results become much more varied: for each algorithm and for each graph, the results for the different runs are varied, as are the differences between the best results of different algorithms.

Results for the Fss Algorithms on Random Graphs

For graphs with 100 vertices, the Ama algorithm outperforms all others and is very robust. It finds the best known solution for all 9 tested graphs. Still, for graphs with 100 vertices, Etb-S-Fss and Foo-Fss perform approximately equally, where the former shows best performance on graphs with density 0.2, and the latter performs best on graphs with density 0.5. Etb-H-Fss performs rather poorly in comparison with the other three algorithms.

For graphs with 200 vertices, Ama and Etb-S-Fss outperform Foo-Fss and Etb-H-Fss. Again, the Ama algorithm is more robust.

For graphs with 300 and 500 vertices, the best results were obtained with Etb-S-Fss (for densities of 0.2 and 0.5) and Etb-H-Fss, which performed very well on high density graphs. Surprisingly, Ama did not perform well. This is probably due to a rather short cpu time limit which does not allow the pool of 20 solutions to reach a very good solution.

The excellent performance of Etb-S-Fss suggests that using the Etb-S scheme inside the Ama algorithm (instead of the static tenure, which depends on the number of vertices only) may yield interesting results. For detailed results see Table 4.9.

Results for the Iss Algorithms on Random Graphs

Overall, the Iss algorithms performed very well on random graphs. Etb-H-Iss performs worse than the other and finds the best known solutions only for graphs with 100 vertices. For larger graphs, Etb-S-Iss finds the very best solutions if the density is low, and Foo-Iss finds the very best solutions for high density graphs. For graphs with density 0.5, both algorithms perform approximately equally. For detailed results, see Table 4.10. There are at least two possible explanations for this behavior: 1) the Foo-scheme performs generally better on high density graphs than on low density graphs; 2) the parameters for the method are tuned in such a way as to induce the observed performance. However, it

is difficult to fine-tune a method on large test cases, due to the large amount of CPU time required (several weeks) to obtain statistically significant data.

4.7.3 Tests with a Large CPU Time Limit

The previous tests did not show a significantly better performance of the AMA algorithm. Further, AMA was outperformed on most large graphs, even by other FSS algorithms. The reason for this is the relatively short CPU time, which did not allow the pool of 20 solutions to reach very good solutions.

We have, therefore, repeated the tests on the graphs with 200 or more vertices, except the sc-graphs, which were solved to optimality by almost all algorithms. The allowed CPU time was the following.

number of vertices	CPU time limit in hours
200	1
300	2
500	5

We have tested the AMA algorithm and one of its variants, ETB-S-AMA. This adaptive memory algorithm uses the ETB-S scheme in each improvement step. The number of iterations without improvement for each step has been set to 10,000 in order to allow the ETB-scheme some time to adapt. Further, we tested the two best ISS algorithms: FOO-ISS and the ETB-S-ISS.

Results for AMA and ETB-S-AMA

With the extended CPU time limit, the AMA algorithm solved all mcm-graphs to optimality. The ETB-S-AMA algorithm, however, was not able to find the optimal solution for the graph mcm500_49_5. For detailed results, see Table 4.11.

For random graphs with 200 vertices, ETB-S-AMA performs slightly better than AMA, especially on graphs with a high density. For larger graphs, AMA almost always outperforms ETB-S-AMA and obtains excellent results. See Table 4.12 for detailed results. This confirms that the tuning of the static tabu tenure was successful and that, for an adaptive memory algorithm, a reactive tabu scheme does not have the time to take effect during the short application of each improvement step.

Results for FOO-ISS and ETB-S-ISS

On the mcm-graphs, the FOO-ISS algorithm clearly outperforms ETB-S-ISS. See Table 4.13. The solutions found by FOO-ISS are better than or equal to the solutions found by ETB-S-ISS. In case of equality, the FOO-ISS algorithm seems to be more robust. The very poor performance on the graph mcm500_49_5 comes from the fact that the number of colors imposed is exactly the optimum or close to it. Due to this fact it the ISS-algorithms have troubles finding a feasible solution (i.e. a coloring without conflicts). With a larger number of colors, both algorithms perform considerably better. (See Table 4.8 to compare.)

For random graphs, the division between the two algorithms is very clear. For graphs with a density of 0.2, ETB-S-ISS always outperforms FOO-ISS, whereas, for densities 0.5 and 0.8, FOO-ISS very clearly outperforms ETB-S-ISS and obtains the very best results on all high density graphs with 500 vertices. One should keep in mind that the parameter k is essential for the ISS methods and that it is difficult to determine using only the ISS methods because numerous executions would be necessary to find a good k. The good performance of the ISS algorithms is partly due to the excellent performance of the FSS algorithms that supply the parameter k.

4.7.4 Summary and Interpretation of the Results

We have tested two types of tabu search algorithms, FSS and ISS. The performance of neither of those two approaches dominates the other. The main difference between them is that the FSS approach is easier to use.

Different reactive tabu schemes have been tested. The FOO-scheme is most effective for the ISS algorithm on graphs with high density. For the ETB-schemes, the algorithm using the similarity measure almost always achieves better performance than the algorithm using the Hamming distance. This is true for both the FSS and the ISS approaches.

Provided enough CPU time is available, the adaptive memory algorithm AMA achieves excellent results and was able to solve all constructed instances to optimality. Using the information on the number of colors $k^\star$ obtained by the AMA algorithm, the ISS algorithms achieve the best results on random graphs.

We were able to solve each constructed instance that was tested to optimality. The obtained data suggests that, for random graphs with 100 vertices, the instances have been solved to optimality as well, as most algorithms find the same minimal weight for different random seeds. We are confident that the algorithms presented herein are competitive, but further research be necessary in order to confirm this.

The results indicate that further investigation of adaptive memory algorithms using the ISS approach would be appropriate. Additionally, further testing with a static tabu tenure would show the impact of reactive tabu schemes.

Algorithm 7 The *single clique method* for the generation of an instance with known optimum for the WCP. Note that $U(a, b)$ returns a continuous random value between a and b, and UNIFORM(n, m) returns an integer uniformly distributed between n and m inclusive.

Input: The number of vertices n, the number of colors k, the density d, and an initial random seed r.

Output: An instance $G = (V, E, w)$ and $w^\star$, the weight of an optimal coloring.

initialize the random number generator with seed r
initialize the edge set $E \leftarrow \emptyset$
generate the vertex set $V \leftarrow \{v_1, \ldots, v_n\}$
initialize the weights $w(v_i) = 0 \ \forall i$
$w^\star \leftarrow 0$
/ Generate the optimal coloring and the weights */*
for all $i = 1, \ldots, n$ **do**
  if $i \leq k$ **then**
    / The k vertices in the clique */*
    $c(v_i) \leftarrow i$
    $w(v_i) \leftarrow 1 - (U(0, 1))^2$ */* favor larger weights */*
    $w^\star \leftarrow w^\star + w(v_i)$
  else
    $c(v_i) \leftarrow$ UNIFORM$(1, k)$ */* assign a random color */*
    / smaller weight than the weight of the corresponding vertex in the clique */*
    $w(v_i) \leftarrow U\left(0, w(v_{c(v_i)})\right)$
  end if
end for
/ Add the edges */*
for all $i = 1, \ldots, n - 1$ **do**
  for all $j = i + 1, \ldots, n$ **do**
    / Either it is a clique-edge, or an allowed random edge */*
    if $j \leq k$ **or** $\left(\text{UNIFORM}(0, 1) \leq d \text{ and } c(i) \neq c(j)\right)$ **then**
      $E \leftarrow E \cup \left\{\{i, j\}\right\}$
    end if
  end for
end for
return $G = (V, E, w)$, $w^\star$

Algorithm 8 The *multiple cliques method* for the generation of an instance with a known optimum for the WCP. Note that $U(a, b)$ returns a continuous random value between a and b, and UNIFORM(n, m) returns an integer uniformly distributed between n and m inclusive.

Input: The size of the cliques ω, the number of cliques $\gamma < \omega$, the number of vertices $n \geq \omega\gamma$, the density d, and an initial random seed r.

Output: An instance $G = (V, E, w)$ with $\chi(G) = \omega$ and $\chi_w(G) = \omega + \gamma - 1$, and $w^\star$, the weight of an optimal coloring.

initialize the random number generator with seed r
$V \leftarrow \{v_{i,j}\}$ for $i \in \{1, \ldots, \gamma\}$ and $j \in \{1, \ldots, \omega\}$
$V \leftarrow V \cup \{u_k\}$ for $k \in \{1, \ldots, n - \gamma\omega\}$.
$E \leftarrow \{\{v_{i,j}, v_{i,j'}\} \mid i \in \{1, \ldots, \gamma\},\ 1 \leq j < j' \leq \omega\}$ /* γ *cliques of size* ω */
$E \leftarrow E \cup \{\{v_{i,1}, v_{i',1}\} \mid 1 \leq i < i' \leq \gamma\}$ /* *clique on the distinguished vertices* */
$w(v_{i,j}) \leftarrow U(1/\omega, 1)$ for $i \in \{1, \ldots, \gamma\}$ and $j \in \{2, \ldots, \omega\}$
$w(v_{i,1}) \leftarrow U(0, 1)$ for $i \in \{1, \ldots, \gamma\}$
/* *render the sum of the weight of the distinguished vertices smaller than* $1/\omega$ */
$s \leftarrow \omega \sum_{i=1}^{\gamma} w(v_{i,1})$
$w(v_{i,1}) \leftarrow w(v_{i,1})/s$ for $i \in \{1, \ldots, \gamma\}$
$c(v_{i,j}) \leftarrow (j + i) \bmod \omega + 1$ for $i \in \{1, \ldots, \gamma\}$ and $j \in \{1, \ldots, \omega\}$
$c^\star(v_{i,j}) \leftarrow 0$ for $i \in \{1, \ldots, \gamma\}$ and $j \in \{0, \ldots, \omega\}$
for all $k \in \{1, \ldots, \omega - 1\}$ **do**
  for all $j \in \{1, \ldots, \gamma\}$ **do**
    $v_{i',j} \leftarrow \arg \max_{\{v_{i,j} \mid i=2,\ldots,\omega,\ c(v_{i,j})=0\}} w(v_{i,j})$
    $m_k \leftarrow w(v_{i',j})$ /* *weight of color* k */
    $c^\star(v_{i',j}) \leftarrow k$
  end for
end for
$c^\star(v_{i,1}) \leftarrow (i + \omega - 1)$, $m_i \leftarrow w(v_{i,1})$ for $i \in \{1, \ldots, \gamma\}$
for all $k \in \{1, \ldots, n - \omega\gamma\}$ **do**
  $c(u_k) \leftarrow$ UNIFORM$(1, \omega)$
  $c^\star(u_k) \leftarrow$ UNIFORM$(1, \omega + \gamma - 1)$
  $w(u_k) \leftarrow U\left(0, m_{c^\star(u_k)}\right)$
end for
for all $\{x, y\} \subseteq V$ **do**
  if $c(x) \neq c(y)$ **and** $c^\star(x) \neq c^\star(y)$ **and** $U(0, 1) < d$ **then**
    $E \leftarrow E \cup \big\{\{x, y\}\big\}$
  end if
end for
$w^\star \leftarrow w(c^\star)$
return $G = (V, E, w)$, $w^\star$

unif01_100.5.43

b \ i	i =1		i =5		i =10		i =20	
	best	mean	best	mean	best	mean	best	mean
b =0.2	11.546	12.094	11.729	12.038	11.621	11.869	11.514	11.982
b =0.3	11.514	11.837	11.549	11.852	11.468	11.867	11.393	11.757
b =0.4	11.316	11.656	11.321	11.541	11.128	11.514	11.170	11.430
b =0.5	11.271	11.429	11.138	11.387	11.128	11.365	11.128	11.174

unif01_200.5.43

b \ i	i =1		i =5		i =10		i =20	
	best	mean	best	mean	best	mean	best	mean
b =0.2	18.873	19.343	19.033	19.389	19.084	19.387	18.762	19.008
b =0.3	18.434	18.858	18.419	18.807	18.347	18.691	17.998	18.451
b =0.4	17.919	18.288	17.767	18.297	17.757	18.119	17.628	18.030
b =0.5	17.819	18.048	17.799	18.022	17.538	17.943	17.732	17.997

unif01_300.5.43

b \ i	i =1		i =5		i =10		i =20	
	best	mean	best	mean	best	mean	best	mean
b =0.2	24.581	24.866	24.524	24.735	24.495	24.734	24.481	24.661
b =0.3	24.189	24.439	24.275	24.520	24.299	24.581	24.232	24.477
b =0.4	24.028	24.387	24.046	24.306	23.956	24.289	24.148	24.382
b =0.5	23.978	24.386	24.093	24.366	24.213	24.493	24.124	24.346

unif01_500.5.43

b \ i	i =1		i =5		i =10		i =20	
	best	mean	best	mean	best	mean	best	mean
b =0.2	39.098	39.879	39.402	40.020	39.297	39.846	39.250	39.750
b =0.3	39.101	39.453	38.794	39.223	38.687	39.269	38.658	39.125
b =0.4	38.557	38.920	38.482	38.779	38.443	38.947	38.486	38.893
b =0.5	38.620	38.988	38.637	39.169	38.550	39.043	38.591	39.159

Table 4.1: Tests to tune the Foo-Fss algorithm. The frequency φ has been fixed to 500, and the increment i and the threshold b have been varied. For every parameter combination, 10 runs with different random seeds were executed. The CPU time limit was set to 10 minutes for all four graphs. For each parameter combination, the best solution found and the mean over the 10 runs is reported. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Algorithm 9 An algorithm to produce a *slightly perturbed order*

Input: The number of elements n, the order of perturbation ρ.
Output: A permutation which moves most elements not farther than ρ positions.

1: */* Initialization to the identity permutation */*
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad \pi(i) \leftarrow i$
4: **end for**
5: **for** $i = n, \ldots, \rho$ **do**
6: $\quad j \leftarrow i - \text{UNIFORM}(0, \rho)$
7: $\quad$ exchange $\pi(i)$ and $\pi(j)$
8: **end for**
9: **return** π

graph	w_{best}	t =30	t =40	t =50	t =60	t =70
unif01_100.2_41	5.402	**5.402(8)**	**5.402(8)**	**5.402(8)**	**5.402(8)**	**5.402(8)**
unif01_100.2_42	5.557	**5.557(8)**	**5.557(8)**	**5.557(8)**	**5.557(8)**	**5.557(8)**
unif01_100.2_43	5.657	**5.657(9)**	**5.657(9)**	**5.657(9)**	**5.657(9)**	**5.657(9)**
unif01_100.5_41	10.670	**10.670(18)**	**10.670(18)**	**10.670(18)**	**10.670(18)**	**10.670(18)**
unif01_100.5_42	10.398	**10.398(19)**	**10.398(19)**	**10.398(19)**	**10.398(19)**	**10.398(19)**
unif01_100.5_43	11.128	**11.128(19)**	**11.128(19)**	**11.128(19)**	**11.128(19)**	**11.128(19)**
unif01_100.8_41	18.579	**18.579(33)**	18.584(32)	**18.579(33)**	**18.579(33)**	**18.579(33)**
unif01_100.8_42	18.628	**18.628(32)**	**18.628(32)**	**18.628(32)**	**18.628(32)**	**18.628(32)**
unif01_100.8_43	19.039	**19.039(33)**	**19.039(33)**	**19.039(33)**	**19.039(33)**	**19.039(33)**
mean		11.673	11.674	11.673	11.673	11.673

Table 4.2: Tests for tuning the tabu tenure of the AMA algorithm for random graphs with 100 vertices. The CPU time limit was set to 5 minutes. The best tenure is $t = 50$, which results in the most robust results. Each test was repeated 20 times for different random seeds. w_{best} is best weight ever found for each instance. For each graph and each tenure, the value of the best solution and the number of colors used is reported. The best values of each line are in boldface. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	w_{best}	t =80	t =90	t =100	t =110	t =120
unif01_200.2_41	8.714	8.883(14)	**8.882(15)**	8.909(14)	8.894(15)	8.999(15)
unif01_200.2_42	8.417	8.622(14)	**8.502(14)**	8.583(14)	8.590(15)	8.644(14)
unif01_200.2_43	8.537	**8.678(14)**	8.685(14)	8.699(14)	8.758(14)	8.771(14)
unif01_200.5_41	17.922	18.120(29)	18.360(29)	**18.053(28)**	18.090(28)	18.136(30)
unif01_200.5_42	17.119	17.638(31)	**17.310(30)**	17.695(31)	17.318(31)	17.481(31)
unif01_200.5_43	17.409	17.724(29)	**17.559(30)**	17.610(31)	17.593(30)	17.569(29)
unif01_200.8_41	32.033	32.237(53)	32.340(53)	**32.165(53)**	32.273(53)	32.352(55)
unif01_200.8_42	31.469	32.167(57)	32.173(54)	32.258(56)	32.208(55)	**32.041(56)**
unif01_200.8_43	31.152	31.467(51)	31.559(52)	31.454(53)	31.294(54)	**31.195(53)**
mean		19.504	19.486	19.492	19.446	19.465

Table 4.3: Tests for tuning the tabu tenure of the AMA algorithm for random graphs with 200 vertices. The CPU time limit was set to 10 minutes. As the best tenure, $t = 90$ has been selected. Each test was repeated 15 times for different random seeds. w_{best} is best weight found for each instance. For each graph and each tenure, the value of the best solution and the number of colors used is reported. The best values of each line are in boldface. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	w_{best}	t =100	t =110	t =120	t =130	t =140
unif01_300.2_41	10.951	11.281(19)	11.221(19)	**11.146(19)**	11.196(18)	11.282(21)
unif01_300.2_42	11.041	11.496(19)	11.443(20)	**11.256(20)**	11.412(19)	11.537(20)
unif01_300.2_43	10.761	11.211(19)	11.132(19)	11.164(20)	11.130(19)	**10.997(19)**
unif01_300.5_41	22.938	**23.733(41)**	23.850(41)	23.989(42)	23.956(41)	23.991(42)
unif01_300.5_42	23.579	24.411(42)	24.507(42)	**24.376(41)**	24.437(42)	24.407(43)
unif01_300.5_43	23.384	**23.861(44)**	24.080(42)	24.093(43)	24.228(42)	24.223(41)
unif01_300.8_41	41.557	42.641(73)	**42.352(75)**	42.611(74)	42.536(77)	42.621(74)
unif01_300.8_42	42.742	44.463(73)	44.287(76)	44.181(75)	44.298(75)	**44.030(76)**
unif01_300.8_43	42.440	44.151(76)	44.233(76)	44.324(76)	44.182(77)	**44.073(77)**
mean		26.361	26.345	26.349	26.375	26.351

Table 4.4: Tests for tuning the tabu tenure of the AMA algorithm for random graphs with 300 vertices. The CPU time limit was set to 20 minutes. As the best tenure, $t = 120$ has been selected. Each test was repeated 15 times for different random seeds. w_{best} is best weight ever found for each instance. For each graph and each tenure, the value of the best solution and the number of colors used is reported. The best values of each line are in boldface. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	w_{best}	t =140	t =150	t =160	t =170	t =180
unif01_500.2_41	16.427	17.354(28)	17.243(28)	17.253(28)	**17.132(28)**	17.249(27)
unif01_500.2_42	16.214	17.181(27)	17.125(28)	17.160(28)	**16.991(30)**	17.172(29)
unif01_500.2_43	16.650	17.483(27)	**17.455(28)**	17.488(29)	17.575(27)	17.507(28)
unif01_500.5_41	35.917	38.271(66)	38.582(63)	38.366(62)	**38.116(63)**	38.382(64)
unif01_500.5_42	35.333	38.274(67)	38.277(62)	38.234(63)	38.363(64)	**38.032(64)**
unif01_500.5_43	36.730	39.104(64)	39.529(63)	39.340(64)	39.367(63)	**39.084(64)**
unif01_500.8_41	65.438	**69.928(115)**	70.848(117)	70.636(117)	70.054(117)	70.140(116)
unif01_500.8_42	64.534	70.639(116)	70.638(118)	**69.753(116)**	70.409(119)	70.050(117)
unif01_500.8_43	67.428	71.666(116)	72.133(116)	71.630(116)	**71.471(115)**	71.875(119)
mean		42.211	42.426	42.207	42.164	42.166

Table 4.5: Tests for tuning the tabu tenure of the AMA algorithm for random graphs with 500 vertices. The CPU time limit was set to 30 minutes. As the best tenure, $t = 170$ has been selected. Each test was repeated 15 times for different random seeds. w_{best} is best weight ever found for each instance. For each graph and each tenure, the value of the best solution and the number of colors used is reported. The best values of each line are in boldface. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Algorithm 10 The recombination operator for the AMA algorithm

Input: Two colorings c_1 and c_2, graph $G = (V, E)$.
Output: A coloring c obtained by recombining c_1 and c_2.

```
1: c(v) ← 0 for all v ∈ V
2: k ← 0 /* Number of colors used for c */
3: while ∃v ∈ V such that c(v) = 0 do
4:    i ← UNIFORM(1, 2)
5:    Let j be such that S_j = {v ∈ V | c(v) = 0 ∧ c_i(v) = j} has maximal cardinality.
      Break ties such that w(S_j) = max_{v∈S_j} w(v) is maximal.
6:    k ← k + 1
7:    for v ∈ V do
8:       if c_i(v) = j and c(v) = 0 then
9:          c(v) ← k
10:      end if
11:   end for
12:   /* Greedily complete color k */
13:   for v ∈ V such that c(v) = 0 do
14:      if v has no neighbor of color k with respect to the coloring c then
15:         c(v) ← k
16:      end if
17:   end for
18: end while
19: return c
```

graph	w_{best}	Foo-Fss		Etb-H-Fss		Etb-S-Fss		Ama	
unif01_100.2_41	5.402	5.591	(9)	5.503	(9)	**5.402**	(8)	**5.402**	(8)
unif01_100.2_42	5.557	5.635	(8)	5.593	(9)	5.624	(9)	**5.557**	(8)
unif01_100.2_43	5.657	5.753	(10)	5.735	(9)	**5.657**	(9)	**5.657**	(9)
unif01_100.5_41	10.670	**10.670**	(18)	10.681	(19)	10.675	(17)	**10.670**	(18)
unif01_100.5_42	10.398	10.585	(20)	**10.398**	(19)	**10.398**	(19)	**10.398**	(19)
unif01_100.5_43	11.128	**11.128**	(19)	11.154	(18)	**11.128**	(19)	**11.128**	(19)
unif01_100.8_41	18.579	18.594	(33)	18.584	(32)	18.584	(32)	**18.579**	(33)
unif01_100.8_42	18.628	18.732	(34)	18.679	(33)	18.679	(33)	**18.628**	(32)
unif01_100.8_43	19.039	19.157	(33)	19.082	(33)	19.118	(32)	**19.039**	(33)
unif01_200.2_41	8.714	9.536	(16)	9.094	(15)	8.898	(14)	**8.891**	(15)
unif01_200.2_42	8.417	9.387	(16)	8.759	(15)	8.738	(14)	**8.547**	(13)
unif01_200.2_43	8.537	9.207	(14)	8.773	(14)	**8.583**	(14)	8.755	(14)
unif01_200.5_41	17.922	18.352	(30)	18.606	(30)	18.371	(30)	**18.324**	(31)
unif01_200.5_42	17.119	17.565	(32)	17.617	(31)	17.492	(31)	**17.374**	(30)
unif01_200.5_43	17.409	17.740	(32)	17.812	(31)	17.698	(31)	**17.622**	(31)
unif01_200.8_41	32.033	**32.033**	(56)	32.431	(56)	32.376	(57)	32.132	(53)
unif01_200.8_42	31.469	32.292	(59)	31.991	(60)	32.091	(60)	**31.596**	(57)
unif01_200.8_43	31.152	31.598	(55)	31.542	(55)	31.633	(57)	**31.190**	(53)
unif01_300.2_41	10.951	11.897	(20)	11.274	(19)	**11.186**	(19)	11.596	(20)
unif01_300.2_42	11.041	12.048	(19)	11.492	(19)	**11.489**	(20)	11.750	(20)
unif01_300.2_43	10.761	12.034	(18)	**11.243**	(19)	11.352	(19)	11.588	(20)
unif01_300.5_41	22.938	**23.515**	(45)	23.662	(42)	23.520	(42)	24.808	(44)
unif01_300.5_42	23.579	24.604	(46)	24.451	(43)	**24.157**	(44)	25.506	(44)
unif01_300.5_43	23.384	23.944	(45)	24.000	(45)	**23.744**	(43)	25.060	(43)
unif01_300.8_41	41.557	42.778	(81)	42.411	(79)	42.583	(81)	**42.298**	(78)
unif01_300.8_42	42.742	44.577	(80)	**43.750**	(78)	43.894	(81)	44.010	(75)
unif01_300.8_43	42.440	43.873	(84)	43.611	(80)	**43.541**	(78)	44.254	(77)
unif01_500.2_41	16.427	18.116	(28)	**17.008**	(27)	17.014	(27)	17.675	(26)
unif01_500.2_42	16.214	17.723	(29)	16.890	(27)	**16.702**	(28)	17.411	(28)
unif01_500.2_43	16.650	18.108	(29)	17.452	(29)	**17.169**	(28)	17.937	(29)
unif01_500.5_41	35.917	37.953	(68)	37.818	(65)	**37.485**	(63)	38.831	(67)
unif01_500.5_42	35.333	37.427	(68)	37.171	(64)	**36.867**	(64)	38.054	(65)
unif01_500.5_43	36.730	38.520	(67)	38.438	(65)	**38.256**	(64)	39.880	(67)
unif01_500.8_41	65.438	70.001	(122)	69.700	(124)	**69.576**	(122)	71.453	(127)
unif01_500.8_42	64.534	70.200	(127)	**68.667**	(121)	69.140	(126)	72.048	(128)
unif01_500.8_43	67.428	71.260	(125)	**69.788**	(121)	69.933	(123)	72.670	(126)

Table 4.6: First series of tests with the Ama and Fss algorithms. The cpu time limit was 5 minutes. For each graph, the best value w_{best} found is reported. The histograms are 1 unit wide and report the distribution of the obtained values over 50 runs. The weight and the number of colors used for the best coloring found is reported for each graph and for each algorithm. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 cpu and 512MB of ram.

graph	$w^\star$	w_{best}	Foo-Fss		Etb-H-Fss		Etb-S-Fss		Ama	
mcm100_15_5	8.889	8.889	**8.889**	(19)	**8.889**	(19)	**8.889**	(19)	**8.889**	(19)
mcm100_19_3	9.353	9.353	**9.353**	(21)	**9.353**	(21)	**9.353**	(21)	**9.353**	(21)
mcm200_25_5	13.751	13.751	13.976	(32)	**13.751**	(29)	**13.751**	(29)	**13.751**	(29)
mcm200_30_4	16.041	16.041	16.465	(34)	16.445	(35)	16.046	(33)	**16.041**	(33)
mcm300_30_5	16.048	16.048	17.193	(43)	16.187	(35)	**16.048**	(34)	**16.048**	(34)
mcm300_36_6	19.262	19.262	20.191	(46)	19.368	(43)	**19.262**	(41)	**19.262**	(41)
mcm500_20_19	12.673	12.673	12.961	(38)	**12.673**	(38)	**12.673**	(38)	**12.673**	(38)
mcm500_49_5	25.771	25.771	29.864	(71)	29.303	(63)	29.345	(66)	**28.856**	(61)
sc100_17	10.711	10.711	**10.711**	(17)	**10.711**	(17)	**10.711**	(17)	**10.711**	(17)
sc100_18	11.263	11.263	**11.263**	(18)	**11.263**	(18)	**11.263**	(18)	**11.263**	(18)
sc200_29	19.039	19.039	**19.039**	(29)	**19.039**	(29)	**19.039**	(29)	**19.039**	(29)
sc200_30	19.318	19.318	**19.318**	(30)	**19.318**	(30)	**19.318**	(30)	**19.318**	(30)
sc200_31	19.585	19.585	**19.585**	(31)	**19.585**	(31)	**19.585**	(31)	**19.585**	(31)
sc300_41	25.963	25.963	**25.963**	(41)	**25.963**	(41)	**25.963**	(41)	**25.963**	(41)
sc300_42	26.400	26.400	**26.400**	(42)	**26.400**	(42)	**26.400**	(42)	**26.400**	(42)
sc300_43	26.543	26.543	**26.543**	(43)	**26.543**	(43)	**26.543**	(43)	**26.543**	(43)
sc500_60	36.694	36.694	**36.694**	(60)	**36.694**	(60)	**36.694**	(60)	**36.694**	(60)
sc500_65	40.543	40.543	**40.543**	(65)	**40.543**	(65)	**40.543**	(65)	**40.543**	(65)
sc500_70	43.900	43.900	**43.900**	(70)	**43.900**	(70)	**43.900**	(70)	**43.900**	(70)

Table 4.7: Tests with the Fss algorithms and a short cpu time limit. For each graph, the optimum $w^\star$ and the best value found w_{best} is reported. For each algorithm, a histogram, the value of the best solution found and, in parentheses, the number of colors used in the best solution are reported. The histogram is one unit wide, and each bucket is 0.05 units wide. The cpu time allowed was 5 minutes for graphs with 100 vertices, 10 minutes for graphs with 200 vertices, 30 minutes for graphs with 300 vertices and 50 minutes for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 cpu and 512MB of ram.

graph	$w^\star$	w_{best}	FOO-ISS		ETB-H-ISS		ETB-S-ISS	
mcm100_15_5	8.889	8.889	**8.889**	(19)	**8.889**	(19)	**8.889**	(19)
mcm100_19_3	9.353	9.353	**9.353**	(21)	**9.353**	(21)	**9.353**	(21)
mcm200_25_5	13.751	13.751	**13.751**	(29)	13.957	(30)	**13.751**	(29)
mcm200_30_4	16.041	16.041	16.264	(35)	**16.145**	(33)	16.157	(33)
mcm300_30_5	16.048	16.048	**16.048**	(34)	18.780	(38)	16.057	(34)
mcm300_36_6	19.262	19.262	**19.262**	(41)	21.169	(44)	**19.262**	(41)
mcm500_20_19	12.673	12.673	**12.673**	(38)	**12.673**	(38)	**12.673**	(38)
mcm500_49_5	25.771	25.771	**26.753**	(60)	28.525	(61)	28.611	(61)
sc100_17	10.711	10.711	**10.711**	(17)	**10.711**	(17)	**10.711**	(17)
sc100_18	11.263	11.263	**11.263**	(18)	**11.263**	(18)	**11.263**	(18)
sc200_29	19.039	19.039	**19.039**	(29)	**19.039**	(29)	**19.039**	(29)
sc200_30	19.318	19.318	**19.318**	(30)	**19.318**	(30)	**19.318**	(30)
sc200_31	19.585	19.585	**19.585**	(31)	**19.585**	(31)	**19.585**	(31)
sc300_41	25.963	25.963	**25.963**	(41)	**25.963**	(41)	**25.963**	(41)
sc300_42	26.400	26.400	**26.400**	(42)	**26.400**	(42)	**26.400**	(42)
sc300_43	26.543	26.543	**26.543**	(43)	**26.543**	(43)	**26.543**	(43)
sc500_60	36.694	36.694	36.698	(61)	**36.694**	(60)	**36.694**	(60)
sc500_65	40.543	40.543	**40.543**	(65)	**40.543**	(65)	**40.543**	(65)
sc500_70	43.900	43.900	**43.900**	(70)	**43.900**	(70)	**43.900**	(70)

Table 4.8: Tests with the ISS algorithms. For each graph, the optimum $w^\star$ and the best value ever found w_{best} is reported. For every graph and for every algorithm, three times ten runs were executed with ten different random seeds and three different values for k, namely $k^\star$, $k^\star + 2$ and $k^\star + 5$, where $k^\star$ is the number of colors for which the best solution with the FSS algorithms was found in the first series of tests (see Table 4.6). For each algorithm, a histogram, the value of the best solution found and, in parentheses, the number of colors used in the best solution are reported. The histograms are one unit wide, and each bucket is 0.05 units wide. The CPU time allowed was 5 minutes for graphs with 100 vertices, 10 minutes for graphs with 200 vertices, 30 minutes for graphs with 300 vertices and 50 minutes for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	w_{best}	Foo-Fss		Etb-H-Fss		Etb-S-Fss		Ama	
unif01_100.2_41	5.402	5.667	(10)	**5.402**	(8)	**5.402**	(8)	**5.402**	(8)
unif01_100.2_42	5.557	5.673	(9)	5.593	(9)	**5.557**	(8)	**5.557**	(8)
unif01_100.2_43	5.657	5.844	(10)	**5.657**	(9)	**5.657**	(9)	**5.657**	(9)
unif01_100.5_41	10.670	**10.670**	(18)	10.681	(19)	**10.670**	(18)	**10.670**	(18)
unif01_100.5_42	10.398	**10.398**	(19)	**10.398**	(19)	**10.398**	(19)	**10.398**	(19)
unif01_100.5_43	11.128	**11.128**	(19)	11.138	(20)	**11.128**	(19)	**11.128**	(19)
unif01_100.8_41	18.579	**18.584**	(32)	**18.584**	(32)	18.600	(33)	**18.584**	(32)
unif01_100.8_42	18.628	**18.628**	(32)	18.669	(34)	18.679	(33)	**18.628**	(32)
unif01_100.8_43	19.039	**19.039**	(33)	19.050	(32)	19.071	(33)	**19.039**	(33)
unif01_200.2_41	8.714	9.517	(15)	9.094	(15)	**8.716**	(14)	8.864	(14)
unif01_200.2_42	8.417	9.482	(16)	8.759	(15)	8.591	(14)	**8.554**	(14)
unif01_200.2_43	8.537	9.137	(14)	8.828	(14)	**8.583**	(14)	8.629	(14)
unif01_200.5_41	17.922	**17.972**	(30)	18.470	(31)	18.140	(30)	18.070	(29)
unif01_200.5_42	17.119	17.459	(32)	17.591	(31)	**17.417**	(30)	17.421	(31)
unif01_200.5_43	17.409	17.548	(31)	17.812	(31)	17.642	(31)	**17.471**	(30)
unif01_200.8_41	32.033	**32.036**	(56)	32.431	(56)	32.376	(57)	32.066	(53)
unif01_200.8_42	31.469	32.185	(59)	**31.726**	(60)	31.840	(59)	32.062	(56)
unif01_200.8_43	31.152	**31.219**	(55)	31.542	(55)	31.633	(57)	31.353	(53)
unif01_300.2_41	10.951	11.928	(20)	11.274	(19)	**11.152**	(20)	11.241	(19)
unif01_300.2_42	11.041	12.168	(22)	**11.388**	(20)	11.408	(19)	11.464	(19)
unif01_300.2_43	10.761	11.992	(22)	11.241	(19)	11.207	(19)	**11.107**	(20)
unif01_300.5_41	22.938	23.518	(44)	23.420	(42)	**23.218**	(42)	23.785	(40)
unif01_300.5_42	23.579	24.459	(46)	24.342	(42)	**24.000**	(42)	24.113	(41)
unif01_300.5_43	23.384	23.841	(45)	24.026	(43)	**23.602**	(43)	23.878	(42)
unif01_300.8_41	41.557	**42.225**	(80)	42.411	(79)	42.404	(78)	42.330	(74)
unif01_300.8_42	42.742	44.694	(81)	**43.105**	(78)	43.656	(79)	43.892	(75)
unif01_300.8_43	42.440	43.575	(82)	**43.443**	(79)	43.541	(78)	44.025	(75)
unif01_500.2_41	16.427	18.016	(29)	17.010	(27)	**16.695**	(27)	17.163	(28)
unif01_500.2_42	16.214	17.707	(28)	16.788	(26)	**16.377**	(27)	16.899	(29)
unif01_500.2_43	16.650	18.167	(30)	17.353	(29)	**16.774**	(28)	17.303	(28)
unif01_500.5_41	35.917	37.321	(66)	37.306	(64)	**36.829**	(63)	38.198	(63)
unif01_500.5_42	35.333	36.921	(66)	36.826	(65)	**36.161**	(62)	37.993	(62)
unif01_500.5_43	36.730	37.776	(69)	38.086	(65)	**37.562**	(64)	38.955	(63)
unif01_500.8_41	65.438	68.112	(123)	67.355	(118)	**66.949**	(119)	68.646	(111)
unif01_500.8_42	64.534	67.994	(127)	67.401	(121)	**67.386**	(122)	68.437	(115)
unif01_500.8_43	67.428	69.701	(125)	**69.556**	(120)	69.606	(121)	70.727	(113)

Table 4.9: Tests with the Fss algorithms on random graphs. For each graph the best value ever found w_{best} is reported. For each algorithm, a histogram, the value of the best solution found; and, in parentheses, the number of colors used in the best solution are reported. For each graph and each algorithm 10 runs with different random seed were made. The histogram is one unit wide, and each bucket is 0.05 units wide. The best value found for each line is in bold face. The cpu time allowed was 5 minutes for graphs with 100 vertices, 10 minutes for graphs with 200 vertices, 30 minutes for graphs with 300 vertices and 50 minutes for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 cpu and 512MB of ram.

graph	w_{best}	Foo-Iss		Etb-H-Iss		Etb-S-Iss	
unif01_100.2_41	5.402	5.530	(8)	5.503	(9)	**5.402**	(8)
unif01_100.2_42	5.557	5.621	(9)	**5.557**	(8)	**5.557**	(8)
unif01_100.2_43	5.657	5.735	(9)	5.716	(10)	**5.657**	(9)
unif01_100.5_41	10.670	**10.670**	(18)	10.715	(18)	**10.670**	(18)
unif01_100.5_42	10.398	**10.398**	(19)	**10.398**	(19)	**10.398**	(19)
unif01_100.5_43	11.128	**11.128**	(19)	11.138	(20)	**11.128**	(19)
unif01_100.8_41	18.579	**18.584**	(32)	**18.584**	(32)	**18.584**	(32)
unif01_100.8_42	18.628	**18.628**	(32)	**18.628**	(32)	**18.628**	(32)
unif01_100.8_43	19.039	**19.039**	(33)	19.050	(32)	19.050	(32)
unif01_200.2_41	8.714	9.602	(15)	9.098	(14)	**8.901**	(13)
unif01_200.2_42	8.417	9.279	(14)	8.880	(14)	**8.552**	(14)
unif01_200.2_43	8.537	9.319	(14)	8.996	(14)	**8.540**	(14)
unif01_200.5_41	17.922	**18.179**	(29)	18.682	(29)	18.387	(30)
unif01_200.5_42	17.119	**17.422**	(30)	17.448	(30)	17.429	(30)
unif01_200.5_43	17.409	**17.417**	(30)	17.876	(30)	17.458	(29)
unif01_200.8_41	32.033	**32.268**	(53)	32.384	(54)	32.546	(56)
unif01_200.8_42	31.469	**31.629**	(59)	32.195	(60)	32.040	(57)
unif01_200.8_43	31.152	**31.305**	(53)	32.027	(54)	31.972	(54)
unif01_300.2_41	10.951	12.068	(20)	11.421	(19)	**10.987**	(19)
unif01_300.2_42	11.041	12.308	(21)	11.504	(19)	**11.104**	(18)
unif01_300.2_43	10.761	12.147	(20)	11.308	(18)	**10.993**	(19)
unif01_300.5_41	22.938	**23.037**	(43)	24.051	(41)	23.197	(41)
unif01_300.5_42	23.579	**23.815**	(42)	24.660	(42)	24.184	(41)
unif01_300.5_43	23.384	23.715	(42)	23.956	(44)	**23.624**	(42)
unif01_300.8_41	41.557	**41.697**	(76)	42.424	(77)	42.832	(76)
unif01_300.8_42	42.742	**42.742**	(77)	44.022	(78)	43.957	(79)
unif01_300.8_43	42.440	**42.995**	(74)	44.319	(77)	44.203	(76)
unif01_500.2_41	16.427	18.303	(26)	17.282	(27)	**16.642**	(27)
unif01_500.2_42	16.214	18.049	(28)	16.952	(27)	**16.435**	(26)
unif01_500.2_43	16.650	18.678	(28)	17.329	(26)	**16.796**	(27)
unif01_500.5_41	35.917	**36.367**	(63)	37.776	(62)	36.515	(62)
unif01_500.5_42	35.333	36.349	(65)	36.958	(63)	**36.112**	(64)
unif01_500.5_43	36.730	**37.378**	(63)	38.866	(62)	37.495	(64)
unif01_500.8_41	65.438	**66.231**	(115)	68.526	(119)	68.037	(116)
unif01_500.8_42	64.534	**65.940**	(116)	67.473	(117)	66.995	(120)
unif01_500.8_43	67.428	**68.252**	(119)	69.948	(121)	69.614	(119)

Table 4.10: Tests with the Iss algorithms on random graphs. For each graph the best value ever found w_{best} is reported. For each algorithm, a histogram, the value of the best solution found; and, in parentheses, the number of colors used in the best solution are reported. For every graph and for every algorithm, three times ten runs were executed with ten different random seeds and three different values for k, namely $k^\star$, $k^\star + 2$ and $k^\star+5$, where $k^\star$ is the number of colors for which the best solution with the Fss algorithms was found in the first series of tests (see Table 4.6). The histograms are one unit wide, and each bucket is 0.05 units wide. The best value found for each line is in bold face. The cpu time allowed was 5 minutes for graphs with 100 vertices, 10 minutes for graphs with 200 vertices, 30 minutes for graphs with 300 vertices and 50 minutes for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2ghz Pentium 4 cpu and 512mb of ram.

graph	$w^\star$	w_{best}	ETB-S-AMA	AMA
mcm200_25_5	13.751	13.751	**13.751** (29)	**13.751** (29)
mcm200_30_4	16.041	16.041	**16.041** (33)	**16.041** (33)
mcm300_30_5	16.048	16.048	**16.048** (34)	**16.048** (34)
mcm300_36_6	19.262	19.262	**19.262** (41)	**19.262** (41)
mcm500_20_19	12.673	12.673	**12.673** (38)	**12.673** (38)
mcm500_49_5	25.771	25.771	28.209 (57)	**25.771** (53)

Table 4.11: Tests with a large CPU time limit for the AMA and the ETB-S-AMA algorithms on mcm-graphs with 200 vertices or more. For each graph, the optimum $w^\star$ and the best value ever found w_{best} is reported. For each algorithm and for each graph, a histogram, the value of the best solution found; and, in parentheses, the number of colors used in the best solution are reported. For every graph and for every algorithm, 5 runs with five different random seeds were executed. The histograms are one unit wide, and each bucket is 0.05 units wide. The best value found for each line is in bold face. The CPU time allowed was 1 hour for graphs with 200 vertices, 2 hours for graphs with 300 vertices and 5 hours for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	w_{best}	ETB-S-AMA		AMA	
unif01_200.2_41	8.714	8.767	(14)	**8.749**	(14)
unif01_200.2_42	8.417	**8.417**	(15)	8.506	(13)
unif01_200.2_43	8.537	8.571	(13)	**8.568**	(14)
unif01_200.5_41	17.922	**17.922**	(30)	**17.922**	(30)
unif01_200.5_42	17.119	**17.206**	(30)	17.267	(30)
unif01_200.5_43	17.409	17.531	(29)	**17.432**	(29)
unif01_200.8_41	32.033	**32.078**	(53)	32.333	(52)
unif01_200.8_42	31.469	**31.827**	(57)	32.077	(57)
unif01_200.8_43	31.152	**31.198**	(53)	31.246	(53)
unif01_300.2_41	10.951	11.164	(19)	**10.989**	(18)
unif01_300.2_42	11.041	11.270	(19)	**11.138**	(20)
unif01_300.2_43	10.761	10.981	(19)	**10.865**	(19)
unif01_300.5_41	22.938	23.532	(41)	**22.944**	(41)
unif01_300.5_42	23.579	24.187	(42)	**23.579**	(41)
unif01_300.5_43	23.384	23.815	(41)	**23.384**	(40)
unif01_300.8_41	41.557	**41.557**	(77)	41.726	(75)
unif01_300.8_42	42.742	43.369	(76)	**43.262**	(75)
unif01_300.8_43	42.440	43.249	(73)	**43.197**	(74)
unif01_500.2_41	16.427	17.223	(27)	**16.697**	(28)
unif01_500.2_42	16.214	17.152	(27)	**16.872**	(28)
unif01_500.2_43	16.650	17.485	(28)	**17.271**	(28)
unif01_500.5_41	35.917	38.243	(62)	**37.037**	(64)
unif01_500.5_42	35.333	37.700	(63)	**36.880**	(61)
unif01_500.5_43	36.730	38.766	(62)	**37.379**	(62)
unif01_500.8_41	65.438	68.238	(109)	**65.992**	(110)
unif01_500.8_42	64.534	68.985	(112)	**65.557**	(110)
unif01_500.8_43	67.428	70.104	(113)	**67.446**	(110)

Table 4.12: Tests with a large CPU time limit for the AMA and the ETB-S-AMA algorithms on random graphs with 200 vertices or more. For each graph, the best value ever found w_{best} is reported. For each algorithm, a histogram, the value of the best solution found; and, in parentheses, the number of colors used in the best solution are reported. For every graph and for every algorithm, 5 runs with five different random seeds were executed. The histograms are one unit wide, and each bucket is 0.05 units wide. The best value found for each line is in bold face. The CPU time allowed was 1 hour for graphs with 200 vertices, 2 hours for graphs with 300 vertices and 5 hours for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	$w^\star$	w_{best}	Foo-Iss	Etb-S-Iss
mcm200_25_5	13.751	13.751	**13.751** (29)	**13.751** (29)
mcm200_30_4	16.041	16.041	**16.083** (33)	16.112 (33)
mcm300_30_5	16.048	16.048	**16.048** (34)	18.888 (36)
mcm300_36_6	19.262	19.262	**19.262** (41)	22.089 (43)
mcm500_20_19	12.673	12.673	**12.673** (38)	**12.673** (38)
mcm500_49_5	25.771	25.771	**33.916** (53)	39.669 (55)

Table 4.13: Tests with a large CPU time limit for the Foo-Iss and the Etb-S-Iss algorithms on mcm-graphs with 200 vertices or more. For each graph, the optimum $w^\star$ and the best value ever found w_{best} is reported. For each algorithm and for each graph, a histogram, the value of the best solution found; and, in parentheses, the number of colors used in the best solution are reported. For every graph and for every algorithm, two times five runs with different random seeds have been executed: five runs with $k = k^\star$ and five runs with $k = k^\star + 2$, where $k^\star$ is the number of colors used in the best coloring in Table 4.11. The histograms are one unit wide, and each bucket is 0.05 units wide. The best value found for each line is in bold face. The CPU time allowed was 1 hour for graphs with 200 vertices, 2 hours for graphs with 300 vertices and 5 hours for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

graph	w_{best}	Foo-Iss		Etb-S-Iss	
unif01_200.2_41	8.714	9.618	(14)	**8.714**	(14)
unif01_200.2_42	8.417	9.069	(15)	**8.483**	(14)
unif01_200.2_43	8.537	9.400	(15)	**8.537**	(14)
unif01_200.5_41	17.922	**17.947**	(28)	18.242	(30)
unif01_200.5_42	17.119	**17.119**	(29)	17.620	(30)
unif01_200.5_43	17.409	**17.409**	(30)	17.627	(29)
unif01_200.8_41	32.033	**32.205**	(54)	32.624	(55)
unif01_200.8_42	31.469	**31.469**	(57)	32.038	(58)
unif01_200.8_43	31.152	**31.445**	(53)	32.046	(52)
unif01_300.2_41	10.951	11.755	(19)	**10.951**	(19)
unif01_300.2_42	11.041	12.689	(18)	**11.041**	(19)
unif01_300.2_43	10.761	12.359	(19)	**10.761**	(19)
unif01_300.5_41	22.938	**22.938**	(42)	23.493	(41)
unif01_300.5_42	23.579	**23.647**	(40)	24.198	(42)
unif01_300.5_43	23.384	**23.454**	(41)	24.211	(41)
unif01_300.8_41	41.557	**41.680**	(75)	42.763	(77)
unif01_300.8_42	42.742	**42.804**	(76)	43.890	(77)
unif01_300.8_43	42.440	**42.728**	(75)	44.539	(75)
unif01_500.2_41	16.427	18.352	(27)	**16.427**	(27)
unif01_500.2_42	16.214	18.408	(28)	**16.214**	(27)
unif01_500.2_43	16.650	18.762	(28)	**16.650**	(28)
unif01_500.5_41	35.917	**35.917**	(62)	36.732	(62)
unif01_500.5_42	35.333	**35.333**	(64)	35.829	(61)
unif01_500.5_43	36.730	**36.730**	(64)	37.863	(61)
unif01_500.8_41	65.438	**65.438**	(112)	70.406	(112)
unif01_500.8_42	64.534	**64.534**	(111)	71.610	(111)
unif01_500.8_43	67.428	**67.428**	(112)	72.551	(111)

Table 4.14: Tests with a large CPU time limit for the Foo-Iss and the Etb-S-Iss algorithms on random graphs with 200 vertices and more. For each graph, the optimum $w^\star$ and the best value ever found w_{best} is reported. For each algorithm and for each graph, a histogram, the value of the best solution found; and, in parentheses, the number of colors used in the best solution are reported. For every graph and for every algorithm, two times five runs with different random seeds have been executed: five runs with $k = k^\star$ and five runs with $k = k^\star + 2$, where $k^\star$ is the number of colors used in the best coloring in Table 4.11. The histograms are one unit wide, and each bucket is 0.05 units wide. The best value found for each line is in bold face. The CPU time allowed was 1 hour for graphs with 200 vertices, 2 hours for graphs with 300 vertices and 5 hours for graphs with 500 vertices. The tests were executed on Linux systems equipped with a 2GHz Pentium 4 CPU and 512MB of RAM.

Conclusion

In the first chapter, we presented our work on *oligomatic* colorings. We first introduced penta$K_{3,3}$, an oligomatic graph which was discovered during our attempts to prove that oligomatic graphs do not exist. From this discovery, we generalized the construction of penta$K_{3,3}$ to *universal graphs*. We have improved known bounds for the chromatic number of universal graphs and have determined the chromatic number and criticality of all graphs $U(k, \lambda)$ with $k \leq 10$ using theoretical arguments and the result of the PARTIALCOL heuristic. The chromatic number and criticality of some larger universal graphs are still unknown.

Since there are graphs and colorings such that every vertex sees fewer than χ different colors in its neighborhood, we investigated the question of finding how large the neighborhood must be to ensure that there is a vertex which sees χ different colors. The main result states that, if a graph is colored with $\chi + p$ colors, then there exists a vertex v such that v sees χ different colors within a distance $\lceil \frac{p}{2} \rceil + 1$. The question of whether this bound is tight or not is still open.

At the end of the chapter, we investigated the existence of oligomatic graphs in special classes of graphs, such as Halin graphs, claw-free graphs, and line graphs. Except for Halin graphs, the question (of whether oligomatic graphs exist in these classes) is only partially answered. The most intriguing open question concerns the existence or non-existence of planar oligomatic graphs. Note that oligomatic graphs are at least four-chromatic and that planar graphs are at most four-chromatic.

In Chapter Two, we have presented a technique to improve the performance of a *tabu search* by introducing several *reactive* schemes to automatically adjust the *tabu tenure*, a crucial parameter for *tabu search*. The techniques developed are simple and flexible enough such that they can be added easily to any existing implementation of tabu search. We have applied these techniques to various implementations of tabu search for solving the graph coloring problem (GCP) and the weighted graph coloring problem (WCP). We have further introduced a similarity measure to compare two graph colorings. This measure has been tested successfully for a reactive scheme and as a selection criterion in an adaptive memory algorithm for the WCP.

In Chapter Three, we presented the PARTIALCOL heuristic to solve the GCP. It is a local search heuristic based on tabu search. It uses partial solutions, which is a barely explored approach for the GCP as well as for general combinatorial optimization problems. This approach has much potential, indicated by the performance of PARTIALCOL, as well as by recent work of other researchers. PARTIALCOL is the first graph coloring heuristic which finds an optimal coloring for the DIMACS benchmark graph flat300_28_0. For further research, we suggest integrating both the PARTIALCOL and TABUCOL algorithms into an

adaptive memory algorithm to take advantage of the benefits of both approaches.

In the final chapter, we have presented a weighted version of the graph coloring problem. We have developed two heuristics based on tabu search. The first uses feasible solutions, and the second uses infeasible solutions. The advantage of the first approach is that, despite the very convenient fact that the number of colors to be used is determined by the algorithm (and not by the user), the performance of each approach is approximately equal to that of the other. We have also implemented several reactive schemes to adapt the tabu tenure. Based on the feasible solution approach, we have devised an adaptive memory algorithm. Further, we have presented several ways of generating instances for the WCP. Tests have shown that instances which have been generated to have a known optimum are generally easy to solve for most of the developed algorithms. Our adaptive memory algorithm was also capable of solving all these instances to optimality. Experiments with the developed algorithms have shown that a reactive scheme is not suited for use with an adaptive memory algorithm because the application of the local search is much too short and the reactive scheme does not have enough time to adjust the tabu tenure properly. We suggest investigation of ways to adjust the tabu tenure over several applications of the local search procedure in order to avoid the need of tuning the static tenure. Further, we suggest building an adaptive memory algorithm using both, the feasible solution approach and the infeasible solution approach.

Graph coloring is an exciting field where intriguing questions and beautiful conjectures come up faster than they can be answered or proven. We hope that the questions raised in the first chapter will stimulate further research. Graph coloring is also a wonderful playground to motivate the development of cutting-edge heuristics. We are confident that the basic ideas developed in this work for improving the efficiency of graph coloring procedures will also lead to substantial improvements in the solution methods of more general combinatorial optimization procedures. We are looking forward to seeing these ideas and procedures exploited and extended to other areas.

Bibliography

[Ada79] D. Adams. *The Hitch Hiker's Guide to the Galaxy*. Pan Books, London, 1979.

[AH77] K. Appel and W. Haken. Every Planar Map is Four Colorable. Part I. Discharging. *Illinois J. Math*, 21:429–490, 1977.

[AHK77] K. Appel, W. Haken, and J. Koch. Every Planar Map is Four Colorable. Part II. Reducibility. *Illinois J. Math*, 21:491–567, 1977.

[AHZ03] C. Avanthay, A. Hertz, and N. Zufferey. A Variable Neighborhood Search for Graph Coloring. *European Journal of Operational Research*, 151:379–388, 2003.

[AN04] Dimitris Achlioptas and Assaf Naor. The two Possible Values of the Chromatic Number of a Random Graph. In *STOC*, pages 587–593, 2004.

[AvHK+03] K. I. Aardal, S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and Solution Techniques for Frequency Assignment Problems. *4OR*, 1:4:261–317, 2003.

[BBd05] D. Bronner, I. Blöchliger, and D. de Werra. Colorations Suboptimales. Master's thesis, École Polytechnique Fédérale de Lausanne (EPFL), Recherche Opérationnelle Sud-Est (ROSE), CH-1015 Lausanne, Switzerland, March 2005.

[BdW04] I. Blöchliger and D. de Werra. On Some Properties of Suboptimal Colorings of Graphs. *Networks*, 43(2):103–108, 2004.

[Ber76] C. Berge. *Graphs and Hypergraphs*. North Holland, Amsterdam, 1976.

[BGH01] I. Blöchliger, M.U. Gerber, and A. Hertz. A new Heuristic for the Graph Coloring Problem. Master's thesis, École Polytechnique Fédérale de Lausanne (EPFL), Recherche Opérationnelle Sud-Est (ROSE), CH-1015 Lausanne, Switzerland, 2001.

[BL76] K. S. Booth and G. S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.

[Bol88] B. Bollobás. The Chromatic Number of Random Graphs. *Combinatorica*, 8(1):49–55, 1988.

[Bol01] B. Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition edition, 2001.

[Bré79] D. Brélaz. New Methods to Color Vertices of a Graph. *Communications of the Association for Computing Machinery*, 22:251–256, 1979.

[Bro41] R.L. Brooks. On Colouring the Nodes of a Network. *Proc. Cambridge Phil. Soc.*, 37:194–197, 1941.

[BT94] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6:126–140, 1994.

[Car86] Michael W Carter. A Survey of Practical Applications of Examination Timetabling Algorithms. *Oper. Res.*, 34(2):193–202, 1986.

[Car04] D. Cariolaro. *The 1-Factorization Problem and Some Related Conjectures*. PhD thesis, The University of Reading, Reading, 2004.

[Cay79] A. Cayley. On the Colourings of Maps. *Proc. Royal Geography Society*, 1:259–261, 1879.

[CH89] A.G. Chetwynd and A.J.W. Hilton. 1-factorizing Regular Graphs of High Degree – an Improved Bound. *Discrete Math*, 75:103–112, 1989.

[Cha82] G.J. Chaitin. Register Allocation and Spilling via Graph Coloring. *Proceedings of ACM SIGPLAN 82 Symposium on Compiler Construction*, pages 98–105, 1982.

[CHdW87] M. Chams, A. Hertz, and D. de Werra. Some Experiments with Simulated Annealing for Coloring Graphs. *European Journal of Operational Research*, 32:260–266, 1987.

[Cul] J. Culberson. Graph Coloring Resources. `http://www.cs.ualberta.ca/~joe/Coloring/`. Webpage.

[Cul92] J. C. Culberson. Iterated Greedy Graph Coloring and the Difficulty Landscape. Technical report, Department of Computer Sciences, University of Alberta, 1992.

[Dav91] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[DdWMP01] M. Demange, D. de Werra, J. Monnot, and V. Th. Paschos. Time Slot Scheduling of Compatible Jobs. Technical Report 182, Université Paris IX - Dauphine, Paris, 2001. To appear in Journal of Scheduling.

[DdWMP02] M. Demange, D. de Werra, J. Monnot, and V. Th. Paschos. Weighted Node Coloring: When Stable Sets are Expensive. In *WG*, pages 114–125, 2002.

[DdWMP05] M. Demange, D. de Werra, J. Monnot, and V. Th. Paschos. A Hypocoloring Model for Batch Scheduling. *Discrete Applied Mathematics*, 146(1):9–26, February 2005.

[DH98] R. Dorne and J.-K. Hao. A New Genetic Local Search Algorithm for Graph Coloring. *Lecture Notes in Computer Sciences*, 1498:745–754, 1998.

[dim] DIMACS Graph Coloring Benchmarks. `http://mat.gsia.cmu.edu/COLORING02/`. Webpage.

[dW97] D. de Werra. Restricted Coloring Models for Timetabling. *Discrete Math.*, 165-166(15):161–170, 1997.

[dWH05] D. de Werra and P. Hansen. Variations on the Roy-Gallai Theorem. *4OR*, 2005. To appear.

[Edm65] J. Edmonds. Paths, Trees, and Flowers. *Canad. J. Math.*, 17:449–467, 1965.

[EFH+86] P. Erdős, Z. Füredi, A. Hajnal, P. Komjáth, V. Rödl, and Á. Seress. Coloring Graphs with Locally few Colors. *Discrete Mathematics*, 59:21–34, 1986.

[EH68] P. Erdős and A. Hajnal. On the Chromatic Number of Infinite Graphs. In P. Erdős and G. Katona, editors, *Theory of Graphs*, pages 83–98. Academic Press, 1968. Proceedings of the 1966 Colloquium at Tihany.

[ER59] P. Erdős and A. Rényi. On Random Graphs I. *Publicationes Mathematical Debrecen*, 6:290–297, 1959.

[ERT79] P. Erdős, A.L. Rubin, and H. Taylor. Choosability in Graphs. *Congressus Numerantium*, XXVI:125–157, 1979.

[FF96] C. Fleurent and J.A. Ferland. Genetic and Hybrid Algorithms for Graph Coloring. *Annals of Operations Research*, 63:437–461, 1996.

[FHRT91] Z. Furedi, P. Hajnal, V. Rodl, and W. T. Trotter. Interval Orders and Shift Graphs. *Sets, Graphs and Numbers, A. Hajnal and V. T. Sos, eds., Colloq. Math. Soc. Janos Bolyai*, 60:297–313, 1991.

[FJS04] G. Finke, V. Jost, and A. Sebő. Batch Processing with Interval Graph Compatibilities Between Tasks. Technical Report 108, Laboratoire Leibniz-IMAG, 38000 Grenoble, France, August 2004.

[Gam86] A. Gamst. Some Lower Bounds for a Class of Frequency Assignment Problems. *IEEE Transaction of Vehicular Technology*, 35(I):8–14, 1986.

[GH99] P. Galinier and J.-K. Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[GJ79] M. Garey and D. S. Johnson. *Computer and Intractability.* Freeman, San Francisco, 1979.

[GL97] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.

[Glo89] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1:190–205, 1989.

[Glo90] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[GR92] A. Gamst and W. Rave. On the Frequency Assignment in Mobile Automatic Telephone Systems. *Proceedings of GLOBECOM'92*, pages 309–315, 1992.

[GR01] C. Godsil and G. Royle. *Algebraic Graph Theory*. Springer-Verlag, New York, 2001.

[GZ97] D. J. Guan and Xuding Zhu. A Coloring Problem for Weighted Graphs. *Inf. Process. Lett.*, 61(2):77–81, 1997.

[Ham50] R. W. Hamming. Error-Detecting and Error-Correcting Codes. *Bell System Technical Journal*, 2(29):147–160, 1950.

[Han86] P. Hansen. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. In *Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy*, 1986.

[HdW87] A. Hertz and D. de Werra. Using Tabu Search Techniques for Graph Coloring. *Computing*, 39:345–351, 1987.

[Her03] A. Hertz. Application des Métaheuristiques à la Coloration des Sommets d'un Graphe. *Traités IC2 Hermès*, 2003. Chapitre 3 dans Métaheuristiques et outils nouveaux en recherche opérationnelle.

[HGZ05] A. Hertz, P. Galinier, and N. Zufferey. An Adaptive Memory Algorithm for the Graph Coloring Problem. *Discrete Applied Mathematics*, 2005. to appear.

[HH02a] J.-P. Hamiez and J.-K. Hao. Scatter Search for Graph Coloring. *Lecture Notes in Computer Sciences*, 2310:168–179, 2002.

[HH02b] F. Herrmann and A. Hertz. Finding the Chromatic Number by Means of Critical Graphs. *Journal of Experimental Algorithmics*, 7:10, 2002.

[Hub82] Z. Hubalek. Coefficients of Association and Similarity Based on Binary (presence, Absence) Data – an Evaluation. *Biological Review*, 57:669–689, 1982.

[JAMS91] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: an Experimental Evaluation, Part II Graph Coloring and Number Partitioning. *Operations Research*, 39:378–406, 1991.

[JM82] A. Johri and D.W. Matula. Probabilistic Bounds and Heuristics Algorithm for Coloring Large Random Graphs. Technical report, Southern Methodist University, Dallas, 1982.

[JT96] D.S. Johnson and M.A. Trick. Proceedings of the 2nd DIMACS Implementation Challenge. *DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences*, 26, 1996. American Mathematical Society.

[Kar72] R. M. Karp. *Complexity of Computer Computations*, chapter Reducibility Among Combinatorial Problems, pages 85–103. Plenum Press, New York, 1972.

[Kel66] G. Keller. *Hesch gseh?* Private communication, 1866. Seldwyla.

[Kob99] D. Kobler. *Modèles Biologique en Optimisation Combinatoire et Modèles Mathématiques en Génétique*. PhD thesis, École Polytechnique Fédérale de Lausanne, Département de Mathématiques, 1999.

[KPS04] J. Körner, C. Pilotto, and G. Simonyi. Local Chromatic Number and Sperner Capacity. *Submitted*, 2004.

[Lei79] F.T. Leighton. A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of Research of the National Bureau Standard*, 84:489–505, 1979.

[Mer96] M. Meringer. Erzeugung Regulärer Graphen. Master's thesis, Universität Bayreuth, 1996.

[Mer99] M. Meringer. Fast Generation of Regular Graphs and Construction of Cages. *Journal of Graph Theory*, 30:137–146, 1999.

[Min80] G. J. Minty. On Maximal Independent Sets of Vertices in Claw Free Graphs. *J. Combin. Th. B*, 28:284–304, 1980.

[MMT05] E. Malaguti, M. Monaci, and P. Toth. A Metaheuristic Approach for the Vertex Coloring Problem. Technical Report OR/05/3, DEIS – University of Bologna, Italy, 2005.

[Mor96] C. Morgenstern. Distributed Coloration Neighborhood Search. *Discrete Mathematics and Theoretical Computer Science*, 26:335–358, 1996. American Mathematical Society.

[MPdW+04] J. Monnot, V. Th. Paschos, D. de Werra, M. Demange, and B. Escoffier. Weighted Coloring on Planar, Bipartite and Split Graphs: Complexity and Improved Approximation. In *ISAAC*, pages 896–907, 2004.

[MT96] A. Mehrotra and M. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal On Computing*, 8(4):344–354, 1996.

[NT74] G. A. Neufeld and J. Tartar. Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem. *Commun. ACM*, 17(8):450–453, 1974.

[NT01] D. Nakamura and A. Tamura. A Revision of minty's Algorithm for Finding a Maximum Weight Stable set of a Claw-free Graph. *J. Oper. Res. Soc. Japan*, 44:194–204, 2001.

[RT95] Y. Rochat and E. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1:147–167, 1995.

[Sch04] D. Schindl. *Some Combinatorial Optimization Problems in Graphs with Applications in Telecommunications and Tomography.* PhD thesis, École Polytechnique Fédérale de Lausanne, Section de Mathématiques, 2004.

[Slo] N. J. A. Sloane. Dedekind Numbers, Sequence A000372. `http://www.research.att.com/projects/OEIS?Anum=A000372`. On-Line Encyclopedia of Integer Sequences, Website.

[SU97] E. R. Scheinerman and D. H. Ullman. *Fractional Graph Theory.* Wiley-Interscience Series in Discrete Mathematics and Optimization, Chichester, john wiley and sons edition, 1997.

[Vas02] M. Vasquez. Arc-Consistency and Tabu Search for the Frequency Assignment Problem with Polarization. *Proceedings of CPAIOR'02, Le Croisic, France*, pages 359–372, March 2002.

[VH01] M. Vasquez and J.-K. Hao. A Logic-Constrained Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite. *Comput. Optim. Appl.*, 20(2):137–157, 2001.

[Viz64] V.G. Vizing. On an Estimate of the Chromatic Class of a p-graph. *Metody Diskret. Analiz.*, 3:9–17, 1964.

[Viz76] V.G. Vizing. Coloring the Vertices of a Graph in Prescribed Colors. *Metody Diskret. Anal. v Teorii Kodov i Schem*, 29:3–10, 1976. In russian.

[WP67] D. J. A. Welsh and M. B. Powell. An Upper Bound on the Chromatic Number of a Graph and its Application to Timetabling Problems. *The Computer Journal*, 10:85–86, 1967.

[Zuf02] N. Zufferey. *Heuristiques pour les Problèmes de la Coloration des Sommets d'un Graphe et d'Affectation de Fréquences avec Polarités.* PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, 2002.

Curriculum vitae

Ivo Blöchliger
June 11, 1976

Ch. de Montelly 43c
1007 Lausanne

+41 79 730 59 91
ivo@bloechligair.ch

Ingénieur mathématicien diplômé EPFL 2001
single, swiss citizen

Education

2001-2005	PhD student of Prof. de Werra in Operations Research at EPFL
2004	Swiss paragliding license for passenger flights
1996-2001	Studies in Mathematics at EPFL
1998-1999	Exchange student at the University of Texas at Austin (UT)
1991-1996	Kantonsschule am Burggraben, St. Gallen, Matura type C (scientific)

Languages

German	native language
French	spoken fluently, average writing skills
English	spoken fluently, average writing skills
Italian	intermediate

Professional Experience

2001-2005	Teaching Assistant and System Administrator (Linux / Windows) for the ROSE research group at EPFL
2001-2004	Project Leader and Programmer for the GNUWin project, a collection of open source software for Windows, available in many languages on CD and online.
2001-2004	Founding President of EPFL's student association *GNU Generation* whose purpose is to promote the use and development of open source software.
2000	Development and implementation of an optimization algorithm at *Portalys SA*, PSE Lausanne.
1997-2000	Teaching Assistant at EPFL in computer science and math (Pascal and C programming, linear algebra, matlab, analysis).
1997, 1998	Teacher of one-week math courses at the Kantonsschule am Burgraben, St. Gallen

Conference Presentations

2005	Weighted Vertex Colorings, *Third Joint Operations Research Days*, Rüschlikon, Switzerland, September 2005.
2004	Color-Blind Graphs and Suboptimal Colorings, *Latin-American Conference on Combinatorics, Graphs and Applications (LACGA '04)*, Santiago, Chile, August 2004.
2004	A Reactive Tabu Search Using Partial Solutions for the Graph Coloring Problem, *CORS-INFORMS Meeting 2004*, Banff, Canada, May 2004.
2004	Color-Blind Graphs: Properties of Suboptimal Colorings, *Troisieme Cycle Romand*, Zinal, Switzerland, March 2004.
2003	A Reactive Tabu Search Using Partial Solutions for the Graph Coloring Problem, *Dagstuhl Seminar 03391 "Graph Colorings"*, Dagstuhl, Germany, September 2003.
2003	A Partial Feasible Solution Approach to Vertex Coloring, *ECCO XVI*, Molde, Norway, June 2003.
2002	On Some Properties of Non-Optimal Colorings of Graphs, *EPSRC Interdisciplinary Scheduling Network 2nd PhD Student Workshop on Scheduling*, London, England, September 2002.
2002	A Penalty Evaporation Heuristic for the Graph Coloring Problem, *ECCO 2002*, Lugano, Switzerland, June 2002.

Additional Publications

I. Blöchliger. Color-blind Graphs and Suboptimal Colorings. *Electronic Notes in Discrete Mathematics*, 18:37–40, 2004. Extended abstract for the *Latin-American Conference on Combinatorics, Graphs and Applications (LACGA'04), Santiago, Chile.*

I. Blöchliger. Modeling Staff Scheduling Problems: a Tutorial. *European Journal of Operational Research*, 158:533–542, 2004.

I. Blöchliger and V. Rezzonico. Historique de GNU Generation. *Flash Informatique*, 9:3–4, 2004.

Computer Skills

Programming	C, C++, MPI, PLAPACK, Perl, Bash, Pascal
Web	HTML, JavaScript, CGI, MySQL
OS	Linux, Unix
Miscellaneous	LaTeX, POV-Ray, Gimp

Spare Time

Paragliding (semi-professional), piano, squash and badminton