UNIT DISK GRAPH COLORING AND ITS REOPTIMIZATION

by

Arman Boyacı

B.S., Industrial Engineering, Galatasaray University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2009

UNIT DISK GRAPH COLORING AND ITS REOPTIMIZATION

APPROVED BY:

Assist. Prof., Tınaz EKİM AŞICI (Thesis Supervisor)	.....
Prof., İ. Kuban ALTINEL	.....
Prof., Marc DEMANGE	.....

DATE OF APPROVAL: ... / ... / .....

ACKNOWLEDGEMENTS

First of all, I am heartily thankful to my supervisor, Assist. Prof. Dr., Tınaz EKİM AŞICI, whose encouragement, guidance and support from the beginning to the end provided me perfect conditions to work in such an intensive study. I am grateful to her for being always accessible and willing to help me. Although it was fatiguing (for both of us), I must say that I enjoyed myself during all this period.

I thank my fellow friends in Boğaziçi University for their support, stimulating discussions and all the fun we have had in the last two years.

In addition, I am thankful to TUBITAK for the uninterrupted financial assistance throughout my master degree.

Finally, I wish to extend my warmest thanks to my parents for their support and to all who really care about me.

ABSTRACT

UNIT DISK GRAPH COLORING AND ITS REOPTIMIZATION

A unit disk graph (UDG) is a graph of intersection of a set of unit disks in Euclidean plane. Motivated by frequency assignment problem in telecommunication, we consider minimum vertex coloring of unit disk graphs (ColorUDG) and its reoptimization which are both NP-hard problems. In the first part of the study, the quality of lower and upper bounds on the optimal value of ColorUDG are investigated by conducting empirical tests. Maximum clique is a lower bound for the minimum coloring. Based on some observations, running time improvement is achieved on the existing $O(n^{4.5})$ maximum clique algorithm. On the other hand, we derive upper bounds by some simple heuristics (sequential algorithms). Two new construction heuristics and a new improvement method are proposed. In the second part, vertex adding/removing reoptimization problems for ColorUDG are defined. Despite the knowledge of the optimum solution of the old instance, we showed that both problems remains NP-hard, therefore some heuristic methods are proposed. The developed reoptimization algorithms, which perform successfully for many instances when applied to randomly generated graphs. Moreover, we modified Brelaz's sequential coloring algorithm to solve exactly vertex adding reoptimization problem.

ÖZET

BİRİM DİSK ÇİZGE BOYAMA VE YENİDEN ENİYİLENMESİ

Bir takım birim diskin Öklid düzleminde kesişimleri birim disk çizgedir(BDÇ). Özellikle telekomünikasyondaki frekans atama probleminin çözümü için önemli ve NP-zor problemler olan birim disk çizge boyama(BDÇBoya) problemini ve yeniden eniyilenmesini ele aldık. Çalışmanın ilk kısmında, deneysel testler gerçekleştirerek BDÇBoya probleminin alt ve üst sınırlarının kalitesi incelendi. Bazı gözlemlere dayanarak, mevcut $O(n^{4.5})$ klik algoritmasının ortalama koşu zamanında iyileştirme sağlandı. Ek olarak, iki yeni boyama sezgiseli ve Kempe'nin Zincirlerinden ilham alınarak yeni bir iyileştirme yöntemi önerildi. İkinci kısımda ise, köşe ekle/çıkar yeniden eniyileme problemleri tanımlandı. Eski örneğe ait eniyi çözüm bilgisine rağmen bu problemlerin NP-zor kaldıklarını gösterdik ve bu nedenle sezgisel yöntemler önerdik. Bu sezgisellerin ortalama performansları, deneysel çalışmalar ile tayin edildi. Ayrıca Brelaz'ın ardışık boyama algoritmasını köşe ekle yeniden eniyileme problemini çözebilecek şekilde uyarladık.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

$C_i^j(v)$	Maximal connected component of G containing vertex v and vertices colored i and j such that $N_{C_j^i}(v)$ has only vertices of color j.
$\chi(G)$	Chromatic number of G
$\omega(G)$	Clique number of G
FAP	Frequency Assignment Problem
DG	Disk Graph
UDG	Unit Disk Graph
CG	Coin Graph
GSM	Global system for mobile
TSP	Traveling salesman problem
SC	Scanning coloring algorithm
DC	Decreasing Order coloring algorithm
CC	Clique coloring algorithm
BC	Balanced coloring algorithm
iSC	Scanning coloring followed by the improvement algorithm
iDC	Decreasing Order coloring followed by the improvement algorithm
iCC	Clique coloring followed by the improvement algorithm
iBC	Balanced coloring followed by the improvement algorithm
CCC	Check color class algorithm
iCCC	Check color class followed by the improvement algorithm
rFF	First-fit for reoptimization algorithm
irFF	First-fit for reoptimization followed by the improvement algorithm

1. INTRODUCTION

Nowadays, wireless communication is indispensable, since it is used in many important domains such as mobile telephones, wireless LANs, satellite communication and military operations. A frequency assignment problem (FAP) arises in each of these applications. In the literature, different modeling approaches are available. Since there is no known polynomial time algorithm to solve FAP, various solution techniques are proposed. Some researchers have suggested modeling and solving FAP with graph theoretical tools. Then, FAP is modeled in the following way: transmitters/receivers are represented by vertices, there exist an edge between two vertices if there is a possible interference between the signals of the corresponding transmitters/receivers. The objective of FAPS being to minimize the number of frequencies used, it can be reduced to the minimum vertex coloring problem on this graph; vertices having the same color indicate transmitters/receivers that will be assigned the same frequency. However, it is known that minimum vertex coloring on an arbitrary graph is NP-hard (1). It is known that NP-hard problems may possibly become polynomially solvable under some restrictions. For FAP, we typically work on hexagonal graphs or on disk graphs(DG). A disk graph is the graph of intersection of a set of disks in the Euclidean plane. Therefore, this simplification requires the following assumption: each transmitter has a circular area of effect as a function of its power. If we also assume that all the transmitters have the same power, corresponding graph is a unit disk graph (UDG). Due to this motivation, UDG are one of the well-studied graphs in the literature.

However, it is known that vertex coloring problem in UDG remains NP-hard (2). One of the ways to handle an NP-hard problem is to find good lower and upper bounds for the value of its optimum solution. The optimum value of the vertex coloring problem, chromatic number can be bounded with clique number and a feasible coloring. It is shown that clique number of a UDG can be determined in polynomial time. On the other hand, various heuristics can be used to generate a feasible coloring. The quality of a bound can be assessed according to two criteria: (1) worst-case and (2) average-case. For the worst-case analysis, theoretical bounds and worst-case example

constructions are used. On the other hand, empirical tests can be conducted to predict the average performance of an algorithm in practice.

In this thesis, we will also be interested in some reoptimization problems related to unit disk graph coloring. In general, reoptimization of a problem can be defined as follows. Given an instance with an optimum solution and a local modification on the instance, we try to determine the optimum solution of the new instance. In that case, optimum solution of the old solution could be helpful to find the new one. Intuitively, this additional information should help us to develop better algorithms.

1.1. Outline of the thesis

Section 2 begins with some basic graph theoretical definitions, which will be necessary throughout the thesis. Then, we continue with defining frequency assignment problem and giving its mathematical formulation. Finally, a short survey on graph theoretical approaches for FAP is presented.

Section 3 is dedicated to UDG. We first start by presenting some properties of UDG and state some basic observations which will be useful in the next sections. Then, we focus on graph problems in UDG; complexity results on the principal problems we deal with are presented. For NP-hard problems, we present existing approximation algorithms.

Since the coloring problem in UDG is NP-hard, in Section 4, we mainly investigate the quality of bounds for the chromatic number. Knowing that the clique number provides us a lower bound, some improvements on the polynomial time maximum clique algorithm are presented. Besides, for generating feasible colorings, construction and improvement heuristics are developed and implemented. Lastly, the details on instance generation process and empirical results on the test set are given.

In Section 5, reoptimization of unit disk graph coloring is considered. First, two reoptimization problems, one dealing with vertex addition, the other one with vertex

removal, are formally defined and their NP-hardness are shown. Since the problems remains NP-hard, heuristic methods are developed and empirical tests are conducted on a new test set. Moreover, a modified exact sequential coloring algorithm is proposed for the reoptimization.

1.2. Contribution of this study

We mainly consider two problems: UDG coloring and its reoptimization. In the first part, based on some observations, we give a modified maximum clique algorithm for UDG, which is significantly faster on empirical tests. We propose new construction heuristics. In addition, an improvement method inspired from Kempe's Chains is developed. According to the empirical study conducted, our implementation of improved heuristic outperforms construction methods. Optimality is mostly reached on sparse graphs. In the second part of the study, we focused on reoptimization of UDG coloring. Vertex adding and removing situations are considered. We showed that both problems are NP-hard and heuristic methods are proposed. According to the empirical study, we can say that our improvement method performs well. Lastly, we also modified an exact sequential coloring algorithm for the reoptimization case.

2. BASIC NOTIONS

This section starts with some standard definitions on graph theory. For a complete source, we recommend to look at the book about graph theory by Berge (3). In the second part, some basic informations are given about frequency assignment which we consider as the primary motivation for the minimum vertex coloring in UDG.

2.1. Graph Theoretical Definitions

A *graph* $G = (V, E)$ is a set of *vertices* V and *edges* E. An edge between two vertices u and v is denoted edge uv. We note $n = |V|$ and $m = |V|$. We say u is *adjacent* to v if uv exists. $N(v)$, the *neighborhood* of a vertex v is the set of all vertices adjacent to v. *Degree* of v is defined as the number of adjacent vertices to v. An *isolated vertex* is a vertex with degree zero.

An *induced subgraph* of G is a subset of vertices of G together with any edges whose endpoints are in this subset. We define $N_H(v)$ as the neighborhood of v in the induced subgraph H. *Complement graph* $\overline{G}$ of a graph $G = (V, E)$ is a graph having same vertex set V and two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G.

A graph is called *dense* if the number of edges is close to the maximal number of possible edges which is $\frac{n(n-1)}{2}$. On the other hand, a *sparse* graph contains relatively few edges. *Density* of a graph is calculated as follows: $\frac{2m}{n(n-1)}$.

A *clique* is a subset of vertices, all pairs of which are adjacent. A *k-clique* is a clique of size k. *Clique number*, $\omega(G)$, is the size of the biggest clique in a graph G. A *stable (independent) set* is a subset of vertices, no two of which are adjacent. A *vertex coloring* is an assignment of colors to the vertices of a graph in a such way that no two adjacent vertices share the same color. *Chromatic number*, $\chi(G)$, is the minimum number of colors necessary to color a graph G.

A clique is *maximal* (inclusion wise) if no more vertex can be included to that clique set. A clique is *maximum* (size wise) if there is no larger clique in the graph. Note that a maximum clique is also maximal, but not necessarily vice-versa.

A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets A and B such that every edge connects a vertex in A to one in B; that is, A and B are independent sets. $K_{a,b}$ is a complete (all possible edges exist) bipartite graph where two disjoints sets are size respectively a and b. A *planar graph* is a graph which can be drawn on the plane in such a way that none of the edges intersect. A *perfect graph* is a graph in which the chromatic number of every induced subgraph equals the clique number of that subgraph.

2.2. Frequency Assignment Problem

Frequency assignment problem(FAP) is first defined in the 1960s (4) and become popular in 1990s and 2000s mainly due to the fast growing demand on wireless telephone networks and implementation of satellite communication projects. (5) and (6) are good literature surveys on FAP which summarize main problems, assumptions and methods on the topic. Different FAP models having different optimization criteria and constraints are proposed, however they share following common features: (1) A set of wireless communication connections must be assigned frequencies from a set of available frequencies of that connection point. (2) Interference of two signals occur if connections are geographically close to each other and if the frequencies of the signals are close on the electromagnetic band. In this part of the study, we will first look at the practical aspects of Frequency Planning and then different mathematical models and corresponding assumptions of these models will be discussed. Finally, graph theoretical approaches for FAP will be stated.

2.2.1. Preliminaries

Wireless communication between two points is established with the use of a transmitter and a receiver. When two transmitters use close frequencies, their signals may

interfere. The quality of the signal depends on different parameters such as distance between transmitters, the geographical position of the transmitters, the power of the signal and the weather condition.

The commercially usable radio spectrum is very scarce. Therefore, frequencies are reused by many transmitters. We can say that planning of the frequency assignments are indispensable for achieving a high performance in a network. The selection of frequencies such that interference is avoided or minimized is called FAP. There exist two main approaches to model this problem:

1. The range of the available frequencies is fixed and the interference is minimized,
2. The interference is strictly avoided, the required range of frequencies is minimized.

In this work, we mainly consider the second model.

The availability of frequencies from the radio spectrum is regulated by national governments and world-wide by International Telecommunication Union(ITU). Frequency bands $[f_{min}, f_{max}]$ are usually partitioned in a set of channels of same bandwidth Δ. $F = \{1, \ldots, N\}$ denotes available channels where $N = (f_{max} - f_{min})/\Delta$. Some channels can be forbidden for a given transmitter. For instance, if the transmitter is close to the border of a country. As a consequence, the channels available for a transmitter v forms a subset $F(v)$ of F.

The placement of the base stations as well as the selection and configuration of antennas are the basis for delivering the desired network coverage. The decision on how many transmitters to be operated in each cell depends on the capacity requirements of the network. For instance, a base station can contain 8 antennas having 12 transmitters on each of them. The level of interference rapidly decreases with distance between the frequencies (assumed to be inversely proportional to the distance). Therefore, frequencies of transmitters sharing the same antenna and being in the same base station must differ by at least some value, referred to *co-cell* separation and *co-site separation*, respectively.

2.2.2. Mathematical Models

In the previous section, we stated some practical aspects of FAP. Now, a mathematical formulation for FAP will be presented. Depending on the objective function and the constraints, there exist different formulation for FAP (5). Here, we present one of the integer programming formulations for FAP, called the Minimum Order FAP. When mobile phones were first introduced in 70s, frequencies were sold per unit and they were expensive. As a consequence, researchers have proposed the Minimum Order FAP formulation where we penalize the usage of frequencies. In other words, the total number of frequencies used is minimized under the constraint that all interferences are avoided. This mathematical formulation for FAP consists of the following set of variables, constraints and an objective function.

$$x_{vf} = \begin{cases} 1 & \text{if frequency } f \in F(v) \text{ is assigned to vertex } v \\ 0 & \text{otherwise} \end{cases}$$

$$y_f = \begin{cases} 1 & \text{if frequency } f \in F \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$\min \quad \sum_{f \in F} y_f \tag{2.1}$$

$$s.t. \quad x_{vf} \leq y_f \qquad \forall v \in V, \quad f \in F(v) \tag{2.2}$$

$$\sum_{f \in F(v)} x_{vf} = m(v) \qquad \forall v \in V \tag{2.3}$$

$$x_{vf} + x_{wg} \leq 1 \qquad \forall \{v, w\} \in V, f \in F(v), g \in F(w) : p_{vw}(f, g) > p_{max} \tag{2.4}$$

$$x_{vf} \in \{0, 1\} \qquad \forall v \in V, f \in F(v) \tag{2.5}$$

$$y_f \in \{0, 1\} \qquad \forall f \in F \tag{2.6}$$

Objective 2.1 minimizes the total number of frequencies used in the network. Constraints 2.2 ensures that used frequencies are accounted in the objective function. The requirement that $m(v)$ frequencies have to be assigned to a vertex v is modeled in constraints 2.3. For each pair of frequencies $f \in F(v)$ and $g \in F(w)$ a penalty parameter is given $p_{vw}(f, g)$. p_{max} is the threshold parameter for admissible interference value. Interference is avoided with constraints 2.4.

Note that, in the frequency assignment problem considered in the following sections (and which is modeled as a vertex coloring problem in UDG), we always assume that interference is only possible for pairs of transmitters/receivers located under some threshold distance d one from another and having exactly the same frequency. More technically, we assume $p_{uw}(f, g) = 1$ if and only id $d(u, w) < d$ and $f = g$, in addition we take $p_{max} = 0$ which means 0 interference is tolerated.

2.2.3. Graph Theoretical Approaches

For a given network, the interference can be represented as follows: Let $G = (V, E)$ be an interference graph; each antenna is represented by a vertex $v \in V$, two vertices for which the corresponding signals may interfere for at least one pair of frequencies are connected by an edge. Since each antenna consists of multiple transmitters,

each vertex can be split into as many vertices as the transmitters carried by the corresponding antenna. This graph is referred to the *split interference graph*. The optimal coloring of an interference graph can be considered as a lower bound for FAP, since we neglect some other requirements of FAP. For instance, not only same frequencies cannot be assigned to neighboring vertices, but also certain distance between frequencies have to be obeyed. In this manner, a generalization of the vertex coloring problem called *T-coloring* is introduced by Hale (7) where the colors of adjacent vertices should differ by at least T. Another practical aspect that should be considered is the presence of blocked channels at some transmitters. This version of coloring is known as *list-coloring*; each vertex can take a color from a given list of colors. List coloring is first discussed by Vizing (8), then Tesman (9) combined T-coloring and list-coloring problems. See also (6) where different mathematical models for FAP are given and variants of some traditional coloring heuristics to solve these models are presented.

3. DISK GRAPHS

A disk graph(DG) is the graph of intersection of a set of disks in the Euclidean plane. There are many applications that use disk graph modeling to represent the corresponding problem. Consequently, different types of graph problems are considered in these applications. For example,

- in chemistry, testing the physical feasibility of a graph-modeled molecular (recognition problem),
- in telecommunication, making frequency assignments for the transmitter/receiver stations (coloring problem),
- selecting a minimum number of transmitters so that all other stations are within the range one of the chosen transmitters (domination problem),
- in facility location, placing facilities to given locations where the proximity of facilities is undesirable (independent set).

With the high increasing demand in GSM telecommunication, the frequency assignment problem became notably important. This area of research is well-studied in the literature. (7) and (5) are some surveys in this topic.

In this subclass of graphs, most of the graph problems remain unfortunately NP-hard. In other words, for such problems there is no known polynomial time algorithm to solve them optimally in DG. However, since there is a strong motivation to study these graphs, approximation algorithms are suggested to generate relatively good solutions in a reasonable time.

Three equivalent models can be defined to represent various problems as DG. In *intersection model*, each vertex corresponds to a disk on a plane and an edge appears between two vertices when the corresponding disks intersect. We assume that tangent disks intersect. In *containment model*, n disks in the plane form a graph with n vertices corresponding to n disks and an edge between two vertices if one of the corresponding

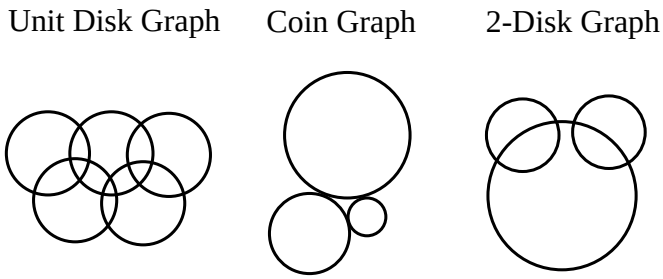


Figure 3.1. Subclasses of Disk Graphs

disks contains the other's center. Finally, a purely geometric definition is also available. n points in the plane form a graph with n vertices corresponding to n points and there is an edge between two vertices if and only if the Euclidean distance between two corresponding points is at most some specified bound d. This last model is referred to *proximity model*. Transforming between intersection and containment models can be easily done by simply doubling or halving the diameter of disks. Transforming between the intersection and proximity models requires only an identification of the disks centers with the points in the plane and the disk diameter with d. Therefore, given any of the three models, the other two can be produced in linear time. In what follows, we use intersection model.

Relevant subclasses of DG are the followings: coin graphs (CG), σ-disk graphs (σ-DG), unit disk graphs (UDG). In CG, disks are not allowed to overlap but to touch. In σ-DG, diameter ratio of disks is bounded by some constant σ. A disk graph is called *unit disk graph* if all disks' diameters are equal.

Some subclasses of UDG relevant with respect to FAP can be stated as follows. A *grid graph* is a unit disk graph in whose intersection model all disks have centers with integer coordinates and radius $\frac{1}{2}$. On the other hand, a *fully triangulated grid graph* is a unit disk graph in whose intersection model all disks have centers with integer coordinates and radius $\frac{\sqrt{2}}{2}$, see Figure 3.2.

Given a UDG, we can transform it to (1) a weighted grid graph or (2) a weighted fully triangulated grid graph by moving all disks to the nearest integer coordinate and by setting the radius of this super-disks to respectively $\frac{1}{2}$ and $\frac{\sqrt{2}}{2}$. Let G be a UDG.

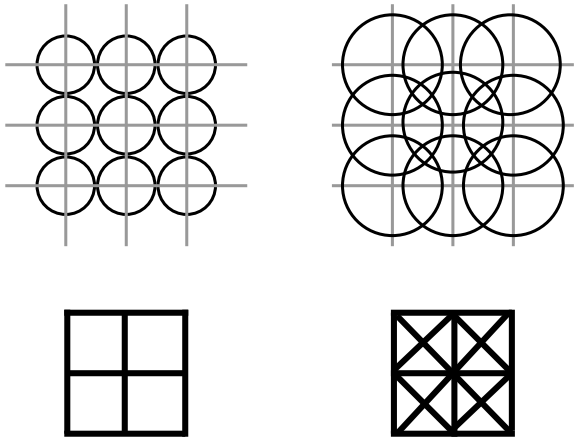


Figure 3.2. Grid Graph and Fully Triangulated Grid Graph

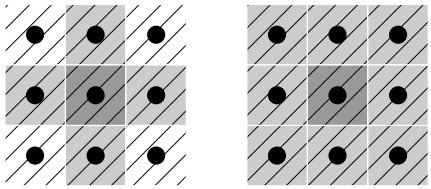


Figure 3.3. Neighbors of the central vertex for a grid graph and a fully triangulated grid graph

We can easily see that the first transformation will add new edges and possibly remove some existing edges. On the other hand, if we apply the second transformation, all the existing edges will remain and some new edges will be added. Let G' be the weighted fully triangulated grid graph obtained using the second transformation to G, then $\chi(G')$ can only increase with respect to $\chi(G)$. It means that $\chi(G')$ constitutes an upper bound for the $\chi(G)$, see on Figure 3.3. However, according to our implementation, we observed that this is not a good approximation for our considered problem. Simply because too many non-existing edges arise. Consequently, we stop our investigation on the complexity of minimum coloring problem on fully triangulated grid graphs.

We know that most graph theoretical problems are easy to solve in grid graphs even for weighted situation (2). However, as far as we know, there does not exist any complexity results for fully triangulated grid graphs. One can easily state that the maximum weighted clique problem in weighted fully triangulated grid graphs can be computed in polynomial time since each maximal clique in G has at most 4 vertices.

So they can be enumerated in polynomial time. On the other hand, the complexity of 3-colorability of a weighted fully triangulated grid graph is an open question.

Obviously, given a disk graph, there is no unique way to place disks on the plane. In some applications, coordinates of disks could be available. This additional information can allow us to develop better algorithms. Given a disk graph with the coordinates of disks will be referred to *a graph with given representation*. All of our analysis will be done for both cases. In other words, complexity results as well as the performance of algorithms will be analyzed on graphs with and without representation. Disks and the vertices representing the disks will be used interchangeably when no confusion arise.

This section is structured in the following way. First, we will enumerate some important properties of disk graphs. These properties will provide us some tools to make our analysis for the approximation algorithms. Secondly, we will discuss the complexity of some graph theoretical problems. We will show that most of the problems are NP-hard in DG. Therefore, next section will cover related approximation algorithms. Starting by defining what an approximation algorithm is, we will state approximation algorithms for various problems in DG. Moreover, we will prove the performance guarantee of these algorithms.

3.1. Characteristics

In the beginning, we will do some basic observations that we will make use of in our analysis. Therefore, in this section we will specify important properties of DG, (10).

Property 3.1.1. *Unit disk graphs are not perfect.*

Proof. Any odd cycle of length five or greater is a UDG. By Strong Perfect Graph Theorem (11), UDG are not perfect. □

Property 3.1.2. *Unit disk graphs are not planar. Since a clique of size* 5 *or greater*

is not planar, UDG are not planar.

Proof. Any clique of size five or greater is a UDG. □

Obviously these properties are true also for DG. It means that DG are not planar nor perfect. Therefore, unfortunately none of the polynomial time algorithms for perfect graphs will be available for UDG and clearly nor for DG.

Property 3.1.3. *Coin graphs are planar.*

Property 3.1.4. *Every triangle-free UDG is planar.*

Proof. (By contrapositive) Let G be a non-planar UDG. Select two intersecting edges, ab and cd. Let a,b,c and d centers of these four disks and e be the intersection point. We denote $r(u)$ as the radius of the disk u.

By triangular inequality:

$$ac + bd \leq (ae + ec) + (be + ed) \tag{3.1}$$
$$= (ae + be) + (ec + ed) \tag{3.2}$$
$$= ab + cd \tag{3.3}$$

We know that $(u, v) \in E$ iff $uv \leq r(u) + r(v)$. Hence,

$$ac + bd \leq ab + cd \tag{3.4}$$
$$\leq (r(a) + r(b)) + (r(c) + r(d)) \tag{3.5}$$
$$= (r(a) + r(c)) + (r(b) + r(d)) \tag{3.6}$$

It means that either $ac \leq r(a) + r(c)$ or $bd \leq r(b) + r(d)$. So either $(a, c) \in E$ or

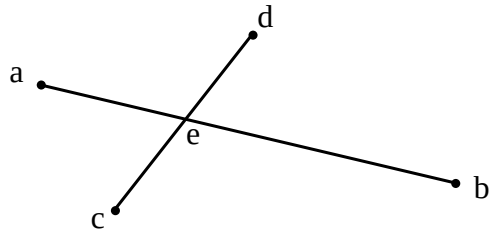


Figure 3.4. Representation of two crossing edges for a non-planar graph

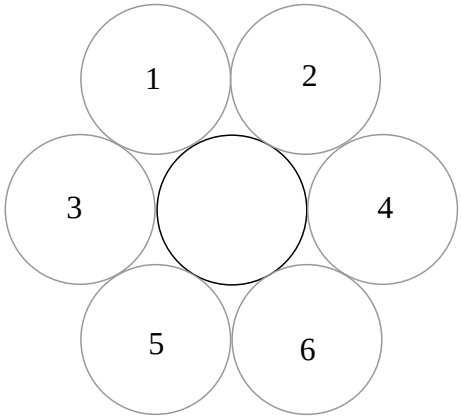


Figure 3.5. A unit disk having six neighborhood disks

$(b, d) \in E$. Similarly $ad + bc \leq ab + cd$. So, either $(a, d) \in E$ or $(b, c) \in E$. In total there four possibilities, any of them implies a triangle.

□

Property 3.1.5. *Let C be a disk of radius r and and let S be a set of disks of radius r such that every disk in S intersects C and no two disks in S intersect each other. Then $card(S) \leq 5$.*

Proof. Suppose $|S| \geq 6$. Let s_i, $1 \leq i \leq 6$, denote the centers of any six disks in S. Let c denote the center of C. Denote the ray $\overrightarrow{cs_i}$ by r_i $(1 \leq i \leq 6)$. Since there are six rays emanating from c, there must be at least one pair of rays r_j and r_k such that the angle between them is at most $\frac{\pi}{3}$. Now, it can be verified that the distance between s_j and s_k is at most $2r$, which implies that disks centered at s_j and s_k intersect, contradicting our assumption. □

Property 3.1.6. *Let G be a unit disk graph, and let v be a vertex such that the unit disk corresponding to v (in some model for G) has the smallest X-coordinate. The size*

of a maximum independent set in $G(N(v))$ is at most 3.

Property 3.1.6 is a natural extension of Property 3.1.5. It can be easily seen on Figure 3.5 that if a unit disk has a smallest X-coordinate, it cannot have more than 3 independent neighborhood.

Property 3.1.7. *Every induced subgraph of a unit disk graph is also a unit disk graph.*

Property 3.1.6 and 3.1.7 will be frequently used in analyzing approximation algorithms in the following sections. An example of a graph that is not a unit disk graph is the star $K_{1,6}$ with one central vertex connected to six leaves: if each of six unit disks touches a common unit disk, some two of the six disks must touch each other.

Property 3.1.8. *For any vertex v of a UDG, degree of v is bounded by $6\omega(G) - 6$.*

Proof. For a given vertex v, $N(v)$ can be divided into 6 equal regions of $\frac{\pi}{3}$. We already stated that disks sharing the same region are all intersecting. Then we can place in each region at most $\omega(G) - 1$ vertices. □

Property 3.1.9. *Let G be a unit disk graph, v be a vertex and $N_i(v)$ be the neighborhood of v colored with color i. Then $|N_i(v)| < 5$.*

Property 3.1.9 is a result of Property 3.1.5 and it will be useful in the complexity analysis of Chains algorithm that will be defined in the following chapter.

Another natural question for UDG is whether for a fixed clique number, there exist a UDG with arbitrarily large chromatic number. If not it means that there is a bound $\chi \leq c \cdot \omega$ where c is a constant. In such a case the challenge is to determine the minimum value of constant c. Cycles are UDG, therefore c is at least $\frac{3}{2}$. *Mycielski graphs* are well known triangle-free graphs with arbitrarily large chromatic number, (12). A Mycielski graph of chromatic number i, M_i is constructed iteratively in the following way:

1. Start with a graph consisting of two adjacent vertices and assign $j = 2$.
2. Replicate all existing vertices and connect replicated vertices with neighbors of the original vertices.
3. Add a central vertex and connect it to all replicated vertices.
4. $j \leftarrow j + 1$
5. Repeat 2,3,4 until $j = i$.

M_2 is a K_2 (a clique of size 2) and M_3 is a C_5 (cycle of size 5) which are also UDG. However M_i for $i \geq 4$ are not UDG.

Claim 3.1.10. *Mycielski graphs M_i for $i \geq 4$ are not UDG.*

Proof. M_4 is triangle-free but not planar. We know that triangle-free UDG are planar, see Property 3.1.4. Therefore M_4 is not a UDG. For $i \geq 5$, M_i contains induced $K_{1,6}$ which is an obstruction graph for UDG. Therefore M_i, $i \geq 4$ are not UDG. □

In the next sections, we will introduce some coloring algorithm for coloring UDG. One of them uses at most $3\omega(G) - 2$ colors, so, $\chi(G) \leq 3\omega(G) - 2$ and this bound is tight, see (13). This means that the minimum c value that we are looking for is at most three. On the other hand this value is at least $\frac{3}{2}$ since such a graph can be constructed using an odd cycle and arbitrarily large chromatic number can be obtained by replicating all the existing points in the same spot the same number of times. We can try to construct worst-case examples for higher c values like 2 however the search for exact value of minimum c value is an open question.

3.2. Complexity Results

Most of the graph problems remain NP-hard in unit disk graphs. Table 3.1 presents a summary of complexity results in this domain. It can be seen that all the problems stated except maximum clique problem are NP-hard. Note also that there is no known result for the maximum clique problem in DG in general. In this section, we

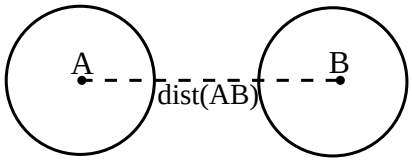


Figure 3.6. Distance between two disks A and B

will give the sketch of the proofs of some NP-hardness results and of the polynomial time algorithm for the maximum clique problem.

Table 3.1. Complexity results for Unit Disk Graphs

Problem	Complexity
Recognition	NP-complete
Independent Set	NP-hard
Dominating Set	NP-hard
Clique	Polynomial
Coloring	NP-hard

3.2.1. Recognition Problem

UDG recognition problem can be stated as given a graph G determining whether G is a unit disk graph or not. Breu and Kirkpatrick (14) showed that this problem is NP-hard by reducing satisfiability problem to it. A graph that simulates SATISFIABILITY is constructed in the following way: Given an instance of SATISFIABILITY problem C, the vertices of the graph correspond to the clauses, variables and negated variables of C, and there is an edge between a literal vertex and a clause vertex if the literal appears in the clause. Then, it is shown that the obtained graph is UDG if and only if there is a true assignment for SAT. The relatively long proof of the reduction can be found in (14).

3.2.2. Maximum Clique Problem

A clique of graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that for any pair of vertices in V', there is an edge between them in E. The clique number $\omega(G)$ is defined as the size of a maximum clique in G.

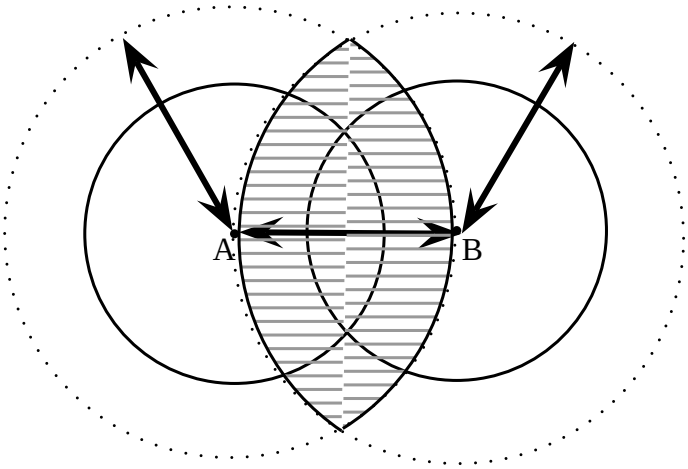


Figure 3.7. Region of intersection of two fictive disks: lens(AB)

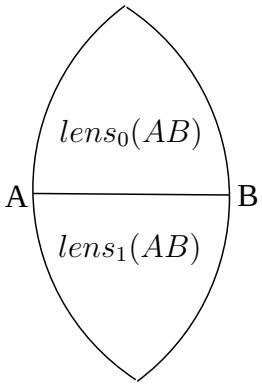


Figure 3.8. Two partitions of $lens(AB)$

Theorem 3.2.1. *(2) There is a polynomial time algorithm for the maximum clique problem in unit disk graphs with representation.*

Proof. Let O_A and O_B be the centers of disks respectively A and B. Let $dist(AB)$ be the distance between O_A and O_B, see Figure 3.6. If $dist(AB) \leq 2r$ then A and B intersects where r is the radius of a disk. Let $lens(AB)$ be the region of intersection of two fictive disks sharing the same centers as disks A and B and of radius $dist(AB)$, see Figure 3.7. Let H_{AB} be the induced subgraph defined by the vertex set having the centers of their corresponding disks in the region $lens(AB)$. We can partition $lens(AB)$ into two parts $lens_0(AB)$ and $lens_1(AB)$ as seen in Figure 3.8. We can observe that every vertex in one of the parts ($lens_0(AB)$ or $lens_1(AB)$) is adjacent to every other vertex in the same part and possibly to some vertices in the other part. Therefore, H_{AB} is the complement of a bipartite graph. A maximum independent set problem in a bipartite graph can be solved in time $O(|V|^{2.5})$.

Observation 3.2.2. *(2) A maximum independent set in a bipartite graph on n vertices can be found in time $O(n^{2.5})$.*

Proof. Given a bipartite graph G, suppose M is a maximum matching of G. Having M, a maximum independent set can be built as follows:

1. Let I_0 be the set of vertices not contained in M. Note that the vertices should form an independent set otherwise we could find a matching larger than M.
2. We complete I_0 in the following way. For each edge in M, we take an endpoint that is not adjacent to any vertex chosen so far. It can be shown that for each matching at least one endpoint must satisfy this property, otherwise we can conclude that there is an augmenting path which will be contradicting with the optimality of M.

In this way, we construct an independent set of size $n - |M|$ (in linear time). This is the largest possible since at most one of the endpoints of each edge in the matching

can be in an independent set. Moreover, a maximum matching in a bipartite graph can be found in time $O(n^{2.5})$ using Edmonds and Karp method (15).

□

It is also true that the maximum clique problem in G is equivalent to the independent set problem in $\overline{G}$. So, we can find a maximum clique in H_{AB} in polynomial time.

Observation 3.2.3. *If A and B are maximally distant points in a set V', then $V' \subseteq lens(AB)$*

Corollary 3.2.4. *If C is a vertex set of a maximum size clique in G, then $C \subseteq H_{AB}$ for some $A, B \in V$ with $dist(AB) \leq 2r$ where r is the radius of a disk.*

Finally, due to Corollary 3.2.4, we can find a maximum clique in the whole graph by taking the clique of maximum size among all maximum cliques searched for all pairs of adjacent vertices (i.e. intersecting disks). This algorithm will be referred to MAXCLIQUE. □

3.2.3. Minimum Vertex Coloring

A coloring of a graph $G = (V, E)$ is an assignment of colors to each vertex in V such that no edge in E connects two identically colored vertices. The chromatic number $\chi(G)$ is defined as the minimum number of colors used for coloring all the vertices of G.

Theorem 3.2.5. *(2) Minimum vertex coloring in UDG is NP-hard.*

Proof. Planar 3-colorability is NP-complete even if the maximum degree is bounded above by 4, (16) . We will reduce this problem to UDG 3-colorability. We transform a planar graph $G = (V, E)$ with maximum degree 4 into a unit disk graph G' such that

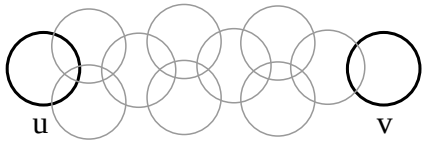


Figure 3.9. Transformation for the reduction from Planar Graphs 3-colorability to UDG 3-colorability

G is 3-colorable if and only if G' is 3-colorable. G' is constructed by making use of the following result.

Lemma 3.2.6. *(17) A planar graph G with maximum degree 4 can be embedded in the plane using $O(|V|)$ area in such a way that its vertices are at integer coordinates and its edges are drawn so that they are made up of line segments of the form $x = i$ or $y = j$, for integers i and j.*

We transform G into G' in the following way: (1) The vertices of G are modeled by disks of radius $\frac{1}{2}$ centered at the location of the vertices in the embedding described in Lemma 3.2.6, (2) the edges of G are replaced by chains of radius $\frac{1}{2}$ disks, see Figure 3.9. Let c$[e_i]$ be the chain of disks in G' that correspond the edge e_i in graph G.

We can observe that any proper 3-coloring of $c[e_i] \cup \{u, v\}$ assigns u and v different colors. On the other hand if we assign different colors to u and v, there exists a proper 3-coloring of $c[e_i] \cup \{u, v\}$. Therefore the reductions is complete. □

3.2.4. Maximum Independent Set Problem

An independent set of a graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that for any pair of vertices in V', there is no edge between them in E. The independence number $\alpha(G)$ is defined as the size of a maximum independent set in G. It can be easily seen that if V' is a vertex cover of G, then $V - V'$ is an independent set. In other words these two problems (minimum vertex cover and maximum independent set) are polynomially equivalent. We will reduce min vertex cover in planar graphs problem which is shown to be NP-complete into vertex cover problem in UDG.

Theorem 3.2.7. *(2) Minimum vertex cover in UDG is NP-hard.*

Proof. We will again use the same construction strategy as in the proof of Theorem 3.2.5. Let G be a planar graph with maximum degree 4 and having a vertex cover S with $|S| \leq k$ and let G' be a unit disk graph having a vertex cover S' with $|S'| \leq k'$. We embed G' in the plane using Lemma 3.2.6. Vertices are modeled as unit disks and each edge uv by a chain having an even number $2k_{uv}$ of disks, which is always possible. We can verify that G has a vertex cover S such that $|S| \leq k$ if and only if G' has a vertex cover S' such that $|S'| \leq k + \sum_{uv \in E(G)} k_{uv}$ □

3.3. Approximation Algorithms

If a problem belongs to NP-hard class, it means that there is no known polynomial time algorithm to solve this problem. In this case, we have three options: (a) we can work in a sub-case where a polynomial time algorithm exists, (b) we can develop a heuristic method, however, there is no guarantee for the optimality or (c) we can find an approximation algorithm. Approximation algorithms cannot guarantee the optimality but at least can give a performance guarantee. We know that the solutions generated from an approximation algorithm lies on an error bound. This is not true for an ordinary heuristic method. Formally, an approximation algorithm for an optimization problem π provides a performance guarantee of ρ if for every instance I of π, the solution value returned by the approximation algorithm is within the factor ρ of the optimal value for I (1). $\rho < 1$ (for a maximization problem) and $\rho > 1$ (for a minimization problem). In our study, since we consider minimum vertex coloring problem ρ is defined as follows $\rho = \frac{\lambda}{OPT}$ where λ is the output of the approximation algorithm. In this section, we will state approximation algorithms for maximum independent set and minimum vertex coloring problems in UDG.

3.3.1. Max Independent Set

We already show that maximum independent set problem in UDG is NP-hard. The following approximation algorithm is suggested in (18).

1. Set IS = $\emptyset$
2. Find a vertex whose neighborhood does not contain an independent set of size larger than 3.
3. Add such a vertex to IS.
4. Remove all its neighbors.
5. Repeat 2,3,4 until G is empty

We start by finding a vertex whose neighborhood does not contain an independent set of size larger than 3. This vertex always exists because of Property 3.1.6 in UDG. We keep this vertex on a list, then we remove all its neighbors. We know from Property 3.1.7 that every induced subgraph of a unit disk graph G is also a unit disk graph. Therefore, we can keep doing these iterations until we get an empty graph.

Lemma 3.3.1. *The performance guarantee of this algorithm is 3.*

Proof. Let us define closed neighborhood of a vertex $v \in V$ as $N(v) \cup \{v\}$ where $N(v)$ is the neighborhood of v. The optimal solution of the maximum independent set problem in G and the output of the algorithm applied to G are respectively referred to $OPT(G)$ and $IS(G)$. By construction, every vertex in $IS(G)$ is in the closed neighborhood of at least one vertex in $OPT(G)$, otherwise, we could add that vertex to $OPT(G)$ which contradicts the optimality. Also by construction, the size of a maximum independent set in the closed neighborhood of every vertex in $IS(G)$ is at most 3. Therefore, closed neighborhood of a vertex $v \in IS(G)$ contains at most 3 vertices from $OPT(G)$. Hence, we have $|OPT(G)| \leq 3|IS(G)|$. □

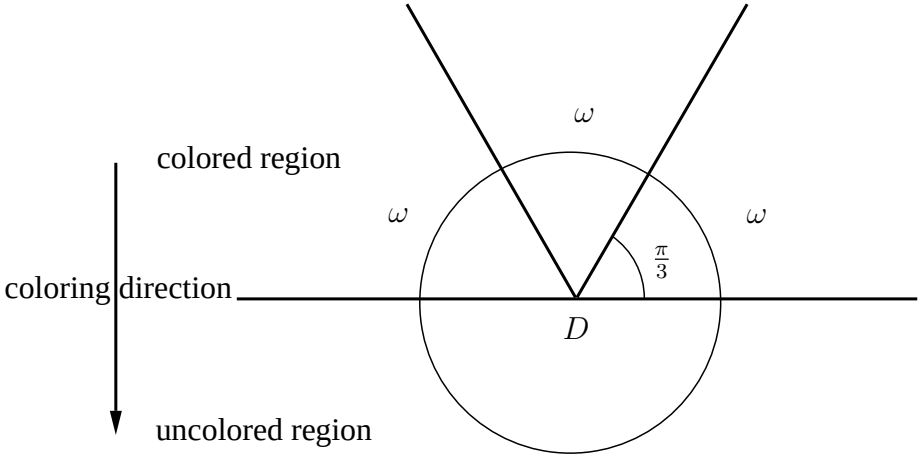


Figure 3.10. 3-approximation coloring algorithm for UDG.

3.3.2. Minimum Vertex Coloring

Minimum vertex coloring is also NP-hard in UDG. We will consider first-fit algorithm with a descending X-coordinate order. We start with the unit disk at the top of the representation, and color it with the first available color and consider remaining graph and repeat this procedure until we color all the vertices. We claim that this algorithm uses at most $3\omega(G) - 2$ colors for a given UDG with representation, see Figure 3.10. First consider a disk D at the time it is assigned its color. Let $N_{up}(D)$ be the set of previously colored disks that intersect D and having a X-coordinate that is not smaller than the X-coordinate of D. We can observe that disks in $N_{up}(D)$ with centers in the same region delimited by an angle of $\frac{\pi}{3}$ must form a clique (together with D). Therefore, we can see that previously colored neighbors of D can be partitioned into most three cliques. In other words $|N_{up}(D)|$ is at most $3(\omega(G) - 1)$. Suppose that all these vertices are colored with different colors. So, the algorithm uses at most $3\omega(G) - 2$ colors. It is always true that $\omega(G) \leq \chi(G)$. Therefore this algorithm has an approximation ratio of 3.

Actually, Malesinska et al. (13) showed that Largest-first order with first-fit coloring strategy is also a 3-approximation algorithm. Moreover this method does not require the graph representation. In addition, Malesinska et al. (19) presented 5-approximation algorithms for DG with and without representation.

In this section of the study, we focused on disk graphs. We have seen that there are many applications that can be modeled using DG and UDG. Therefore, UDG are one of the well-studied graphs in the literature. We started by giving some important properties of UDG. Then we stated some complexity results. We remarked that all the problems we consider, except maximum clique, are NP-hard in UDG. Consequently, approximation algorithms are suggested in the literature. Some properties of DG (UDG) allow us to derive constant approximation ratios, which is not possible for the general graphs. Note that (20) prove that minimum coloring cannot be approximated within a ratio of $n^{\frac{1}{\epsilon}}$ in polynomial time, unless $P = NP$

There are still some open questions in this domain; for example the complexity of maximum clique problem in DG. Besides, since all the algorithms presented are simple in a sense, we can try to improve approximation ratios by developing new complex but still polynomial time algorithms. As we stated before, the use of the additional graph representation information can also be a way to obtain better approximation ratios.

4. COLORING UNIT DISK GRAPHS

We already mentioned that ColorUDG is NP-hard. One way to deal with an NP-hard problem is to look for good bounds, which are preferably generated in polynomial time. Clique number can be considered as a lower bound for the chromatic number and hopefully can be determined in polynomial time for UDG. Clique number does not only provide us a lower bound but it is also required for the exact sequential algorithm and some coloring algorithms that we will present later in this section. Therefore, some modifications will be made on the original maximum clique algorithm in order to improve the running time. On the other hand, heuristic methods can generate feasible solutions constituting an upper bound for this minimization problem. Optimum solutions of the generated test instances will also be calculated in order to assess the performance of the bounds. To this purpose, an integer programming formulation of minimum coloring will be solved using CPLEX.

In a theoretical point of view, we know that the ratio $\frac{\chi(G)}{\omega(G}$ is bounded. In other words the chromatic number of any UDG is bounded by its clique number times a constant, $\chi(G) \leq c \cdot \omega(G)$ and the best known c value is 3, see Section 3. There are some theoretical worst-case results for UDG coloring heuristics. However, in this study we will investigate the empirical performance of bounds for ColorUDG. We will design an experiment in which the quality of the bounds will be assessed in terms of some graph parameters such as number of vertices and density. Besides two construction heuristics, one improvement method will be proposed. According to the empirical results we can say that proposed improvement heuristic demonstrates good performance. Moreover, a sequential exact coloring algorithm for general graphs will be presented. This algorithm will be modified in Section 5 to solve reoptimization problems.

4.1. Exact Coloring

4.1.1. Mixed Integer Programming

In order to asses the real quality of the bounds, we first need to determine the optimal value of ColorUDG. For this purpose, we start by giving the mathematical formulation for ColorUDG problem which is a pure integer (binary) programming.

$$\min \quad \sum_{k \in K} y_k \tag{4.1}$$

$$s.t. \quad E_{u,v}(x_{u,k} + x_{v,k}) \leq 1 \quad \forall (u,v) \in V, k \in K \tag{4.2}$$

$$\sum_{k \in K} x_{u,k} = 1 \qquad \forall u \in V \tag{4.3}$$

$$x_{u,k} \leq y_k \qquad \forall u \in V, k \in K \tag{4.4}$$

$$x_{u,k} \in \{0,1\} \qquad \forall u \in V, k \in K \tag{4.5}$$

$$y_k \in \{0,1\} \qquad \forall k \in K \tag{4.6}$$

If a color k is available then y_k takes value 1 (and 0 otherwise) and variable $x_{u,k}$ takes value 1 if vertex u is colored with color k (and 0 otherwise). Objective 4.1 minimizes the total number of colors used in the graph. $E_{u,v}$ parameter takes value 1 if there is an edge between u and v (and 0 otherwise). Constraints 4.2 are necessary for the feasibility of coloring, in other words, so that adjacent vertices take different colors. Every vertex takes a unique color thanks to the set of constraints 4.3. Constraints 4.4 ensures that used colors are accounted in the objective function.

GAMS software is one of the most popular mathematical programming tools. The model presented above is programmed in GAMS and solved using CPLEX solver. In Appendix A, ColorUDG model code for GAMS can be found. Since the problem is NP-hard, the optimal solution can be found for a limited size instances in a reasonable

time. In our case, our computational resource capacity limits us to work with instances of 100 vertices and density 0.80. For bigger instances, exact solution cannot be obtained in a reasonable time interval (2 hours).

4.1.2. Exact Sequential Algorithm

An exact sequential algorithm for graph coloring is proposed first by Brown (21). For this, we construct the following solution tree : A partial solution of level p is any assignment in which only vertices $x_1, \ldots, x_p$ have been given a color. Let K_{pq} be a coloring of $x_1, \ldots, x_p$ with q colors. Fixing K_{qp}, we obtain all possible solutions by the following procedure.

1. Introduce a new vertex x_{p+1}.
2. Assign colors $1, 2, \ldots, q+1$ to x_{p+1} successively.
3. Eliminate cases which causes infeasibility.
4. Repeat this procedure for all new partial solutions.

Sequential method with first-fit coloring gives the path the most to the left of the solution tree. In order to find an exact solution, the enumeration technique used is called *backtrack programming.*

The basic idea of the algorithm is the following: first, we construct an order of vertices by their non-increasing degree. Then, for each vertex, we determine $U(x_k)$ as the set of available colors which are not used in the actual partial solution of level $k-1$ from $\{1, \ldots, \min(u_k + 1, q-1)\}$ where x_k is the vertex of rank k and u_k is the number of colors used for the actual partial solution of level $k-1$ and q is the upper bound on the number of colors. q is firstly determined with a heuristic method and when a better feasible solution is obtained, we update q. Then, vertices are colored one by one according to the order with the first available color. If we achieve to color all the vertices using q colors then we try with $q-1$ colors. If $U(x_k)$ is empty then we call *label* procedure and backtrack to the maximal ranked vertex among all labeled vertices. Label procedure labels all the unlabeled vertices which possesses all the

following properties: (1) smaller rank than rank of x_k (2) adjacent to x_k, (3) minimal rank among all the vertices of their color. The algorithm stops when we backtrack to the vertex of rank 1.

Later, Brélaz (22) improved this algorithm by observing that a clique can be colored arbitrarily at the beginning without increasing the number of colors. In other words, any coloring of a maximum clique with $\omega(G)$ colors can be extended to an optimum coloring. Therefore, we first detect a maximum clique, put it at the top of the order and color it arbitrarily. During the algorithm if we backtrack to one of the clique vertices the algorithm stops. Finally Peemöller (23) gave the corrected version of Brelaz's algorithm. A modified version of this algorithm will be used to develop an exact algorithm for the reoptimization of unit disk graph coloring in Section 5.

4.2. Lower Bound: Clique Number

Clique number of a graph, denoted as $\omega(G)$, is a natural lower bound for the chromatic number of that graph, $\chi(G)$. It can be easily seen that in order to have a proper coloring, we need at least $\omega(G)$ colors, maybe more. In Section 3, we presented a polynomial-time algorithm for the maximum clique problem for UDG. The time complexity of this algorithm is $O(n^{4.5})$. In this section, we will try to improve this in terms of running time.

MAXCLIQUE does the following: for each pair of vertices (at most $O(n^2)$ times) it runs a slave algorithm, see Section 3, of complexity $O(n^{2.5})$ which finds a maximum clique in a restricted subgraph. Then it is shown that a maximum clique of the whole graph can be obtained by choosing a clique of maximum size among these cliques, (2). We will keep the same slave algorithm but try to avoid calling it for all pairs of vertices so that the performance of the algorithm will be improved in terms of running time. For this purpose, we will list some observations: situations in which calling the slave algorithm is avoidable.

Observation 4.2.1. *For a pair of vertices (A, B), if $dist(AB) \leq \frac{2r}{\sqrt{3}}$ where r is the radius of a disk, then all vertices in $lens(A, B)$ induce a clique, hence it is unnecessary*

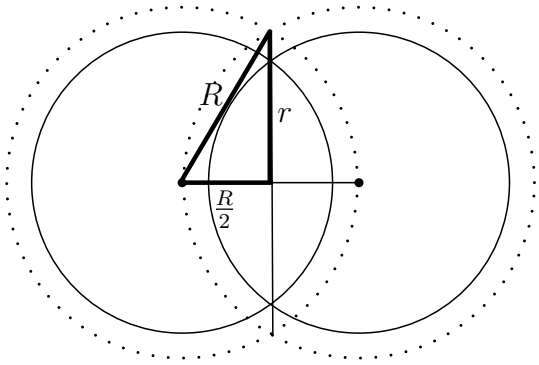


Figure 4.1. Geometrical proof for Observation 4.2.1

to call the slave algorithm.

Proof. We consider disks whose centers are contained in $lens(AB)$. If maximum distance in $lens(AB) \leq 2r$ then it is clear that all disks in that region are touching, therefore we don't need to call the slave algorithm. It can be easily observed that the maximum distance in the region $lens(AB)$ is between north and south poles of that region. So, let $R = dist(AB)$ and we look for a particular R value, $R^\star$ such that the distance between two poles is equal to $2r$, see Figure 4.1. With the use of simple geometry, we can conclude that $R^\star = \frac{2r}{\sqrt{3}}$.

□

Observation 4.2.2. *For a pair of vertices (A, B), if the number of centers of disks contained in $lens(AB)$ is less than the clique number already calculated minus two then it is unnecessary to call the slave algorithm.*

Proof. We know that for a pair of disks (A, B) the size of a clique generated by MAX-CLIQUE could be at most the number of disk centers contained in $lens(AB)$. □

Observation 4.2.3. *Intuitively, maximum clique should be located in one of the dense regions of the graph.*

This intuition can be easily illustrated visually. Figure 4.2 shows the plot of a UDG. If one would determine the maximum clique by just looking at the graph then

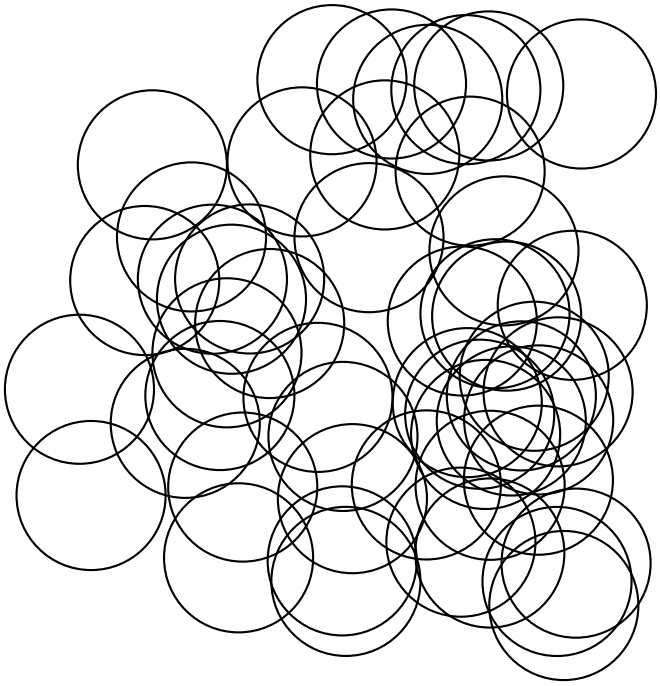


Figure 4.2. Plot of a UDG

it is natural to start with dense regions of it. This idea can be implemented in our maximum clique algorithm by sorting vertices according to their degree. The algorithm requires choosing pairs of vertices, therefore we select first the pairs with high degree. In other words, we consider all pairs of vertices, sum their degrees and sort these sums decreasingly. Improved algorithm chooses pairs of vertices in this order instead of choosing them randomly as in MAXCLIQUE.

An experiment can be conducted in order to measure the effect of these observations on the running time of maximum clique algorithm. Table 4.1 presents average running times results for three algorithms applied to the test set. For each n value and d (density) level, test set contains 25 randomy generated instances (UDG). Section 4.4 describes in detail how the instances are generated. First algorithm is the original MAXCLIQUE algorithm. The second algorithm, called MAXCLIQUE-IMP1, considers Observation 4.2.1 and 4.2.2 in order to reduce running time of MAXCLIQUE. The last algorithm called MAXCLIQUE-IMP2 considers all improvement strategies of Observations 4.2.1, 4.2.2 and 4.2.3. Gain is defined by $\frac{a-b}{a}$ where a is the result of MAXCLIQUE and b denotes the output from one of the improved maximum clique algorithm (MAXCLIQUE-IMP1/MAXCLIQUE-IMP2). In other words, gain represents

time saving percentage of each improved algorithm.

According to this empirical study, MAXCLIQUE-IMP1 and MAXCLIQUE-IMP2 provide in average respectively 20% and 70% time savings. We can observe that the gain of MAXCLIQUE-IMP1 is increasing as density increases. MAXCLIQUE-IMP2 is significantly better than MAXCLIQUE-IMP1 in terms of running time for all types of instances. The only difference between two algorithms is the implementation of the strategy stated in Observation 4.2.3. In other words, sorting vertices by their degree is definitely speeding up finding a maximum clique.

4.3. Upper Bound

Heuristics are tools for generating feasible (not necessarily optimal) solutions in a reasonable time. Therefore, the output of a heuristic algorithm is an upper bound for a minimization problem. Clearly, lower values for an upper bound are preferable. In other words, among the output of different algorithms, one with the minimum output value will be better than other algorithms. In this section, four heuristic methods for ColorUDG will be covered:

1. Scanning Coloring(SC),
2. Clique Coloring(CC),
3. Decreasing Order(DC),
4. Balanced Coloring(BC).

In addition to these construction methods for ColorUDG, inspired from Kempe's Chain, an improvement method is proposed.

4.3.1. Construction Heuristics

The algorithms, which will be described in this section are all sequential coloring algorithms. A sequential coloring algorithm is a generic method to obtain a feasible coloring, which requires an ordering of vertices and a coloring strategy. According to

Table 4.1. Running time results (seconds) for maximum clique algorithms

n	d	MAXCLIQUE	MAXCLIQUE-IMP1		MAXCLIQUE-IMP2	
		Time	Time	Gain	Time	Gain
50	Very low	0,051	0,046	8,71%	0,039	23,91%
	Low	0,284	0,258	9,16%	0,118	58,28%
	Medium	2,082	1,657	20,39%	0,618	70,30%
	High	5,004	3,714	25,77%	1,566	68,71%
75	Very low	0,198	0,184	6,92%	0,107	45,82%
	Low	1,440	1,247	13,40%	0,387	73,13%
	Medium	9,707	7,115	26,70%	2,297	76,33%
	High	24,832	17,529	29,41%	7,600	69,40%
100	Very low	0,494	0,462	6,56%	0,196	60,38%
	Low	4,307	3,544	17,71%	0,887	79,40%
	Medium	32,323	23,380	27,67%	9,023	72,09%
	High	83,622	59,340	29,04%	28,196	66,28%
150	Very low	2,602	2,338	10,13%	0,565	78,28%
	Low	22,255	17,305	22,24%	3,493	84,30%
	Medium	153,298	108,621	29,14%	43,486	71,63%
200	Very low	7,912	6,812	13,90%	1,173	85,18%
	Low	67,081	49,953	25,53%	9,817	85,36%
	Medium	473,609	334,010	29,48%	154,389	67,40%
250	Very low	19,866	16,588	16,50%	2,186	89,00%
	Low	166,321	120,978	27,26%	26,531	84,05%
	Medium	1278,510	919,981	28,04%	462,409	63,83%

Table 4.2. Sequential coloring heuristics for ColorUDG

Method	Order	Coloring Strategy
Scanning Coloring	non-increasing X-coordinate	First Fit
Clique Coloring	non-increasing clique size	First Fit
Decreasing Order	non-increasing degree	First Fit
Balanced Coloring	max clique; non-increasing degree	Least used available color

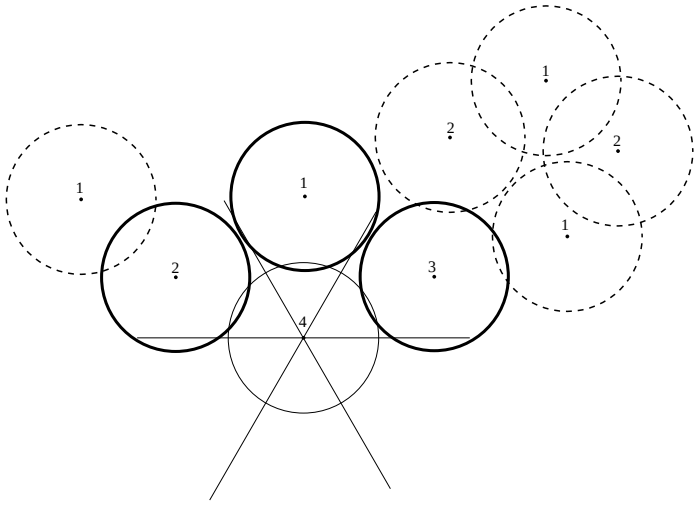
this order, vertices are considered one by one and colored consecutively with one of the available colors imposed by the coloring strategy. Table 4.2 resumes the corresponding input (order, coloring strategy) for different sequential algorithms.

4.3.1.1. Scanning Coloring. Scanning coloring algorithm is the most considered method for ColorUDG in the literature due to its nice geometric property, which enables an approximation analysis (18; 10). Basically, the algorithm sort the disks by their X-coordinate in time $O(n \log n)$, then disks are considered up-to-down one by one and take the first available color, which can be achieved in time $O(n + m)$. In Section 3, we stated that this algorithm has a 3-approximation ratio.

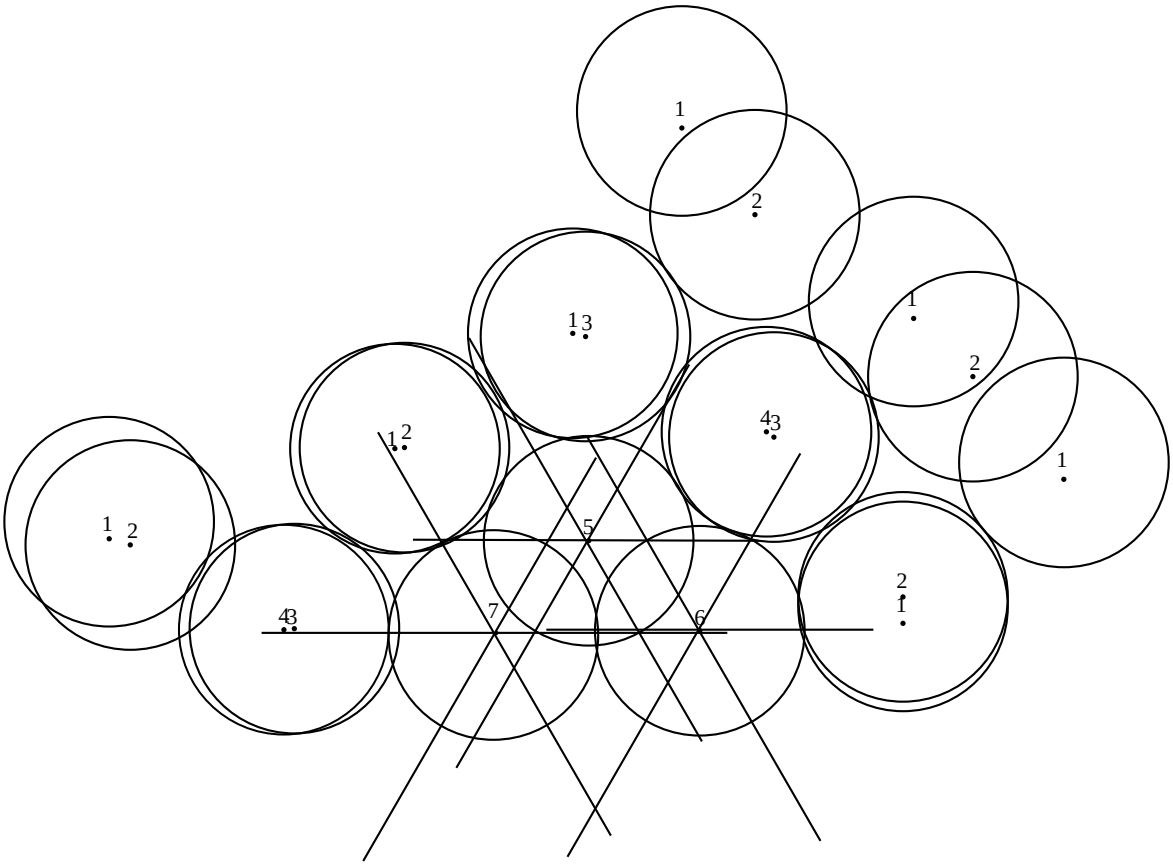
Claim 4.3.1. *SC for ColorUDG has a tight 3-approximation.*

For $\omega = 2$ and $\omega = 3$, worst-case examples could be generated, see Figure 4.3. Malesinska et al. (13) gives a construction of graphs with arbitrarily large ω and such that SC uses exactly $3\omega - 2$ colors.

4.3.1.2. Clique Coloring. During the empirical study, we noticed that $\omega(G)$ is often equal to $\chi(G)$, especially for graphs with lower density value. We can ask the following question. Given a maximum clique of a UDG, which can be determined in polynomial time, and assuming that $\omega(G) = \chi(G)$ holds for this specific instance, is it possible to use this information in order to find an optimal coloring? For this purpose, an algorithm, called Clique Coloring, is suggested and does the following: First, MAXCLIQUE is called and meanwhile all cliques generated throughout the algorithm are memorized. Recall that MAXCLIQUE finds a maximum clique in some induced sub-



(a)



(b)

Figure 4.3. Worst-case examples of Scanning Coloring for (a) $\omega = 2$ and (b) $\omega = 3$

```
Require: G(V, E), MAXCLIQUE
Ensure: A feasible coloring of G
 1: Colors ← {1}
 2: call MAXCLIQUE, memorize all cliques
 3: build an order of cliques: sort all cliques by non-increasing size
 4: detect isolated vertices; color them with 1
 5: while G is not completely colored do
 6:     consider the first non-labeled clique C in the order
 7:     for v ∈ C do
 8:         if v is not colored then
 9:             determine available colors from Colors which are missing in N(v)
10:             if no available color then
11:                 Colors ← Colors ∪ {new color}
12:                 color v with new color
13:             else
14:                 color v with the first available color
15:             end if
16:         end if
17:     end for
18:     label C
19: end while
```

Figure 4.4. Clique Coloring Algorithm

graph H_{AB} for each pair of intersecting disks A and B. This takes time $O(n^{4.5})$. At the end, cliques are considered in sequence according to their decreasing size, until all the vertices are colored in a greedy fashion, (in time $O(n + m)$). Of course, isolated vertices will not appear on any maximum clique, which is memorized. Those vertices can be easily colored with any color available so far. In other words, we try to use $\omega(G)$ colors and add new colors only if it is necessary.

4.3.1.3. Decreasing Order. Decreasing Order (also called Largest First) is one of the standard methods for graph coloring. Vertices are ordered by their decreasing degree in time $O(n \log n)$. Disks are considered according to this order. Then, the vertices are colored in a greedy fashion in time $O(n+m)$. The basic intuition of this algorithm

Require: $G(V, E)$, MAXCLIQUE
Ensure: A feasible coloring of G
determine the maximum clique C_{max} of G using MAXCLIQUE
build an order of vertices: C_{max} then sort remaining vertices by non-increasing degree
color C_{max} using $\omega(G)$ colors
$Colors \leftarrow 1 : w$
for $i = (\omega + 1)$ to n **do**
    Consider the i^{th} vertex v_i
    determine available colors from $Colors$ which are missing in $N(v)$
    if no available color **then**
        $Colors \leftarrow Colors \cup \{\text{new color}\}$
        color v with new color
    else
        determine the size of all color classes for available colors
        color v with the color of the smallest size color class
    end if
end for

Figure 4.5. Balanced Coloring Algorithm

is that, we first color hard-to-color vertices. No geometric information is required for this algorithm which is the essential difference from previous methods presented in this section. Malesinska et al. (13) show that DC is also a 3-approximation coloring algorithm for UDG.

4.3.1.4. Balanced Coloring. This proposed method also tries to color the graph using $\omega(G)$ colors. First, a maximum clique of the graph is determined in time $O(n^{4.5})$. Corresponding vertices are put on the head of the coloring order. Other vertices are sorted by degree value in a non-increasing fashion in time $O(n \log n)$. Based on the observation that in an optimum solution, classes of colors have usually similar sizes, instead of first-fit coloring strategy, we use the least used color among available colors.

4.3.2. Improvement Heuristic

Given a graph, construction methods generate a feasible solution. Then, an improvement method tries to improve this solution while keeping it feasible. A well-known example is 2-OPT method for traveling salesman problem (TSP).

4.3.2.1. Chains. Inspired from Kempe's Chains method for the famous four-color problem (24), we propose the following improvement heuristic method. Starting from a feasible solution, this method tries to decrease the number of colors used. First, we define the induced subgraph $C_j^i(v)$ as the maximal connected component of G containing vertex v and vertices colored i and j such that $N_{C_j^i}(v)$ has only vertices of color j, see Figure 4.6. If the subgraph $(C_j^i(v) \setminus v) \cup (C_i^j(v) \setminus v)$ is disconnected then we switch the colors i and j in one of the components and give the missing color in $N(v)$ (either i or j) to v, otherwise we skip this couple of colors and try another one, until all (i, j) couples are examined. Having a feasible coloring is equivalent to partitioning the graph into stable sets.

Each stable set corresponds to a color class. A color class CC_k which is a set of vertices all colored with the same color k, is selected and we try to color each vertex in this color class with an available color different from k. In this manner, if the color class becomes empty then we can say that the output of the algorithm gives us a better solution since one less color is used to color the graph. We will refer *improvement heuristic* as Chains algorithm applied to all the vertices. Time complexity of the improvement heuristic is $O(n\chi^2(n+m))$. Improvement heuristic applied to the feasible solutions obtained by SC, CC, DC and BC are respectively called iSC, iCC, iDC and iBC.

4.3.2.2. Remarks for the implementation. Before Chains algorithm is called, one can make sure that for the given vertex v, all colors in the given feasible coloring are present in $N(v)$, otherwise we can trivially change the color of v. This consideration ensures that no C_i^j is empty. In the implementation, we should consider the following situation

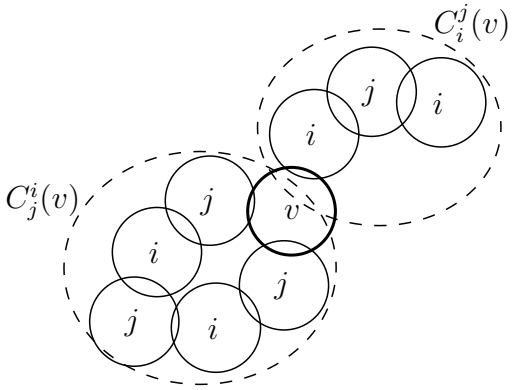


Figure 4.6. $C_i^j(v)$ and $C_j^i(v)$

Require: $G(V, E)$, a feasible coloring $\mathcal{C}$, v
Ensure: a feasible coloring where v has a different color than in $\mathcal{C}$
1: $Colors \leftarrow$ colors of feasible coloring
2: **for** $i \in Colors$ **do**
3: **for** $j \in Colors$ **do**
4: build $C_i^j(v)$ and $C_j^i(v)$
5: **if** $(C_i^j(v) \cup C_j^i(v)) \setminus \{v\}$ is disconnected **then**
6: switch colors in component $C_i^j(v)$
7: color v with i
8: **end if**
9: **end for**
10: **end for**

Figure 4.7. Chains Algorithm

in order to reduce the running complexity of the algorithm. If during the process, we fail to change the color of a vertex, we should skip for the remaining vertices of that color class because there is no longer a hope to empty that color class. In addition, color classes should be updated dynamically, each time that a vertex transfers one color class to the other.

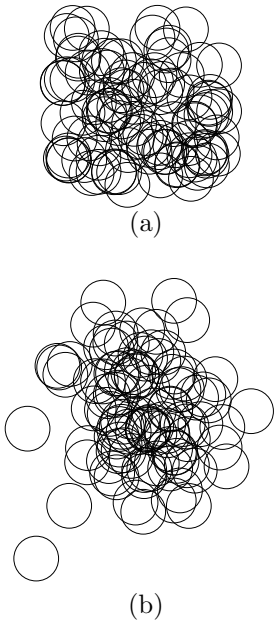
4.4. Design of Experiment

In this part of the study, our aim is to design an experiment where we can assess the quality of both lower (maximum clique size) and upper bounds (output of heuristic methods) for the optimum solution of ColorUDG. First, we should form a set of test

Require: $G(V, E)$, a feasible coloring with l colors
Ensure: a feasible coloring with possibly less than l colors
1: $Colors \leftarrow$ colors of feasible coloring
2: sort non-increasingly Color Classes, CC by size
3: **for** $k = 1$ to l **do**
4: **for** $v \in CC_k$ **do**
5: **if** there exist a missing color in $N(v)$ from $\{1, \ldots, l\} \setminus \{k\}$ **then**
6: color v with the first missing color
7: **else**
8: Chains(v)
9: **end if**
10: **end for**
11: **if** CC_k is empty **then**
12: $Colors \setminus \{k\}$
13: **end if**
14: **end for**

Figure 4.8. Improvement Algorithm

instances; it means that we need a procedure to generate random unit disk graphs. Moreover, we need a control mechanism over the density parameter for the randomly generated UDG. A random graph of n vertices and density value of d is generated as follows (25): for each pair of vertices an edge is put with a probability d. Note that, the graphs that are generated in this way are not necessarily UDG. In addition, the recognition of a UDG is NP-hard. Consequently, we are convinced that we should have another approach to generate random UDG. For this purpose, following procedure is proposed: First, a geometrical area is defined where disks will be put in. This area can be defined as follows: $[0 : x_{max}] \times [0 : y_{max}]$. The X-coordinate and Y-coordinate of a disk is selected according to uniform distribution. There is no incertitude that a graph generated in this way is UDG, it remains to calibrate the density of the graph, since the quality of the bounds will probably be affected by this parameter. For a fixed number of points to be placed, it is clear that if the area gets smaller, the density of the graph increases. Different area values are tried and instances are aggregated according to their density levels: very low(0.06,0.10) low(0.15,0.25), medium(0.39,0.55),



(a)

(b)

Figure 4.9. Two Instances of Type (a) Uniform (b) Normal having same density value

high(0.62,0.78). One can notice that the exact density of our generated graphs are not decided beforehand, but rather calculated after the construction as in (25). For each n value (50,75,100) and d category (very low, low, medium, high) 25 instances are generated.

One can notice that in a real life scenario the distribution of disks on the plane is not homogeneous. Some part of the corresponding graph could be dense/sparse if the corresponding zone is respectively urban/rural. In order to simulate this situation, another set of sample instances are generated using normal distribution instead of uniform distribution. In order to have UDG of different density values we tried different variance values for the normal distribution. Algorithms will be run on both type of instances, and we will try to infer conclusions according to the empirical results. Figure 4.9 (a) and (b) represent instances of type Uniform and Normal respectively.

Another consideration for a real application can be stated as the coverage. It means that a well functioning GSM network should cover the defined area. Figure 4.10 shows the coverage of instances of very low, low, medium and high density respectively. In the figure, all of the instances have the same number of vertices therefore in order to increase the density value; vertices are put in a smaller area. For the sake of visualization, disks are bigger for higher density values. Note that we could not afford

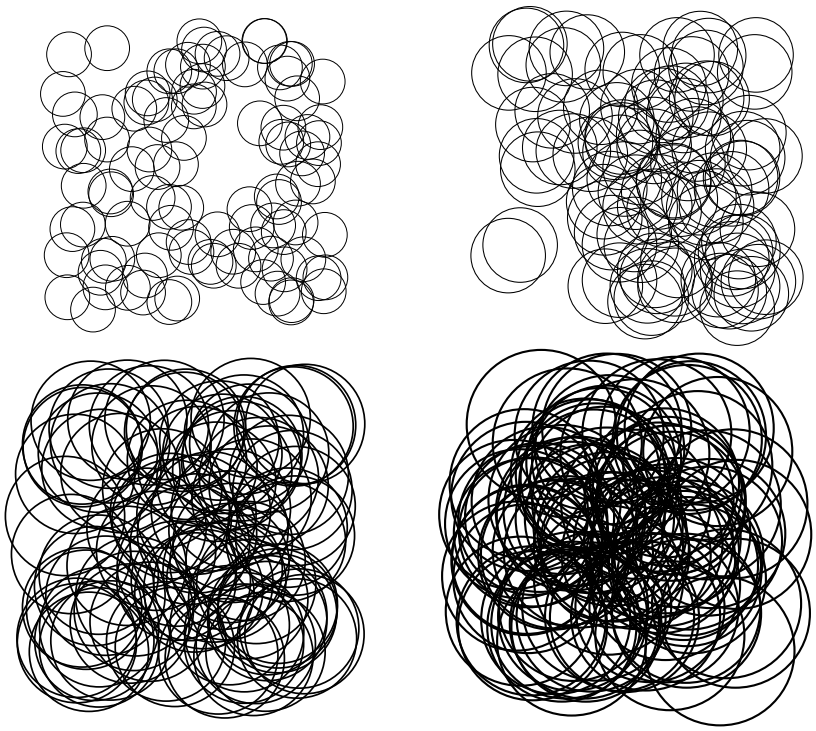


Figure 4.10. Plots of instances: Very Low, Low, Medium and High density

to make some tests for large n =(75 or 100) and high density. However this is not really a drawback knowing that even with the low density, the network has usually a good coverage. Therefore, we can assume that even for low-density value, the coverage requirement is satisfied. Moreover, in practice networks are designed with an objective of good coverage. It means that in real-life applications even a very low-density value for UDG can be sufficient to meet the coverage requirement of the network.

4.5. Results & Discussion

In this section, the quality of the computed lower bounds and upper bounds for ColorUDG is analyzed. All the algorithms mentioned above are run on test sets of type uniform and normal. Detailed empirical results can be found in Appendix B. Table 4.3 and Table 4.4 contain the average of the results on test sets of type uniform and normal respectively. We observed that in some cases bounds were tight, in other words give the exact optimal values. For each algorithm the number of times hitting the optimal value are presented in Table 4.5 and in Table 4.6 for Uniform and Normal type test sets respectively. Four construction heuristic methods give an upper bound

for each instance. Among these three upper bounds, the minimum will be considered the best upper bound for a specific instance. For each heuristic the number of times being best upper bound is counted and presented in Table 4.7 and in 4.8 for test sets of type Uniform and Normal respectively.

According to the results, clique number is a good lower bound especially on sparse graphs. Deviations from χ is below 2% on all type of instances. We can observe that the lower bound is worse in normal type instances than instances of type uniform. DC and BC outperforms other construction methods. Although SC and DC are both 3-approximation algorithms SC has a poor average performance especially on large instances (20% deviation on $n = 75$, $d =$'High'). However, even the worst empirical results are far from the theoretical worst-case result (200%). On ther other hand, DC and BC outperforms other construction heuristics. There is no significant difference between the performances of DC and BC. However, recall that the time complexity of BC is higher.

When we apply improvement heuristic to any construction heuristic, we observe that percentage deviations from χ consistently and significantly decreases. Optimality is almost always reached on sparse graphs. Remark that improvement heuristic applied to a better construction heuristic leads to a better performance as could be expected. Consequently, iDC and iBC outperforms other methods. There is no significant difference between iDC and iBC. We also observe that the performance of iSC consistently worse than the other improved algorithms.

Table 4.3. Average percentage deviations from χ on test set of type uniform

n	d	ω	χ	DC	CC	SC	BC	iDC	iCC	iSC	iBC
50	Very low	-0,72	5,52	0,72	2,17	4,35	1,45	0,00	0,00	0,00	0,72
	Low	-0,45	8,92	4,04	6,28	6,28	3,59	0,45	0,00	1,79	0,90
	Medium	-2,33	17,16	4,90	5,36	11,89	3,03	0,23	1,17	1,40	0,23
	High	-5,16	25,56	1,88	3,76	8,14	1,56	0,16	0,47	0,78	0,47
75	Very low	-0,55	7,28	2,75	2,75	4,40	3,30	0,00	0,00	0,00	0,00
	Low	-3,30	12,12	3,30	5,61	10,23	3,63	0,66	0,33	1,98	0,33
	Medium	-4,22	23,72	5,06	6,58	8,94	3,20	0,51	0,51	1,35	0,34
	High	-1,33	33	10,55	12,36	18,91	9,82	6,30	7,15	8,73	6,42
100	Very low	-1,92	8,32	2,40	6,25	10,10	2,40	0,00	0,00	0,00	0,00
	Low	-2,67	15	5,60	10,13	12,27	5,60	1,33	0,80	1,60	0,53
	Medium	-9,52	30,24	7,41	10,19	12,96	5,82	1,19	1,85	2,78	0,93

Table 4.4. Average percentage deviations from χ on test set of type normal

n	d	ω	χ	DC	CC	SC	BC	iDC	iCC	iSC	iBC
50	Very low	-1,88	6,40	3,12	3,75	8,12	3,75	0,00	0,00	0,00	0,00
	Low	-1,88	10,64	3,01	6,39	11,28	3,01	0,00	0,75	0,75	0,38
	Medium	-4,48	17,84	2,24	2,91	13,00	2,02	0,00	0,45	1,35	0,00
75	Very low	-1,39	8,64	1,85	5,56	8,80	3,24	0,00	0,00	1,39	0,46
	Low	-3,05	14,44	3,60	5,54	8,86	3,60	0,28	0,55	1,39	0,28
	Medium	-6,02	24,60	2,28	3,09	10,57	2,44	0,33	0,49	1,63	0,33
100	Very low	-2,40	10,00	3,60	6,40	8,40	2,80	0,40	0,40	1,20	0,00
	Low	-4,84	18,20	3,52	7,25	11,65	4,40	0,88	1,98	2,64	0,22
	Medium	-8,09	29,16	3,98	4,94	13,44	3,29	0,69	1,23	3,16	0,27

Table 4.5. Percentage of hitting the optimal value on test set of type uniform

n	d	ω	DC	CC	SC	BC	iDC	iCC	iSC	iBC
50	Very Low	96	96	88	80	92	100	100	100	96
	Low	96	64	56	56	68	96	100	84	92
	Medium	72	32	36	4	52	96	84	76	96
	High	28	56	36	4	60	96	88	80	88
75	Very Low	96	80	80	68	76	100	100	100	100
	Low	68	64	56	12	64	92	96	76	96
	Medium	28	4	32	16	44	88	92	80	92
	High	12	8	8	0	8	64	40	20	56
100	Very Low	84	80	52	28	80	100	100	100	100
	Low	68	40	32	16	44	80	88	76	92
	Medium	8	4	12	0	8	64	56	52	60

Table 4.6. Percentage of hitting the optimal value on test set of type normal

n	d	ω	DC	CC	SC	BC	iDC	iCC	iSC	iBC
50	Very low	88	80	76	56	76	100	100	100	100
	Low	80	68	44	24	68	100	92	92	96
	Medium	48	60	60	0	64	100	92	80	100
75	Very low	88	84	64	40	72	100	100	88	96
	Low	64	52	48	16	56	96	92	84	96
	Medium	24	52	48	12	44	92	88	68	92
100	Very low	76	68	48	48	72	96	96	88	100
	Low	40	44	40	4	40	84	72	56	88
	Medium	12	32	28	0	32	80	68	32	92

Table 4.7. Percentage of being best construction heuristic on test set of type uniform

n	d	DC	CC	SC	BC
50	Very low	96	88	80	92
	Low	84	68	68	88
	Medium	60	64	8	92
	High	84	48	8	88
75	Very low	84	84	72	80
	Low	80	60	20	76
	Medium	36	44	24	80
	High	60	36	0	76
100	Very low	92	60	36	92
	Low	80	40	28	76
	Medium	44	20	8	80

Table 4.8. Percentage of being best construction heuristic on test set of type normal

n	d	DC	CC	SC	BC
50	Very low	84	80	60	80
	Low	76	48	24	76
	Medium	68	64	0	72
75	Very low	92	72	44	80
	Low	72	56	28	72
	Medium	60	52	12	60
100	Very low	72	52	52	80
	Low	72	48	12	56
	Medium	56	48	0	68

Table 4.9. Percentage of being best improvement heuristic on test set of type uniform

n	d	iDC	iCC	iSC	iBC
50	Very low	100	100	100	96
	Low	96	100	84	92
	Medium	96	84	76	96
	High	96	88	80	88
75	Very low	100	100	100	100
	Low	92	96	76	96
	Medium	92	92	80	96
	High	80	60	28	76
100	Very low	100	100	100	100
	Low	84	92	80	96
	Medium	76	60	44	80

Table 4.10. Percentage of being best improvement heuristic on test set of type normal

n	d	iDC	iCC	iSC	iBC
50	Very low	100	100	100	100
	Low	100	92	92	96
	Medium	100	92	80	100
75	Very low	100	100	88	96
	Low	100	96	84	100
	Medium	96	92	68	96
100	Very low	96	96	88	100
	Low	88	72	56	100
	Medium	84	72	36	96

5. REOPTIMIZATION OF COLORING UNIT DISK GRAPHS

In operations research, we are interested in solutions of optimization problems. Traditionally, instances of these problems contain only parameters, such as vertices and edges in graph problems. However, in some real-life applications we could have a prior knowledge about similar problem instances and their solutions. For example, assume that we are given a GSM network with a current optimal frequency assignment for its transmitters and imagine that one or more transmitters will be added to the existing network. Instead of solving the problem instance from scratch, we should intuitively use the prior information, in other words the existing assignment. Therefore, our new problem instance contains not only parameters but also an optimal solution of the original instance. The main idea to preserve the old solution is that for small modifications in the original instance, one can make the hypothesis that the solution of the new instance will not change too much. We should check this idea: in which type of problems is this old solution helpful to find the new optimum solution? How much does it help in terms of runtime and quality of the solution? We will start by defining formally reoptimization problems and then we will state current results about this topic. Finally, we will focus on the reoptimization of vertex coloring problem in UDG.

A reoptimization problem can be defined as follows: for a given instance I of a problem U, an optimal solution S of I and a locally modified instance I', find an optimal solution of the instance I'. Local modifications in graph theoretical problems can be stated as follows: inserting or deleting vertices or edges, modifications on edge costs etc. Clearly, if U is polynomially solvable, we can also solve in polynomial time the instance I' by omitting the old optimal solution. On the other hand, for an NP-hard problem, the knowledge of an optimal solution for I could help to solve U on instance I'. For example, in metric Traveling Salesman Problem(TSP), we get a better constant factor 1.4-approximation against the non reoptimization case 1.5-approximation,

Böckenhauer et al. (26) design a 1.5-approximation algorithm for the reoptimization of the Steiner Tree problem which is an improvement over the old best known approximation algorithm which achieves an approximation ratio of 1.55. On the other hand, for metric TSP with deadlines, there is no known improvement, see (26). This kind of approach is not new; it is firstly applied in 1980s to polynomial time solvable problems such as minimum spanning tree (27), shortest path (28). Since the considered problems are polynomial time solvable, the main aim is to reduce the time complexity of the existing algorithms. In the literature, applications appears recently for NP-hard problems such as: TSP (29) (30), Steiner Trees (31), scheduling (32). As far as we know the reoptimization of vertex coloring problem is not considered in the literature. Therefore, we first start by showing complexity results on reoptimization version of vertex coloring problem in unit disk graphs.

5.1. Problem Definition

5.1.1. Adding a New Vertex

When an optimal solution (chromatic number and an optimal coloring) for ColorUDG is available for a unit disk graph G and a new vertex has to be added, related reoptimization problem is referred to ColorUDG+. This problem can be formally described as follows: Given a unit disk graph $G(V, E)$, an optimal coloring of G, a new vertex v^+ with corresponding coordinates/edges, determine an optimal coloring of $G^+(V^+, E^+)$ where $V^+ = V \cup \{v_+\}$ and $E^+ = E \cup E_+$ where E_+ is the set of edges incident to v_+.

5.1.2. Removing a Vertex

When an optimal solution for ColorUDG is available and a vertex has to be removed from the unit disk graph G, related reoptimization problem is referred to ColorUDG-. This problem can be formally described as follows. Given a unit disk graph $G(V, E)$, an optimal coloring of G, a vertex $v \in V$, determine an optimal coloring of $G^-(V^-, E^-)$ where $V^- = V \setminus \{v_-\}$ and E^- is the set of edges induced by V^- in G.

We can also define edge adding/removing reoptimization problems. From practical point of view, these situations can be encountered when the power of the transmitters can be changed. This application suggests that edges can not be added/removed arbitrarily, the addition/removal of edges should satisfy the intersection rules imposed by disk representation. If the power of all transmitters is required to be the same then corresponding graph is still a UDG and the corresponding problem can be stated as follows: Given a UDG with an optimal coloring, determine the chromatic number of the new graph for a modified radius value. On the other hand, if the power of transmitters can be changed independently from each other, then we deal with the problem of reoptimization related to (not unit) disk graphs. Note that edge addition/removal reoptimization problem are not studied in the framework of this thesis; they can be considered as a possible extension.

5.2. Complexity Results

Theorem 5.2.1. *ColorUDG+ is NP-hard.*

Proof. Suppose that there exists a polynomial time exact algorithm for solving ColorUDG+. Let G be a UDG and H_i be an induced subgraph of G with i vertices. Starting from one vertex (with trivial optimum coloring with one color), the algorithm is applied to find an optimal solution of ColorUDG for H_2. Since an optimal solution of ColorUDG for H_2 is available, the algorithm can generate an optimal solution for H_3. Algorithm can be applied repeatedly until we obtain an optimal solution of ColorUDG for H_n which is isomorphic to G. Thus, an optimal solution of ColorUDG for G can be obtained in polynomial time by applying $n-1$ times that algorithm. On the other hand, we know that ColorUDG is NP-hard. Therefore, there cannot exist such an algorithm unless P=NP. □

Theorem 5.2.2. *ColorUDG- is NP-hard.*

Proof. Given a unit disk graph $G(V, E)$ with n vertices as an instance of ColorUDG, we define an instance G' of ColorUDG- as follows: Let $G' = G \cup K_n$ where K_n is an

independent clique of size n. G' is still a UDG since every clique is UDG and independent union of two UDG is a UDG. By independence, $\chi(G') = \max(\chi(G), \chi(K_n))$. It is clear that $\chi(K_n) = n$ and $\chi(G) \leqslant n$. Therefore, $\chi(G') = n$ and an optimal coloring of G' can easily be obtained by giving a different color to each vertex in G and in K_n. Suppose that there is a polynomial time algorithm, say A-, to solve ColorUDG-. We define $G'_i = G \cup K_{n-i}$, in other words the graph obtained from G' by removing i vertices from K_n. Starting from $i = 1$, A-(G'_i, G'_{i-1},A-(G'_{i-1})) is called and we increase i by one until $\chi(G'_i) > n - i$. The optimal solution of ColorUDG for G, $\chi(G) = n - i^\star + 1$ can be obtained in polynomial time where $i^\star$ is the last iterated i value. On the other hand, we know that ColorUDG is NP-hard. Therefore, there cannot exist such an algorithm unless P=NP. □

5.3. Reoptimization Algorithms

We defined two reoptimization problems related to mimimum coloring in UDG: ColorUDG+ and ColorUDG-. In this section we will suggest polynomial-time (not exact) algorithms for these problems which will be tested on generated instances in Section 4. We showed that both problems are NP-hard to answer. This means that there exist no polynomial time algorithm to solve these problems optimally unless $P = NP$. If one could generate all optimal colorings of a graph then it will be an easy task to add/remove a vertex to the original graph and find an optimal coloring of the new graph. Unfortunately, given an optimal coloring, there is no known polynomial time algorithm to generate all optimal colorings. Although we cannot generate all optimal colorings, our main idea for developing algorithms for ColorUDG+ and ColorUDG- will be generating some of them in a smart way. When we add/remove a vertex to/from G, we will try to switch colors on some vertices on the optimal coloring by not harming the feasibility in a such a way that the total number of colors does not increases/decreases.

Table 5.1. Possible situations for solving ColorUDG+ with any algorithm

Output of Algorithm \ $\chi(G^+)$	$\chi(G)$	$\chi(G)+1$
$\chi(G)$	Optimal	Not Possible
$\chi(G)+1$	Non-Optimal	Optimal

5.3.1. Algorithms for ColorUDG+

An instance of ColorUDG+ contains a UDG, say G, an optimal coloring of G and a new vertex v_+. When a new vertex is added to a graph, clearly the chromatic number could remains the same or increases at most by one. Table 5.1 shows all possible situations when we try to solve an instance of ColorUDG+. If the output of the reoptimization algorithm is $\chi(G)$ then it is optimal. Otherwise, the solution could be optimal or not depending on the chromatic number of the new graph, which is unknown.

Naturally, the first idea for an algorithm would be first-fit. This greedy algorithm checks all the neighbors of v_+ and color it with the first available color. If existing colors are not sufficient to color it then we create a new color; so the number of used colors increases by one. Figure 5.1 illustrates a situation where this algorithm fails to find the optimal solution. First-fit for reoptimization (rFF) is a linear time algorithm, only neighbors of v_+ are considered. Another and more time consuming approach could be applying first rFF and then Chains Algorithm to all added vertices (irFF). The running time complexity of this method is definitely smaller than Chains Algorithm applied to all vertices.

5.3.2. Algorithms for ColorUDG-

An instance of ColorUDG- contains a UDG G, an optimal coloring of $G(V, E)$ and a vertex $v_- \in V$. When we remove v_- from G, the chromatic number could remains the same or decreases at most by one. Table 5.2 shows all possible situations when we try to solve an instance of ColorUDG-. If the output of the algorithm is $\chi(G) - 1$

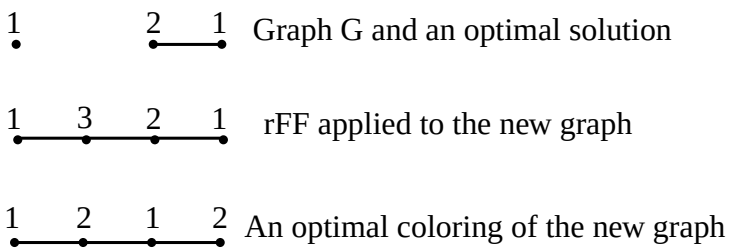


Figure 5.1. A bad example for rFF

Table 5.2. Possible situations for solving ColorUDG- with any algorithm

Output of Algorithm \ $\chi(G^-)$	$\chi(G)-1$	$\chi(G)$
$\chi(G)-1$	Optimal	Not Possible
$\chi(G)$	Non-Optimal	Optimal

then clearly the solution is optimal. Otherwise the solution could be optimal or not depending on the chromatic number of the new graph.

We will suggest two algorithms for ColorUDG-. Suppose that, in the given optimal coloring of G, the color class containing v_- has no other vertices then the chromatic number decreases by one and the optimal coloring of G reduced to G^- gives us an optimal coloring of G^-. Otherwise, the idea is to look at all the other vertices in the same color class as v_-. The procedure CheckColorClass(CCC) tries to change the color of these vertices greedily. For this purpose, every vertex in that class, CCC checks all the neighbors of v_-. Whenever it is possible, it changes the color of that vertex to an already used color. If all the vertices in the color class of v_- can be moved to other color classes, then an optimal coloring of G^- using $\chi(G)-1$ colors is obtained. Otherwise, we get a $\chi(G)$-coloring of G^-. The idea is illustrated in Figure 5.2. Note that this algorithm can fail to find the optimal solution, see Figure 5.3. A more complex algorithm (iCCC) can be developed by applying Chains Algorithm for all vertices in the color class containing v_- in order to empty the color class under consideration.

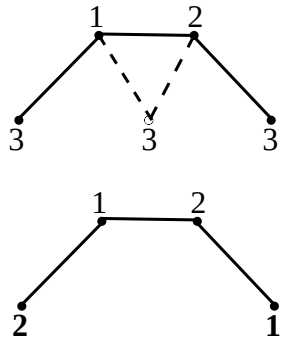


Figure 5.2. An application of CCC algorithm for ColorUDG- where the white vertex is v_{-}.

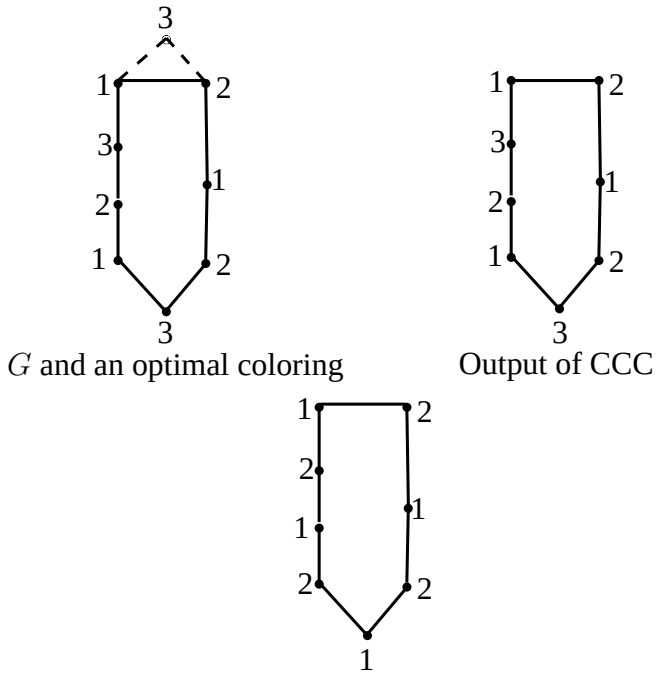


Figure 5.3. A bad example for CheckColorClass algorithm

5.4. Theoretical Analysis of Reoptimization Algorithms

From approximation point of view, any polynomial time reoptimization algorithm for ColorUDG+ and ColorUDG- admits respectively $\frac{\chi+1}{\chi}$ and $\frac{\chi}{\chi-1}$ as approximation ratios where $\chi = \chi(G)$. Due to the NP-hardness of these problems, we cannot expect to have a better constant ratio because this would mean that we can solve them in polynomial time. On the other hand, for practical reasons the following scenarios also need to be considered. In the same setting, instead of one vertex, k vertices are added/removed (given together with the set of edges incident to each one in G^+/G^-), corresponding graphs are referred to G^{+k}/G^{-k} and related problems are referred as ColorUDG+k and ColorUDG-k.

A natural question to ask is whether the approximation results remain the same. In other words, can we say that for any algorithm for ColorUDG+k, the approximation ratio is $\frac{\chi+k}{\chi}$?

First, let us denote ColorG+k as the reoptimization of minimum coloring problem of the following input: a given graph (not necessarily a UDG) with an optimal coloring and k vertices to be added together with corresponding edges.

Proposition 5.4.1. *Any algorithm for ColorG+k where $k \leq \omega(G)$ that does not change the optimum coloring of the original graph has a tight approximation ratio of $\frac{\chi+k}{\chi}$.*

Proof. Since k vertices are added to the graph then chromatic number can increase at most by k. Consequently, any algorithm for ColorG+k is a $\frac{\chi+k}{\chi}$-approximation algorithm. In order to prove the tightness we do the following. Let G be a graph containing $\omega(G)$ independent cliques (i.e. no two vertices from different cliques are adjacent), each of size $\chi(G)$. Let's define u_k^i as the i^{th} vertex in clique k. Color u_k^i with color i for $i = 1, \ldots, \omega(G)$ and $k = 1, \ldots, \omega(G)$. Assume this is the given optimal coloring of G. Let v_j be the vertices to be added with $j = 1, \ldots, k$ to obtain G^{+k}. Connect v_j to u_l^t where $t = l - 1 + j \pmod{\omega(G)}$ for $j = 1, \ldots, k$ and $l = 1, \ldots, \omega(G)$ and by convention $n \pmod{n} = n$. The output of any algorithm which does not change

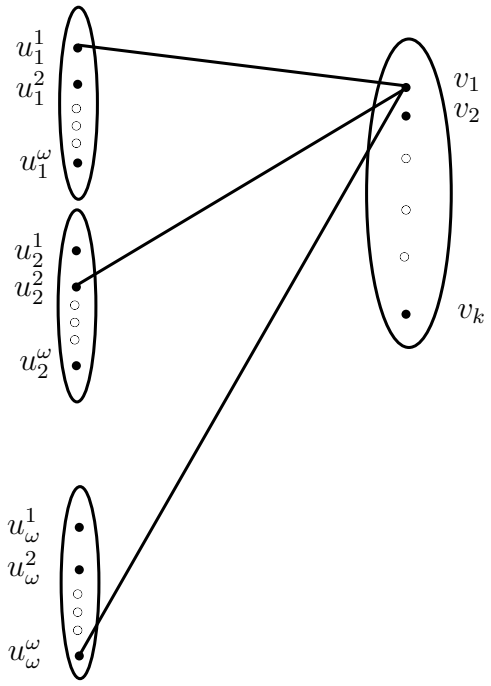


Figure 5.4. Worst-case example construction for ColorUDG+ algorithm

the original optimal coloring of G will be $\omega(G)+k$, since every vertex v_j has neighbors of all $\omega(G)$ colors in G. However $\chi(G^{+k}) = \omega(G)$ which can be obtained by reversing the coloring of the vertices of the last clique. Formally, color u_ω^i with $\omega(G) - i + 1$ for $i = 1, \ldots, \omega(G)$ and then color v_j with the color $\omega(G) + j - 1 \pmod{\omega(G)}$ for $j = 1, \ldots, k$ □

Constructed graph G^{+k}, represents worst case for the first fit algorithm with an input of a general graph. However G^{+k} is not a UDG for $\omega(G) > 5$ since it contains $K_{1,6}$ as a induced subgraph.

Corollary 5.4.2. *Any algorithm for ColorUDG+k where $k \leq \omega(G) \leq 5$ that does not change the optimum coloring of the original graph has a tight approximation ratio of $\frac{\chi+k}{\chi}$.*

Corollary 5.4.3. *rFF is a tight $(\frac{\chi+k}{\chi})$-approximation algorithm for ColorUDG+k where $k \leq \omega(G) \leq 5$.*

Similarly, for ColorUDG-k theoretical bound for the approximation ratio is $\frac{\chi}{\chi-k}$. Let us denote ColorG-k as the reoptimization of minimum coloring problem of the

following input: a given graph (not necessarily a UDG) with an optimal coloring and k vertices to be removed from G together with the corresponding edges.

Proposition 5.4.4. *CCC is a tight ($\frac{\chi}{\chi-k}$)-approximation algorithm for ColorG-k where $k \leq \omega(G^{-k})$.*

Proof. Since we remove k vertices from the graph, the chromatic can decrease at most by k. Consequently, any algorithm for ColorG+k is a $\frac{\chi}{\chi-k}$-approximation algorithm. In order to prove the tightness we do the following. Let G contain a $(\omega+k)$-clique, K_k, another independent clique of size k and a stable set of size $k\omega$, where $\omega = \omega(G^{-k})$. Let's define v_i as the i^{th} vertex of K_k for $i = 1, \ldots, k$. Let S_j^i be the vertices of independent set for $i = 1, \ldots, k$, $j = 1, \ldots, \omega$. Moreover, $N(v_i) \backslash K_k = \{S_j^i | j = 1, \ldots, \omega\}$ for $i = 1, \ldots, k$, see Figure 5.5. $\chi(G) = \omega + k$ since $k \leq \omega$. Color v_i with the color $(\omega + i)$ for $i = 1, \ldots, k$ and S_j^i with color j for $i = 1, \ldots, k$, $j = 1, \ldots, \omega$. Assume this is the given optimal coloring of G.

Let G^{-k} be the instance of ColorUDG-k obtained by removing vertices of colors $\omega + 1, \ldots, \omega + k$ in $K_{\omega+k}$ from G. Note that vertices K_k can be colored using k colors and each vertex in the stable set can take color from $\{1, \ldots \omega\}$ different from the color of the vertex connected in K_k. Therefore, $\chi(G^{-k}) = \omega$ with $k \leq \omega$.

On the other hand, the output of the algorithm with the input G^{-k} is $\omega + k$, since no color change is possible because CCC check vertices of color $\omega + i$ and they have all neighbors colored with $\{1, \ldots, \omega + k\}$.

Described input is a worst case for the CCC with where G and G^{-k} are general graphs. G and G^{-k} are not UDG for $\omega(G^{-k}) > 5$, since they contain $K_{1,6}$ as a induced subgraph. □

Corollary 5.4.5. *CCC is a tight ($\frac{\chi}{\chi-k}$)-approximation algorithm for ColorUDG-k where $k \leq \omega(G) \leq 5$.*

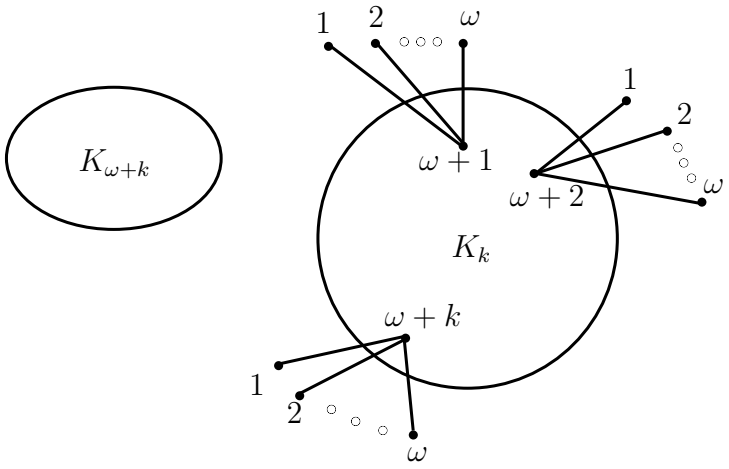


Figure 5.5. Worst-case example construction for CCC

For a next step, we should investigate in order to generalize these theoretical worst-case results of rFF/CCC for ColorUDG+k/ColorUDG-k to be valid for any clique number.

Clearly, optimal coloring of the original graph will help us to rapidly generate feasible solutions compared to starting from nothing. However, another interesting question is whether the optimal solution of original graph only decreases the running times or does it also increase the output quality. We will try to answer these questions by commenting the results of our algorithms on the generated test instances.

5.5. Empirical Results

In the previous part of the study, we have already generated test instances for ColorUDG. These instances are used as original graph for our reoptimization instances. For generating ColorUDG+k instances some new vertices will be added. The coordinates of an added vertex on the existing area of the graph are determined according to uniform distribution. On the other hand, for generating ColorUDG-k instances some of the existing vertices will be removed. Removed vertices are selected randomly according to uniform distribution. As we have already mentioned, the case where only one vertex is added/removed is not very interesting. Therefore, we will add/remove p percentage of the existing vertices to generate respectively ColorUDG+k and ColorUDG-k instances. Our aim is not only to assess the performance of reoptimization algorithms

Table 5.3. Average percentage of deviations from χ on test set ofColorUDG+k

n	Density	p	χ	rFF	irFF	DC
50	Very low	10	5,96	4,02%	1,34%	1,34%
		20	6	11,33%	1,33%	2%
	Low	10	9,64	6,66%	1,25%	4,16%
		20	10,24	9,76%	2,73%	4,29%
	Medium	10	18,44	4,98%	1,95%	4,77%
		20	19,72	7,50%	3,65%	6,69%
	High	10	27,76	3,02%	0,86%	1,72%
		20	30,32	5,01%	1,45%	2,50%
100	Very low	10	8,8	5,90%	1,36%	2,27%
		20	9,44	8,89%	0,42%	3,38%
	Low	10	15,92	8,29%	2,76%	6,78%
		20	17,32	10,16%	3,23%	6,00%

but also to try to answer the question of how the optimum solution of the original graph is helping to find the optimum solution of the new graph. Intuitively, it will help, however empirical tests will check if it is true and how much it helps. For this purpose, a ColorUDG heuristic will be applied to the reoptimization instances. This heuristic solves the problem from the scratch, in other words it does not use the optimum solution of the original graph as additional information. Decreasing Order method is chosen as the ColorUDG heuristic because of its good time complexity and empirical output performance. If a fast ColorUDG heuristic is as good as ColorUDG+k / ColorUDG-k heuristics then we can conclude that there is no need to use the reoptimization algorithms we developed. However, our results will show that this is not the case.

5.5.1. Discussion for ColorUDG+k Algorithms

According to the empirical study, DC is better than rFF. However, irFF is better than DC on all type of instances. We can interpret this results as follows: Although rFF uses the knowledge of optimal coloring of old instance, it does not benefit from

Table 5.4. Percentage of hitting the optimal value on test set forColorUDG+k

n	Density	p	rFF	irFF	DC
50	Very low	10	76%	92%	92%
		20	40%	92%	88%
	Low	10	44%	92%	64%
		20	20%	72%	60%
	Medium	10	36%	64%	32%
		20	12%	44%	16%
	High	10	32%	76%	56%
		20	16%	64%	44%
100	Very low	10	48%	88%	80%
		20	32%	96%	68%
	Low	10	16%	60%	40%
		20	12%	56%	32%

Table 5.5. Percentage of being best upper bound on test set forColorUDG+k

n	Density	p	rFF	irFF	DC
50	Very low	10	80%	96%	96%
		20	40%	92%	88%
	Low	10	48%	100%	72%
		20	36%	96%	84%
	Medium	10	48%	92%	56%
		20	28%	88%	40%
	High	10	40%	88%	64%
		20	24%	92%	64%
100	Very low	10	56%	96%	88%
		20	36%	100%	72%
	Low	10	24%	92%	52%
		20	24%	96%	52%

Table 5.6. Average percentage of deviations from χ on test set ofColorUDG-k

n	Density	p	χ	CCC	iCCC	DC
50	Very low	10	5,12	1,56%	0%	0%
		20	4,88	0,81%	0%	0%
	Low	10	8,24	4,39%	0,48%	2,92%
		20	7,88	2,04%	1,02%	1,53%
	Medium	10	15,84	2,77%	1,01%	3,78%
		20	14,56	3,57%	0,82%	3,84%
	High	10	23	2,43%	0,34%	1,56%
		20	20,96	3,43%	0,95%	2,67%
100	Very low	10	7,88	2,03%	0%	2,03%
		20	7	6,28%	0,57%	6,85%
	Low	10	13,8	4,05%	1,15%	3,76%
		20	12,76	5,64%	0,31%	4,70%

Table 5.7. Percentage of hitting the optimal value on test set forColorUDG-k

n	Density	p	CCC	iCCC	DC
50	Very low	10	92%	100%	100%
		20	96%	100%	100%
	Low	10	68%	96%	76%
		20	84%	92%	88%
	Medium	10	64%	84%	48%
		20	52%	88%	48%
	High	10	52%	92%	64%
		20	48%	84%	48%
100	Very low	10	84%	100%	84%
		20	56%	96%	52%
	Low	10	48%	84%	56%
		20	40%	96%	52%

Table 5.8. Percentage of being best upper bound on test set forColorUDG-k

n	Density	p	CCC	iCCC	DC
50	Very low	10	92%	100%	100%
		20	96%	100%	100%
	Low	10	72%	100%	80%
		20	92%	100%	96%
	Medium	10	72%	100%	60%
		20	60%	96%	56%
	High	10	52%	92%	64%
		20	52%	96%	60%
100	Very low	10	84%	100%	84%
		20	60%	100%	56%
	Low	10	64%	100%	64%
		20	44%	100%	52%

this information because we obtain better results by solving from scratch with DC. However, irFF outperforms other methods, this means that the optimal coloring of old instance is helpful to find a good coloring for the new instance. We can observe that when we add more vertices, performance of all algorithms consistently decrease. This result is not surprising since when more and more vertices are added, we loose the relative importance of having an optimum coloring of the old instance.

5.5.2. Discussion for ColorUDG-k Algorithms

Empirical results show that DC is better than CCC. However, iCCC outperforms both methods with an average deviation from χ which is less than 1%. In addition, optimality is almost always reached. However, additional empirical study is required to check the performance of iCCC. As in results for ColorUDG+k, the performance of iCCC can be interpreted as follows: An optimal coloring of the old instance is helpful to find a coloring of good quality for the new instance.

5.6. Exact Sequential Algorithm For ColorUDG+

In Section 4, we described an exact sequential coloring algorithm for ColorUDG. Coudert (33) stated that exact coloring of graphs is easy if $\chi(G) = \omega(G)$. According to this empirical study, Brelaz's algorithm can solve exactly most of the graph coloring instances by backtracking for a limited number of times. Inspired from this result, we try to develop an exact sequential algorithm for ColorUDG+. Since we have the knowledge of old instance's optimal coloring in hand, instead of fixing only $\omega(G)$ colors (which is done in Brelaz's algorithm), in our case we start by fixing the coloring of the old instance and hope that this algorithm developed for ColorUDG+ does less number of backtracking compared to the Brelaz's algorithm.

With some modifications, Brelaz's exact sequential coloring algorithm can also be used to solve ColorUDG+ instances. Let G be a UDG with a known optimum coloring, we add a new vertex v_+ and call the new graph G^+. As an input for the exact coloring algorithm, an ordering of vertices should be determined. Moreover maximum clique should be placed at the head of the order. In our case, there are two possibilities: either $\omega(G^+) = \omega(G)$ or $\omega(G^+) = \omega(G)+1$ is true. In the first case, since the maximum clique is not changed, we can use the ordering specified in Figure 5.6. On the other hand, if $\omega(G^+) = \omega(G) + 1$ is true then it means that clique number is changed and v_+ is in a maximum clique. Here, we have also two possibilities if $\chi(G) = \omega(G)$ then definitely $\chi(G^+) = \chi(G) + 1$ and we can obtain an optimal coloring by using the old coloring and by giving to v_+ the color $(\chi(G) + 1)$. In all remaining cases, once vertex ordering is determined, all the vertices of G^+ except v_+ is colored using optimal coloring of G and we try to color v_+ with one of the $\chi(G)$ colors. If it is not possible, backtracking is done as defined in Section 4.1.2.

This algorithm is implemented and compared with Brelaz's implementation. We observed that contrary to our expectations the optimal solution of the old instance is not helpful to find the new one faster. Most of the time, Brelaz's algorithm was faster than our implementation.

Figure 5.6. An ordering for sequential exact coloring algorithm

6. CONCLUSIONS

In this study, minimum vertex coloring of UDG and its reoptimization are considered. Since both problems are NP-hard, we mainly focus on the quality of the bounds. Maximum clique is a lower bound for the chromatic number. Firstly, according to some basic observations, an improved maximum clique algorithm is implemented. Empirical results show that the running time performance of the proposed algorithm is significantly better than the original algorithm. Then, some coloring heuristics are considered to derive upper bounds. In the literature, Scanning Coloring with first-fit coloring strategy is the most analyzed heuristic method. One of the reasons is that this algorithm allows us to make a worst-case analysis due to its nice geometric structure. However, empirical results show us that other coloring algorithms such as Largest First outperforms Scanning Coloring. It means that from a practical point of view, Scanning Coloring is not a good choice for UDG coloring. During empirical studies, we observed that many of the generated instances have $\omega(G) = \chi(G)$, especially those having a low density value. Consequently, we investigated the problem of coloring a UDG optimally, given that its chromatic number is equal to its clique number. As $\omega(G)$ can be determined in polynomial time, we proposed two new algorithms based on this idea. Beside these construction methods, we also proposed an improvement method based on Kempe's Chains. Given a feasible solution generated from a construction heuristic, Chains algorithm tries to decrease the number of used colors. According to the empirical results, the output of the algorithm hits the chromatic number in average for 80% of the instances up to 100 vertices.

In the second part of the study, reoptimization of coloring UDG is considered. Four reoptimization problems are defined and two of them are analyzed. We showed that ColorUDG+ and ColorUDG- are NP-hard. Then, we proposed simple heuristic methods. Empirical results show that these heuristics followed by Chains improvement method are promising good performance. Our last challenge was to develop an exact exponential-time coloring algorithm for reoptimization. We presented Brelaz's modified sequential exact coloring method to solve ColorUDG+. However, according to the

empirical results, we observed that the optimal solution of the old instance is not helpful to find the new one faster. Most of the time, starting from beginning was faster than using the old instance's optimal coloring information as an input.

6.1. Open Questions

This study arises several interesting future research directions:

- We know that ColorUDG is NP-hard. However, during empirical study we observed that most of the instances for which $\omega(G) = \chi(G)$ holds are solved exactly. Therefore one may question the complexity of the following problem: Given a UDG G and the information that $\chi(G) = \omega(G)$, can we find an optimum coloring of G in polynomial time? If the case fails to rise (i.e., if this problem remains NP-hard), does the additional information that $\chi(G) = \omega(G)$ helps to derive better approximation algorithms? If this problem is polynomial time solvable then the following question can also be interesting to investigate. Given a UDG G, can we check whether $\chi(G) = \omega(G)$ in polynomial time.
- UDG are not perfect. On the other hand, it is shown that for a given ω value, we cannot construct graphs with arbitrarily large χ. Of course the inequality $\chi(G) \leq c \cdot \omega(G)$ holds for large c values. However finding the lowest c value for all UDG can be an interesting research direction. We know that c is between $\frac{3}{2}$ and 3. In order to have a better lower bound, a construction should be proposed, on the other hand, for an upper bound one should come up with an approximation algorithm with a factor better than 3.
- Maximum clique problem in UDG is solvable in polynomial time, the existing MAXCLIQUE algorithm runs in time $O(n^{4.5})$. This algorithm does the following: checks all pairs of vertices and call a slave algorithm which finds a maximum matching in a bipartite graph in time $O(n^{2.5})$. Based on some simple observations we improved MAXCLIQUE by avoiding checking all pairs. According to our empirical study, we gain in average 70% in terms of running time. It would be interesting to show that this empirical improvement can be also proved theoretically, by making a theoretical complexity analysis which is better than

$O(n^{4.5})$.

- We presented theoretical worst-case results of some simple algorithms for vertex adding/removing reoptimization problem. Results presented are valid for general graphs. However, constructed graphs are UDG if $\omega \leq 5$. Therefore, as a next step we should investigate the generalization of these results for any clique number in UDG.
- In this study, we consider adding/removing vertex case reoptimization for ColorUDG. As we mentioned before, adding/removing edges by increasing/decreasing radius of disks reoptimization problem is also interesting from application point of view. The complexity situation of these reoptimization problems could be the next step to follow.

APPENDIX A: GAMS Code for Solving Vertex Coloring Problem

```
$title Vertex Coloring Problem
set K colors /K1*K25/;
$call GDXXRW.exe diskgraph.xls set=V Rdim=1 par=E rng=A1:K11 Cdim=1 Rdim=1
$GDXIN diskgraph.gdx
set V(*) vertices;
$LOAD V
alias(V,Vp);
Display V;
parameter E(V,V)
$LOAD E;
$GDXIN
Display E;

binary variables         x(V,K) decision variables for vertives
                         y(K)   decision variables for colors;
variable       z      objective;

equations
uniqueColor(V)
feasibility(V,Vp,K)
isColorAvailable(V,K)
totalColorUsed;

uniqueColor(V)..         sum(K,x(V,K)) =e= 1;
feasibility(V,Vp,K)..    E(V,Vp)*(x(V,K)+ x(Vp,K)) =l= 1;
isColorAvailable(V,K)..  x(V,K) =l= y(K);
```

```
totalColorUsed..          z =e= sum(K,y(K));

model coloring Vertex Coloring Problem /all/;
solve coloring minimizinz z using mip;
display x.l,y.l,z.l
```

APPENDIX B: Empirical Results for Bounds of ColorUDG

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	50	7,59	5	5	5	5	5	5	0	0	0	0
2	50	7,35	6	6	6	6	6	6	0	0	0	0
3	50	7,27	6	6	6	6	6	6	0	0	0	0
4	50	8,08	7	7	7	7	7	7	0	0	0	0
5	50	6,20	4	4	4	5	5	5	0	4	4	4
6	50	7,76	5	5	5	6	7	5	0	5	5	5
7	50	7,92	6	6	6	6	6	6	0	0	0	0
8	50	7,43	5	5	5	5	5	5	0	0	0	0
9	50	7,51	5	5	5	5	5	5	0	0	0	0
10	50	6,86	5	5	5	5	5	5	0	0	0	0
11	50	6,37	6	6	6	6	6	6	0	0	0	0
12	50	7,27	6	6	6	6	7	7	0	0	6	0
13	50	7,84	6	6	6	6	6	6	0	0	0	0
14	50	6,61	4	4	4	5	4	4	0	4	0	4
15	50	9,06	6	6	6	6	6	6	0	0	0	0
16	50	7,27	5	5	5	5	5	5	0	0	0	0
17	50	7,35	6	6	6	6	6	6	0	0	0	0
18	50	8,73	5	6	6	6	7	6	0	0	6	0
19	50	8,73	5	5	6	5	5	5	5	0	0	0
20	50	7,51	5	5	5	5	5	5	0	0	0	0
21	50	7,84	5	5	5	5	5	5	0	0	0	0
22	50	8,08	6	6	6	6	6	6	0	0	0	0
23	50	7,27	6	6	6	6	7	6	0	0	6	0
24	50	7,27	6	6	6	6	6	6	0	0	0	0
25	50	8,41	6	6	6	6	6	6	0	0	0	0

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	50	18,37	8	8	8	8	8	8	0	0	0	0
2	50	17,96	9	9	9	9	9	9	0	0	0	0
3	50	19,59	10	10	10	10	10	10	0	0	0	0
4	50	18,94	11	11	11	11	12	11	0	0	11	0
5	50	16,98	7	7	7	8	8	8	0	7	7	7
6	50	21,14	8	8	9	9	8	9	8	8	0	8
7	50	19,59	8	8	9	10	9	9	8	8	8	8
8	50	26,20	11	11	11	13	12	11	0	11	11	11
9	50	19,51	8	8	8	8	8	8	0	0	0	0
10	50	18,94	8	8	9	9	10	9	8	8	9	8
11	50	19,10	9	10	10	10	10	10	0	0	0	0
12	50	18,04	10	10	10	10	10	10	0	0	0	0
13	50	17,96	8	8	9	9	9	9	8	8	8	8
14	50	17,80	8	8	8	9	8	8	0	8	0	8
15	50	19,18	9	9	9	9	9	9	0	0	0	0
16	50	17,71	10	10	10	10	10	10	0	0	0	0
17	50	19,51	8	8	9	9	9	9	9	8	9	9
18	50	19,59	7	7	8	8	9	7	7	7	8	7
19	50	20,82	10	10	10	10	10	10	0	0	0	0
20	50	17,71	9	9	10	9	9	9	9	0	0	0
21	50	23,18	9	9	10	11	11	10	9	9	9	9
22	50	19,76	12	12	12	12	12	12	0	0	0	0
23	50	17,71	8	8	9	8	9	9	8	0	9	0
24	50	19,84	9	9	9	9	10	9	0	0	9	0
25	50	15,59	8	8	8	9	8	8	0	8	0	8

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	50	52,82	19	19	21	20	20	20	19	19	19	19
2	50	50,12	21	21	21	21	23	21	0	0	21	0
3	50	45,63	15	16	17	17	19	17	16	16	16	16
4	50	39,18	15	15	16	15	17	15	16	0	15	0
5	50	51,76	18	19	20	21	21	19	19	19	20	0
6	50	44,33	14	14	15	15	16	15	14	15	14	15
7	50	52,49	17	17	17	20	20	18	0	19	17	17
8	50	49,88	17	17	18	18	19	18	17	17	18	17
9	50	45,88	16	16	18	17	19	17	16	16	17	16
10	50	42,45	16	17	18	18	19	18	17	17	17	17
11	50	43,51	17	17	17	17	18	17	0	0	17	0
12	50	43,51	16	16	16	17	16	16	0	16	0	0
13	50	49,80	16	16	16	17	18	17	0	16	16	16
14	50	53,47	18	19	19	20	20	19	0	19	19	0
15	50	45,80	17	17	18	17	18	17	17	0	17	0
16	50	47,59	15	15	16	16	18	16	15	16	15	15
17	50	51,92	18	18	19	18	20	18	18	0	18	0
18	50	46,78	15	16	17	16	19	16	16	0	16	0
19	50	47,67	17	17	18	19	19	18	17	17	17	17
20	50	54,69	17	20	21	20	22	20	20	0	20	0
21	50	44,08	16	16	18	18	19	17	16	16	17	16
22	50	51,76	18	18	18	18	20	18	0	0	18	0
23	50	49,63	16	16	18	19	19	18	16	17	17	16
24	50	46,53	17	17	18	18	19	17	17	17	17	0
25	50	50,78	18	20	20	20	22	20	0	0	21	0

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	50	62,12	22	22	22	23	23	22	0	22	22	0
2	50	73,14	26	27	28	29	28	28	27	27	27	27
3	50	69,80	24	24	25	24	25	24	24	0	24	0
4	50	71,10	25	28	28	29	29	28	0	28	28	0
5	50	74,86	24	26	27	27	29	27	27	26	27	27
6	50	67,67	24	24	25	24	27	25	24	0	24	24
7	50	71,92	27	27	29	27	29	27	27	0	27	0
8	50	69,47	25	25	25	27	29	25	0	25	26	0
9	50	68,33	24	24	24	25	27	24	0	24	25	0
10	50	65,71	24	25	25	25	26	25	0	0	25	0
11	50	66,86	24	25	25	25	26	25	0	0	25	0
12	50	78,12	27	28	28	29	31	29	0	28	28	28
13	50	67,67	21	23	24	24	27	24	23	23	23	23
14	50	66,12	23	24	24	26	25	25	0	24	24	24
15	50	65,80	24	25	25	25	25	25	0	0	0	0
16	50	78,94	28	30	31	31	32	31	30	30	30	30
17	50	71,59	25	27	27	27	29	27	0	0	27	0
18	50	74,94	24	29	29	30	30	29	0	29	29	0
19	50	68,82	22	25	26	26	26	25	25	25	25	0
20	50	75,76	28	29	29	29	32	29	0	0	29	0
21	50	71,84	23	24	25	26	26	25	24	25	24	25
22	50	67,02	22	24	24	24	26	24	0	0	24	0
23	50	73,80	26	26	26	27	28	26	0	26	26	0
24	50	70,94	23	24	25	27	29	25	24	25	25	24
25	50	71,67	21	24	25	27	27	25	24	25	25	25

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	75	8,04	7	7	7	7	7	7	0	0	0	0
2	75	7,78	7	7	8	7	7	8	7	0	0	7
3	75	8,43	7	7	7	7	7	7	0	0	0	0
4	75	9,51	8	8	8	8	8	8	0	0	0	0
5	75	7,32	8	8	8	8	8	8	0	0	0	0
6	75	7,86	6	6	6	6	6	6	0	0	0	0
7	75	7,35	5	5	6	6	6	6	5	5	5	5
8	75	8,29	7	7	7	8	7	8	0	7	0	7
9	75	7,39	7	7	7	7	7	7	0	0	0	0
10	75	7,82	7	7	7	7	8	8	0	0	7	7
11	75	7,93	8	8	8	8	8	8	0	0	0	0
12	75	7,78	7	7	7	7	8	7	0	0	7	0
13	75	7,96	8	8	8	8	9	8	0	0	8	0
14	75	6,59	7	7	8	7	8	8	7	0	7	7
15	75	7,24	7	7	7	7	7	7	0	0	0	0
16	75	7,57	8	8	9	8	8	8	8	0	0	0
17	75	7,78	7	7	7	7	7	7	0	0	0	0
18	75	7,57	7	7	8	7	8	8	7	0	7	7
19	75	7,64	8	8	8	9	8	8	0	8	0	0
20	75	8,14	8	8	8	8	9	8	0	0	8	0
21	75	7,39	7	7	7	8	7	7	0	7	0	0
22	75	7,10	6	7	7	8	7	7	0	7	0	0
23	75	9,73	11	11	11	11	12	11	0	0	11	0
24	75	8,04	7	7	7	7	7	7	0	0	0	0
25	75	7,75	6	6	6	6	6	6	0	0	0	0

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	75	19,57	12	12	12	12	12	13	0	0	0	12
2	75	19,17	12	12	13	13	14	13	12	12	13	12
3	75	19,64	11	12	12	14	14	12	0	12	12	0
4	75	19,82	14	14	14	14	16	14	0	0	15	0
5	75	18,45	11	11	12	12	12	11	11	11	12	0
6	75	20,94	11	11	13	14	13	13	12	11	12	12
7	75	17,91	12	13	13	13	14	13	0	0	13	0
8	75	20,07	12	12	13	12	14	12	12	0	12	0
9	75	19,14	12	13	13	13	14	14	0	0	13	13
10	75	21,80	13	15	15	15	15	15	0	0	0	0
11	75	19,68	11	11	12	11	12	12	11	0	12	11
12	75	18,99	12	13	13	13	15	13	0	0	13	0
13	75	18,27	11	11	11	12	12	11	0	11	11	0
14	75	18,13	13	13	13	13	15	13	0	0	13	0
15	75	18,02	10	10	10	12	10	10	0	10	0	0
16	75	18,67	10	11	11	12	12	11	0	11	11	0
17	75	21,80	12	12	12	13	13	12	0	12	12	0
18	75	18,95	12	12	13	12	13	13	12	0	12	12
19	75	20,04	12	14	14	14	16	14	0	0	14	0
20	75	21,26	12	13	13	13	14	13	0	0	13	0
21	75	21,33	12	12	13	12	13	13	13	0	12	12
22	75	18,95	12	12	12	12	13	12	0	0	12	0
23	75	18,59	11	11	11	12	12	11	0	11	11	0
24	75	20,90	11	11	12	13	12	12	11	11	12	11
25	75	22,05	12	12	13	14	14	14	12	13	12	12

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	75	45,44	22	22	23	26	23	23	22	22	22	22
2	75	46,63	24	25	26	26	25	26	25	25	0	25
3	75	45,73	25	25	26	25	27	25	25	0	25	0
4	75	45,12	25	25	26	25	25	25	25	0	0	0
5	75	51,57	28	29	30	29	31	29	29	0	29	0
6	75	45,12	21	22	23	25	25	23	22	22	22	22
7	75	46,77	23	25	27	25	29	27	25	0	26	25
8	75	43,60	20	22	23	24	23	22	23	22	22	0
9	75	43,28	20	20	22	23	24	23	21	20	22	20
10	75	45,73	23	25	26	25	26	25	25	0	25	0
11	75	45,30	25	26	27	26	28	26	26	0	26	0
12	75	47,32	21	22	24	24	25	24	22	22	22	23
13	75	44,25	19	20	21	25	24	21	21	22	22	21
14	75	47,35	25	25	26	25	26	25	25	0	25	0
15	75	41,77	21	21	22	23	21	22	21	21	0	21
16	75	44,54	20	22	23	24	23	22	22	22	22	0
17	75	45,51	21	21	22	24	25	22	21	22	21	21
18	75	42,67	21	22	22	23	24	22	0	22	22	0
19	75	46,20	22	25	26	26	29	26	25	25	26	25
20	75	44,50	22	23	24	24	23	24	23	23	0	23
21	75	50,09	23	24	25	26	27	24	24	24	24	0
22	75	51,03	26	27	28	27	29	27	27	0	27	0
23	75	56,14	27	28	30	29	31	29	28	28	28	28
24	75	47,06	22	23	25	25	27	24	23	23	25	23
25	75	47,82	22	24	26	28	26	26	24	24	24	24

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	75	67,75	35	37	38	38	40	38	37	37	37	37
2	75	68,04	31	32	35	35	36	36	34	33	34	33
3	75	69,62	32	35	37	38	39	36	35	35	36	36
4	75	65,44	31	31	35	34	34	33	31	32	33	32
5	75	65,26	30	32	33	35	36	34	33	33	33	33
6	75	74,56	32	37	39	39	42	39	38	38	40	37
7	75	73,33	35	39	39	39	43	39	0	0	39	0
8	75	72,79	36	38	39	40	41	39	38	39	39	38
9	75	64,47	31	32	34	33	37	34	32	32	34	33
10	75	66,81	28	30	32	33	37	33	30	32	32	31
11	75	72,04	40	40	41	40	43	40	40	0	41	0
12	75	72,90	32	35	38	39	42	36	35	36	36	35
13	75	62,52	27	28	31	31	34	31	29	29	30	29
14	75	71,53	37	37	37	38	42	38	0	37	37	37
15	75	69,95	30	33	36	36	38	34	34	34	34	34
16	75	66,09	30	33	35	34	37	34	33	33	34	33
17	75	70,02	32	34	37	37	38	36	35	34	35	35
18	75	72,00	35	36	37	39	42	37	36	37	37	36
19	75	66,16	29	31	33	35	36	32	32	32	34	31
20	75	71,53	34	35	36	38	41	36	35	35	36	35
21	75	70,88	32	35	36	38	39	36	35	36	36	35
22	75	70,13	32	36	38	37	40	37	36	36	36	36
23	75	71,89	30	34	35	37	40	37	34	35	35	34
24	75	76,29	39	0	43	44	44	43	43	43	44	43
25	75	72,11	34	35	38	40	40	38	36	37	35	36

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	100	7,45	7	7	8	8	8	8	7	7	7	7
2	100	7,74	7	7	8	9	8	8	7	7	7	7
3	100	8,04	8	8	9	9	10	9	8	8	8	8
4	100	8,00	9	9	9	9	9	9	0	0	0	0
5	100	7,62	8	8	8	8	9	8	0	0	8	0
6	100	6,67	8	8	9	8	9	8	8	0	8	0
7	100	6,97	8	8	8	8	8	8	0	0	0	0
8	100	7,52	7	7	7	8	8	7	0	7	7	0
9	100	7,45	8	9	9	10	10	9	0	9	9	0
10	100	7,72	9	9	9	10	9	9	0	9	0	0
11	100	7,54	7	8	8	9	9	9	0	8	8	8
12	100	7,21	8	9	9	10	10	9	0	9	9	0
13	100	9,17	12	12	12	12	14	13	0	0	12	12
14	100	8,06	8	8	8	8	10	8	0	0	8	0
15	100	6,89	8	8	8	8	9	8	0	0	8	0
16	100	7,17	8	8	8	8	9	8	0	0	8	0
17	100	7,19	7	7	7	8	8	7	0	7	7	0
18	100	7,64	7	8	8	8	9	8	0	0	8	0
19	100	7,94	9	9	9	9	9	9	0	0	0	0
20	100	7,09	8	8	8	8	9	8	0	0	8	0
21	100	7,90	11	11	11	11	11	11	0	0	0	0
22	100	7,60	8	8	9	9	9	8	8	8	8	0
23	100	6,46	7	7	7	8	7	7	0	7	0	0
24	100	6,93	8	8	8	8	9	8	0	0	8	0
25	100	7,68	9	9	9	10	9	9	0	9	0	0

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	100	18,87	13	13	14	16	14	15	13	13	13	13
2	100	21,86	13	14	15	18	18	16	15	15	15	15
3	100	19,66	15	15	17	17	17	16	16	16	15	15
4	100	19,86	13	14	15	16	16	15	14	14	14	14
5	100	20,42	15	15	16	16	18	15	15	15	16	0
6	100	20,16	16	16	16	16	18	16	0	0	16	0
7	100	18,83	15	16	16	16	16	16	0	0	0	0
8	100	23,01	17	17	19	19	20	19	17	17	17	17
9	100	19,23	14	16	17	16	16	17	16	0	0	16
10	100	19,47	14	15	16	16	19	16	15	15	15	15
11	100	19,07	13	13	14	15	15	14	13	13	14	13
12	100	20,32	14	14	16	18	16	16	15	14	15	14
13	100	18,61	17	17	17	17	18	17	0	0	17	0
14	100	19,98	18	18	18	19	21	18	0	18	19	0
15	100	18,79	14	16	16	16	17	16	0	0	16	0
16	100	19,35	14	15	15	16	17	15	0	15	15	0
17	100	19,15	16	16	16	16	16	17	0	0	0	16
18	100	19,45	14	14	16	16	17	14	14	14	14	0
19	100	18,71	13	13	14	15	15	14	13	13	13	13
20	100	17,82	16	16	16	16	17	16	0	0	16	0
21	100	20,20	14	14	15	17	16	16	15	14	14	14
22	100	20,44	14	15	15	16	17	15	0	15	16	0
23	100	18,79	14	14	15	16	16	15	14	14	14	14
24	100	19,78	16	16	16	16	16	16	0	0	0	0
25	100	19,86	13	13	16	18	15	16	14	14	13	14

instance #	n	density	ω	χ	DC	CC	FC	BC	iDC	iCC	iFC	iBC
1	100	43,25	24	26	28	29	28	28	26	27	26	26
2	100	42,69	27	27	30	30	33	29	29	27	28	27
3	100	51,58	28	35	35	35	42	35	0	0	37	0
4	100	49,27	28	33	34	33	34	33	33	0	33	0
5	100	49,60	28	32	33	35	36	35	32	32	32	32
6	100	46,06	25	30	34	33	34	32	30	30	31	30
7	100	45,64	27	28	31	29	32	31	28	28	28	28
8	100	50,22	28	32	35	36	36	33	33	33	33	32
9	100	42,53	25	26	29	30	29	28	27	26	26	26
10	100	45,31	31	32	33	32	34	33	32	0	32	32
11	100	48,53	27	29	30	38	32	30	29	31	29	30
12	100	47,15	25	28	32	32	33	30	30	29	31	29
13	100	47,60	27	28	32	33	33	30	28	29	30	29
14	100	49,84	25	30	33	35	34	33	30	31	30	29
15	100	52,93	28	32	34	35	37	34	33	34	34	34
16	100	52,75	31	33	34	35	36	34	33	33	33	33
17	100	51,45	25	30	32	33	38	32	31	31	33	31
18	100	48,24	30	31	34	35	34	33	31	32	31	31
19	100	45,90	28	28	30	31	30	30	28	28	29	29
20	100	48,46	30	31	32	33	35	33	32	31	32	31
21	100	51,27	30	34	36	37	39	36	34	34	34	35
22	100	51,88	31	34	37	36	38	37	34	35	36	34
23	100	48,28	27	30	32	33	32	31	29	30	30	29
24	100	47,92	24	29	31	33	32	30	29	29	29	29
25	100	47,62	25	28	31	32	33	30	29	30	30	29

REFERENCES

1. Garey, M. R. and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

2. Clark, B. N., C. J. Colbourn, and D. S. Johnson, "Unit disk graphs", *Discrete Mathematics*, Vol. 86, No. 1-3, pp. 165–177, December 1990.

3. Berge, C., *Graphs and hypergraphs*, North-Holland Pub. Co.; American Elsevier Pub. Co., Amsterdam, New York,, [rev. ed.] translated by edward minieka. edition, 1973.

4. Metzger, B. H., "Spectrum Management Technique", 1970, presentation at 38th National ORSA meeting (Detroit, MI).

5. Aardal, K. I., C. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano, "Models and Solution Techniques for the Frequency Assignment Problem", ZIB-report 01–40, Konrad-Zuse-Zentrum fur Informationstechnik Berlin, Berlin, Germany, 2001, `http://www.zib.de/Publications/abstracts/ZR-01-40/`.

6. Eisenblätter, A., M. Grötschel, and A. M. C. A. Koster, "Frequency Assignment and Ramifications of Coloring", *Discussiones Mathematicae Graph Theory*, Vol. 22, pp. 51–88, 2002, `http://www.zib.de/Publications/abstracts/ZR-00-47/`.

7. Hale, W. K., "Frequency Assignment: Theory and Applications", *Proceedings of the IEEE*, Vol. 68, pp. 1497–1514, 1980.

8. Vizing, V. G., "Critical graphs with a given chromatic class", *Diskret*, 1965.

9. Tesman, B. A., *T-colorings, list T-colorings, and set T-colorings of graphs*, Ph.D. thesis, Department of Mathematics, Rutgers University, 1989.

10. Marathe, M. V., H. Breu, H. B. H. Iii, S. S. Ravi, and D. J. Rosenkrantz, "Simple

Heuristics for Unit Disk Graphs", *Networks*, Vol. 25, pp. 59–68, 1995.

11. Chudnovsky, M., N. Robertson, P. Seymour, and R. Thomas, "The strong perfect graph theorem", *Annals of Mathematics*, Vol. 164, pp. 51–229, 2006.

12. Mycielski, J., "Sur le coloriage des graphs", *Coloqium Mathematicum*, Vol. 3, pp. 161–162, 1955.

13. Malesinska, E., S. Piskorz, and G. Weißenfels, "On the chromatic number of disk graphs", *Networks*, Vol. 32, No. 1, pp. 13–22, 1998.

14. Breu, H. and D. G. Kirkpatrick, "Unit Disk Graph Recognition is NP-hard", *Computational Geometry. Theory and Applications*, Vol. 9, pp. 9–1, 1998.

15. Edmonds, J. and R. M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", *J. ACM*, Vol. 19, No. 2, pp. 248–264, 1972.

16. Garey, M. R., D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems", *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pp. 47–63, ACM, New York, NY, USA, 1974.

17. Valiant, L. G., "Universality Considerations in VLSI Circuits", *IEEE Trans. Computers*, Vol. 30, No. 2, pp. 135–140, 1981.

18. Fishkin, A. V., *Approximation and Online Algorithms*, chapter Disk Graphs: A Short Survey, Springer Berlin / Heidelberg, 2004.

19. Malesinska, E., *Graph-Theoretical Models for Frequency Assignment Problems*, Ph.D. thesis, Technische Universitat Berlin, 1997.

20. Feige, U. and J. Kilian, "Zero Knowledge and the Chromatic Number", *IEEE Conference on Computational Complexity*, pp. 278–287, 1996, `citeseer.ist.psu.edu/feige96zero.html`.

21. Brown, J., "Chromatic Scheduling and the Chromatic Number Problem", *Management Science*, Vol. 19, No. 4, pp. 456–463, 1972.

22. Brélaz, D., "New methods to color the vertices of a graph", *Commun. ACM*, Vol. 22, No. 4, pp. 251–256, 1979.

23. Peemöller, J., "A correction to Brelaz's modification of Brown's coloring algorithm", *Commun. ACM*, Vol. 26, No. 8, pp. 595–597, 1983.

24. Wikipedia, "Five Colour Theorem", `http://en.wikipedia.org/wiki/Five%5Fcolour%5Ftheore`

25. Erdös, P. and A. Renyi, "On the Evolution of Random Graphs", *Publ. Math. Inst. Hungar. Acad. Sci.*, Vol. 5, pp. 17–61, 1960.

26. Böckenhauer, H.-J., J. Hromkovic, T. Mömke, and P. Widmayer, "On the Hardness of Reoptimization", *SOFSEM*, pp. 50–65, 2008.

27. Frederickson, G. N., "Data structures for on-line updating of minimum spanning trees", *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pp. 252–257, ACM, New York, NY, USA, 1983.

28. Rohnert, H., "A dynamization of the all pairs least cost path problem", *Proceedings on STACS 85 2nd annual symposium on theoretical aspects of computer science*, pp. 279–286, Springer-Verlag New York, Inc., New York, NY, USA, 1985.

29. Archetti, C., L. Bertazzi, and M. G. Speranza, "Reoptimizing the traveling salesman problem", *Networks*, Vol. 42, No. 3, pp. 154–159, 2003.

30. Böckenhauer, H.-J. and D. Komm, "Reoptimization of the Metric Deadline TSP", *MFCS '08: Proceedings of the 33rd international symposium on Mathematical Foundations of Computer Science*, pp. 156–167, Springer-Verlag, Berlin, Heidelberg, 2008.

31. Bilò, D., H.-J. Böckenhauer, J. Hromkovič, R. Královič, T. Mömke, P. Widmayer, and A. Zych, "Reoptimization of Steiner Trees", *SWAT '08: Proceedings of the 11th Scandinavian workshop on Algorithm Theory*, pp. 258–269, Springer-Verlag, Berlin, Heidelberg,

2008.

32. Bartusch, M., R. H. Mohring, and F. J. Radermacher, "Scheduling project networks with resource constraints and time windows", *Ann. Oper. Res.*, Vol. 16, No. 1-4, pp. 201–240, 1988.

33. Coudert, O., "Exact coloring of real-life graphs is easy", *DAC '97: Proceedings of the 34th annual conference on Design automation*, pp. 121–126, ACM, New York, NY, USA, 1997.