

# **Maximum Clique Conformance Measure for Graph Coloring Algorithms.**

**A Thesis Submitted in Partial Fulfillment of the requirements  
of the Master Degree in Computer Science  
Middle East University**

By

**Abdel Mutaleb Mohammad Al-zou'bi**

Supervisors

**Prof. Mohammad M. Al-Haj Hassan**

**Dr. Mohammad E. Malkawi**

**Amman, Jordan**

**July 2011**

## جامعة الشرق الأوسط

### أقرار تفويض

أنا، عبد المطلب محمد الزعبي، أفوض جامعة الشرق الأوسط بتزويد نسخ من رسالتي للمكتبات أو الهيئات أو الأشخاص عند طلبهم حسب التعليمات النافذة في الجامعة.

التوقيع: 

التاريخ: 21-9-2011

**Middle East University**

**Authorization statement**

I, Abdel Mutaleb Mohammad Al-Zou'bi Authorize the Middle East University to supply copies of my Thesis to libraries, establishments or individuals upon their request, according to the university regulations

**Signature:** 

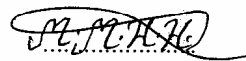
**Date:** 21-9-2011

### Committee Decision

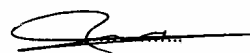
This thesis (Maximum Clique Conformance Measure for Graph Coloring Algorithms) was successfully defended and approved on July 23th 2011.

#### **Examination Committee Signature**

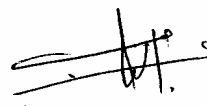
Prof. Mohammad M. Al-Haj Hassan.  
Professor  
(Middle East University)



Dr. Belal Abuhajja  
Associate Professor  
(Middle East University)



Dr Abdel Rauf M. Rjoub  
Associate Professor  
(Jordan University of Science and Technology)



## DECLARATION

I do hereby declare that the present research work has been carried out by me, under the supervision of Prof. Mohammad M. Al-Haj Hassan and Dr. Mohammad E. Malkawi. And this work has not been submitted elsewhere for any other degree, fellowship or any other similar title.

Abdel Mutaleb M. Al-zou'bi

Department of Computer Science  
Faculty of Information Technology  
Middle East University

## ACKNOWLEDGMENTS

The research that has been presented in this thesis is a result of my continual interaction with my advisors, colleagues and every one interested in this area of research.

I feel very proud to have worked with my advisors, Prof. Mohammad Al-Haj Hassan and Dr Mohammad Malkawi. To each of them I owe a great debt of thanks for their patience, motivation and friendship. My advisors have taught me a lot in the field of computer science. They also inspired into myself the ability of discovery and investigation which is the heart of research.

Many thanks to my family for their incredible support. I feel very lucky to be one of you.

*Abdel Mutaleb*

## CONTENTS

<b>LIST OF FIGURES.....</b>	<b>X</b>
<b>LIST OF TABLES.....</b>	<b>XII</b>
<b>LIST OF ABBREVIATION.....</b>	<b>XIII</b>
<b>ABSTRACT.....</b>	<b>XIV</b>
<b>ABSTRACT IN ARABIC.....</b>	<b>XVI</b>
<b>CHAPTER ONE: INTRODUCTION.....</b>	<b>1</b>
1.1 PRELIMINARIES .....	1
1.1.1 Graphs .....	1
1.1.2 Graph Coloring .....	5
1.1.3 Cliques .....	6
1.1.4 P, NP, NP-Complete Problems .....	7
1.2 THE PROBLEM.....	8
1.2.1 The largest Degree Algorithm .....	9
1.2.2 Clique detection problem.....	13
1.2.3 The modified largest degree algorithm.....	15
1.3 OBJECTIVES OF THE STUDY .....	15
1.4 SCOPE AND SIGNIFICANCE OF THE STUDY .....	16
<b>CHAPTER TWO: LITERATURE SURVEY.....</b>	<b>17</b>
2.1 GRAPH COLORING USING THE EXHAUSTIVE SEARCH (BACKTRACKING).....	17
2.2 LOCAL SEARCH METHOD .....	20
2.3 GREEDY COLORING.....	21
<b>CHAPTER THREE: THE MODIFIED GRAPH COLORING ALGORITHM.....</b>	<b>25</b>
3.1 SATURATION DEGREE .....	25
3.2 GRAPH COLORING USING THE MODEFIED ALGORITHM.....	28
3.3 CLIQUE DETECTION USING THE MODEFIED ALGORITHM.....	32
3.3.1 Graph partitioning:.....	32
3.3.2 Clique Finding .....	34

3.3.3 Clique Finding Evaluation .....	38
<b>CHAPTER FOUR:IMPLEMENTATION .....</b>	<b>40</b>
4.1 GRAPH GENERATION.....	40
4.2 NODE CALSS:.....	46
4.3 GRAPH CLASS .....	48
4.3 FIXEDVALUES CLASS .....	49
4.4 GRAPHREADER CLASS.....	50
4.5 LARGESTDGREE CLASS .....	51
4.6 MODEFIEDLARGESTDEGREE CLASS .....	54
<b>CHAPTER FIVE: COMPLEXITY AND PERFORMANCE ANALYSIS .....</b>	<b>56</b>
5.1 COMPLEXITY ANALYSIS .....	57
5.1.1 Complexity of the LDC algorithm.....	57
5.1.2 Complexity of LDSC algorithm .....	58
5.2 PERFORMANCE ANALYSIS .....	58
<b>CHAPTER SIX: CONCLUSIONS AND FUTURE WORKS .....</b>	<b>71</b>
6.1 SUMMERY .....	71
6.2 CONCLUSIONS .....	72
6.3 FUTURE WORK .....	73
<b>REFERENCES.....</b>	<b>73</b>



---

## LIST OF FIGUERS

Figure 1.1: Indirectgraph, Node, Edge,and Adjacent Nodes.....	2
Figure 1.2: Direct graphs and Indirect graphs.....	3
Figure 1.3: An indirect weighted graph G represents exam schedule.....	4
Figure 1.4: Adjacency matrix represents indirect graph.....	5
Figure 1.5: Examples of invalid and valid graph coloring.....	5
Figure 1.6: Map Coloring .....	6
Figure 1.7: Examples of cliques.....	7
Figure 1.8: An indirect unweighted graph G represents exam schedule ...	9
Figure 1.9: Graph coloring example using the largest degree algorithm.....	12
Figure1.10: Finding 4-clique in a graph of 7 vertices using the brute force ...	14
Figure 2.1: Solving the maze using the backtracking method.....	17
Figure 2.2: The exhaustive search algorithm for solving the graph coloring...	19
Figure 2.3: Using the min-conflicts algorithm to improve the coloring.....	21
Figure 2.4: Predefined orders of visiting graph nodes.....	22
Figure 2.5: Predefined orders of colors.....	23
Figure 3.1: Saturation Degree.....	25
Figure 3.2: Dsatur Algorithm.....	27
Figure 3.3: Graph Coloring using the modified algorithm.....	31
Figure 3.4: Graph Division.....	33
Figure 3.5: Recoding the coloring track using the ColorsOrderLis.....	37
Figure 4.1: Indirect graph and its corresponding adjacency matrix.....	41
Figure 4.2: Heavy density graph, Regular density and Low density graph.....	42

<b>Figure 4.3:</b>	<b>GraphGenerater class .....</b>	<b>42</b>
<b>Figure 4.4:</b>	<b>CreateGraph calling statements.....</b>	<b>43</b>
<b>Figure 4.5:</b>	<b>CreateGraph body.....</b>	<b>44</b>
<b>Figure 4.6:</b>	<b>WriteInBenchMark Method calls.....</b>	<b>45</b>
<b>Figure 4.7:</b>	<b>WriteInBenchMark body.....</b>	<b>45</b>
<b>Figure 4.8:</b>	<b>Screenshot for a benchmark.....</b>	<b>46</b>
<b>Figure 4.9:</b>	<b>GraphReeader.java.....</b>	<b>51</b>
<b>Figure 4.10:</b>	<b>Compare method.....</b>	<b>52</b>
<b>Figure 4.11:</b>	<b>ColorNode method.....</b>	<b>52</b>
<b>Figure 4.12:</b>	<b>CCI_Dev_Conv Method.....</b>	<b>54</b>
<b>Figure 4.13:</b>	<b>Compare method.....</b>	<b>54</b>
<b>Figure 5.1:</b>	<b>LDC and LDSC Coloring – Low Density Graphs .....</b>	<b>64</b>
<b>Figure 5.2:</b>	<b>LDC and LDSC Coloring – Regular Density Graphs.....</b>	<b>64</b>
<b>Figure 5.3:</b>	<b>LDC and LDSC Coloring – Heavy Density Graphs.....</b>	<b>65</b>
<b>Figure 5.4:</b>	<b>Clique Conformance Index – Low Density Graphs.....</b>	<b>66</b>
<b>Figure 5.5:</b>	<b>Clique Conformance Index – Regular Density Graphs.....</b>	<b>66</b>
<b>Figure 5.6:</b>	<b>Clique Conformance Index –Heavy Density Graphs.....</b>	<b>67</b>
<b>Figure 5.7:</b>	<b>1000 Nodes Low, Regular, and Heavy Density Graphs.....</b>	<b>67</b>
<b>Figure 5.8:</b>	<b>Coloring Time for Backtracking Algorithm.....</b>	<b>69</b>

## LIST OF TABLES

<b>Table 4.1: Node.java Attributes.....</b>	<b>46</b>
<b>Table 4.2: Node.java Methods.....</b>	<b>47</b>
<b>Table 4.3: Graph.java Attributes.....</b>	<b>48</b>
<b>Table 4.4: Graph.java Methods.....</b>	<b>49</b>
<b>Table 4.5: FixedValues.java Attributes.....</b>	<b>49</b>
<b>Table 5.1: First Experiment .....</b>	<b>59</b>
<b>Table 5.2: Second Experiment.....</b>	<b>60</b>
<b>Table 5.3: Third Experiment .....</b>	<b>61</b>
<b>Table 5.4: Average results for Three Experiments .....</b>	<b>63</b>
<b>Table 5.5: Comparing LDC and LDSC with Backtracking Algorithm.....</b>	<b>68</b>

## LIST OF ABBREVIATION

<b>CCI</b>	<b>Clique Conformance Index</b>
<b>GCP</b>	<b>Graph Coloring Problem</b>
<b>SD</b>	<b>Saturation Degree</b>
<b>LDC</b>	<b>Largest Degree Coloring</b>
<b>LDSC</b>	<b>Largest Degree Saturation Degree</b>

## ABSTRACT

### Maximum Clique Conformance Measure for Graph Coloring Algorithms.

By

Abdel Mutaleb M. Al-zou'bi

Supervisors

Prof. Mohammad Al-Haj Hassan

Dr. Mohammad Malkawi

The graph coloring problem (GCP) is an essential problem in graph theory [22], it has many applications such as the exam scheduling problem [38], register allocation problem[2], timetabling [15], and frequency assignment[19]. The maximum clique problem is also another important problem in graph theory and it arises in many real life applications like managing the social networks[7]. These two problems have received a lot of attention by the scientists, not only for their importance in the real life applications, but also for their theoretical sides [31,32,32,33,34,35,36].

Unfortunately, solving these problems is very complex, and the proposed algorithms for this purpose are able to solve only the small graphs with up to 80 nodes. On the other hand and in order to module the real life applications we need graphs of hundreds or thousands of nodes.

This thesis presents a new algorithm for solving these hard problems, the new algorithm will run in a considerable time and it shows a competitive results for both of

special purpose graphs as well as the general random graphs. It also introduces a new measure for the deviation of the algorithms from maximum clique based coloring.

**Keywords:** Clique conformance Index, Convergence Rate, Deviation Rate, Graph, Graph Coloring Problem, Maximum Clique Problem, NP-Complete Problems.

## ملخص

مقياس توافق خوارزميات تولين المخططات مع المخطط الجزئي الأعظم المكتمل

الترابط

إعداد

عبد المطلب محمد الزعبي

إشراف

الأستاذ الدكتور محمد الحاج حسن

الدكتور محمد ملكاوي

تعتبر مشكلة تولين المخططات من المشاكل الرئيسية في نظرية المخططات □ حيث يوجد لها الكثير من التطبيقات العملية مثل مشكلة جدولة الامتحانات □ ومشكلة إدارة الذاكرة □ ومشكلة الجداول الزمنية □ ومشكلة تخصيص الترددات. كما ان مشكلة تحديد المجموعة المكتملة الترابط تعتبر ايضا من المشكلات المهمة في نظرية المخططات وهي تدخل في الكثير من التطبيقات العملية مثل إدارة الشبكات الإجتماعية. حظيت هاتان المشكلتان بالكثير من الاهتمام من قبل العلماء وذلك ليس نظرا الى اهميتهما العملية فحسب وإنما لأهميتهما في الجانب النظري ايضا.

إلا أن ايجاد حل لهاتين المشكلتين يعتبر معقدا جدا □ وأن الخوارزميات المقدمة لهذا الغرض هي فقط لحل المشاكل التي تتمثل في المخططات صغيرة فقط والتي تتكون من 80 رأس كحد اعلى. بينما في الجهة الاخرى □ من اجل نمذجة اي تطبيق عملي فإننا نحتاج الى مخططات تتكون من مئات او ألوف النقاط.

تقدم هذه الرسالة خوارزمية جديدة لحل هاتين المشكلتين الصعبتين □ تعمل في وقت معقول ولديها القدرة على معالجة المخططات ذات الحجم الكبير واطهرت نتائج جيدة لجميع انواع المخططات.

# CHAPTER ONE

## INTRODUCTION

---

This work represents an analysis and improvement for what had been proposed by Dr. Mohamad Malkawi and Dr. Mohammad Al-HajHasan in their paper [37]. In that paper they proposed a new algorithm for solving the graph coloring problem and they used the coloring algorithm to provide a solution for the exam scheduling problem. In this thesis we introduce a new improvement for that algorithm. We also provide an analysis for the original and the improved algorithm in terms of how closely the algorithms follow the maximum clique base coloring of a graph by introducing a new measure for the deviation of the algorithms from maximum clique based coloring.

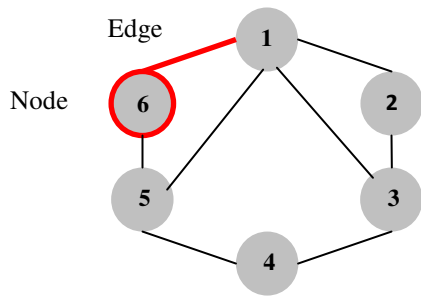
### 1.1 PRELIMINARIES

This section will be devoted to review some important concepts related to the subject of the thesis, together with some examples.

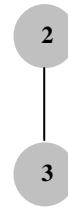
#### 1.1.1 GRAPHS

An indirect graph  $G$  is an ordered pair  $(N, E)$  where  $N$  is a set of Nodes and  $E$  is a set of non-direct edges between nodes. Two nodes are said to be adjacent if there is an edge between them. See Figure 1.1 for illustration.





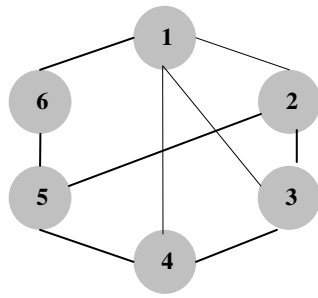
An indirect graph G, composed from 6 nodes and 8 edges.



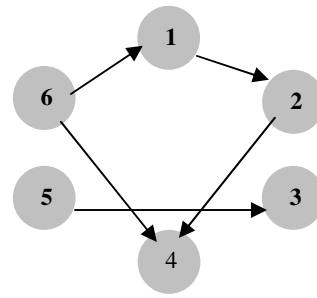
Nodes 2 and 3 are adjacent.

**Figure 1.1 Indirectgraph, Node, Edge,and Adjacent Nodes**

Depending on the type of edges between nodes, graphs may be classified as directed graphs or indirect graphs, in the direct graphs each edge represents a relationship between one node with another and not vice versa; in other words there is a relation between two nodes in one way. While in the indirect graphs each edge represents two relationships between both of the connected nodes. See Figure 1.2 which represents two graphs the first is indirect graph while the second is a direct graph. In the rest of this thesis we will not address the direct graphs; our concentration will be only on the indirect graphs



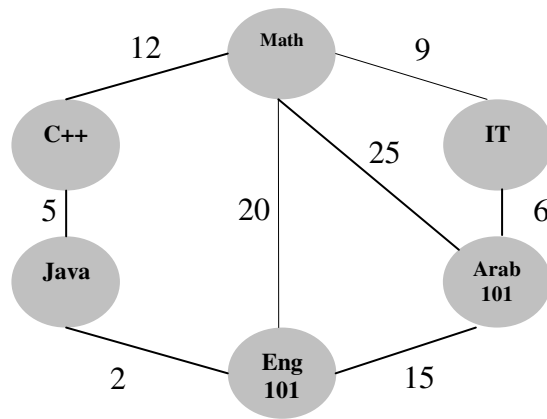
An indirect graph G. Every edge represents a relation in two ways. For example, the edge between node 1 and node 2 means that there are two relations, the first connects node 1 with node 2 while the second connects node 2 with node 1.



A direct graph G. Every edge represents only a single relation. For example, the edge between node 1 and node 2 means that there is only one relation which connects node 1 with node 2.

**Figure 1.2 Direct graphs and Indirect graphs**

Another classification for graphs, which also depends on the nature of edge, is based on the concept "edge's weight". According to this classification, graphs can be weighted graphs or unweighted graphs. Edge is considered weighted when it is associated with any valuable symbol such as numbers or characters. See Figure 1.3, which represents an example for a weighted graph for an exam schedule in a particular university; nodes in this graph stand for a class exam name, and weights on edges stand for the number of students who are common in both classes.



**Figure 1.3 an indirect weighted graph G represents exam schedule in a particular university.**

The importance of studying graphs comes from its ability to model many problems. The philosophy of graphs states that there are many nodes and there are relations between them. Many applications with distributed elements and relations between these elements can be modeled as a graph.

Graphs can be represented in computer systems in many ways such as adjacency matrix; in this format, an  $[n, n]$  matrix is used to represent the graph, where  $n$  is the number of nodes. Matrix element  $[x, y]$  is one if and only if there is a relationship between node  $x$  and node  $y$ . and it is 0 otherwise See Figure 1.4 for illustration.

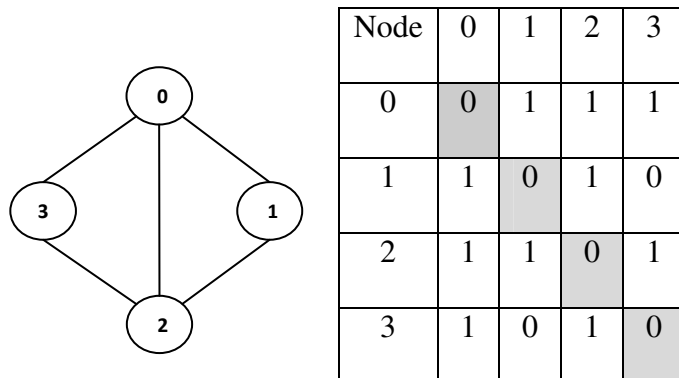


Figure 1.4 Adjacency matrix represents indirect graph

### 1.1.2 GRAPH COLORING

General graph coloring is the process of labeling the graph's elements with special labels (usually called colors) under special conditions. Node coloring is the process of coloring the nodes such that no two adjacent nodes have the same color. See Figure 1.5.

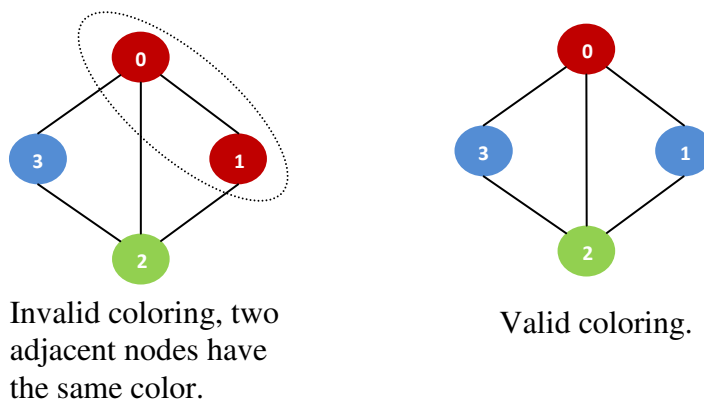
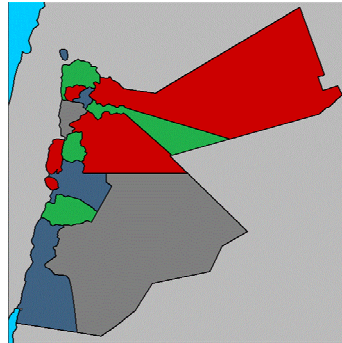


Figure 1.5 Examples of invalid and valid graph coloring

An interesting coloring problem is the four colors theorem [9]. The theorem resulted from the scientist Francis Guthrie attempts in coloring the countries of England with four colors. He noted that four colors are enough to color the map so that no countries having the same border share the same color; see Figure 1.6. This suggestion was still

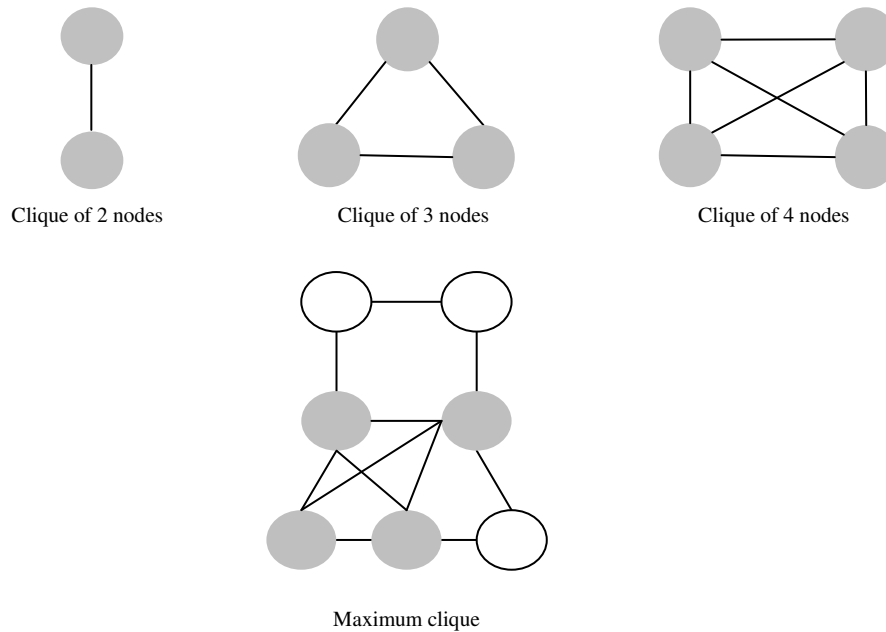
under study till 1879; in this year Alfred Kempe [25] published a paper that claimed he proved the theorem. The graph coloring problem continued to be under research by mathematical scientists until it finally was proved numerically in 1976 by the computer scientists Kenneth Appel and Wolfgang Haken [3,4].



**Figure 1.6 Map Coloring**

### **1.1.3 CLIQUES**

Clique in an indirect graph  $G$  is a subset of nodes  $N$  such that for every two nodes in  $N$ , there exists an edge connecting both nodes, while the maximum clique is that one which has the maximum number of nodes. See Figure 1.7. Luce & Perry 1949 were the first who used the term "Clique" in graph theoretic, in their work they have used cliques in modeling social networks (groups of people who all know each other) [28].



**Figure 1.7 Examples of cliques**

### 1.1.4 P, NP, NP-COMPLETE PROBLEMS

This section will introduce three types of complexity classes, P, NP and NP-Complete problems. This brief introduction aims to give the reader good idea about these topics in order to make him/her able to understand any of these terms in the context of this thesis.

In studying these complex classes of graphs we focus on two important things, first is the required time (how many steps does it take to solve the problem), second is the required space (how much memory does it take to solve the problem) with more emphasis on the time factor since the huge progress in manufacturing storage devices reduces the importance of memory space.

The Problems with (P) complexity are those which can be solved using deterministic sequential solution in polynomial time. In this definition there are three concepts: deterministic, sequential and polynomial. Deterministic is a concept in which there is only one possible action that the system (a state machine) might take in response to any

input. For example, let's take the quick sort algorithm. The algorithm is composed of ordered steps of statements that will be repeated for any input to produce the required output. Sequential means that the input for next step is the output of the current step. Polynomial time refers to the number of steps required to solve the problem and it is represented by the following notation  $O(n^k)$  where  $k \geq 0$ .

The NP (Nondeterministic Polynomial) problems are these which can be verified very quickly but they can't be solved quickly. For example let  $S = \{3,4,2,1,-10,15,8\}$ . The problem is to find if a subset of the set  $S$  has add up to zero, this can be checked (verified) quickly because the result of adding the first five elements is zero. However finding such elements seems to be difficult; hence this problem is NP problem.

Despite of the class complexity  $P$  seems to be an independent class but it is contained into the class NP which contains many other problems such as the sales man problem, the graph coloring problem and the maximum clique problem. The hardest problems in the NP class are called the NP-complete problems; these problems don't have algorithms that run in a polynomial time.

## 1.2 THE PROBLEM

This thesis will address the graph coloring problem and the maximum clique problem. In particular, this study will focus on the efficiency of the largest degree algorithm as outlined in [37], especially when dealing with any general purpose indirect randomly generated graphs as well as special purpose graphs. The study will also introduce a new improvement on that algorithm; this improvement will reduce the number of colors used in the coloring process. The third issue addressed in this thesis is the ability of both

algorithms to match the maximum clique-based coloring of a graph. The study will introduce a measure for the deviation of the algorithms from maximum clique based coloring. This measure can be further used to measure the optimality of graph coloring algorithms. These issues are further explained in the following subsections.

### 1.2.1 THE LARGEST DEGREE ALGORITHM

(Malkawi, et. al; 2008) introduced this algorithm for exam scheduling using graph coloring. The algorithm's idea was derived from the properties of exam scheduling, namely, universities tend to first schedule classes with relatively large number of students and large cross registration with other courses. Such courses, in graph terms, are said to have large degrees. Taking care of such courses first, and then moving on to take care of all the courses in which students in the first course are registered, allows for resolving conflicts between exams in a systematic manner. See Figure 1.8

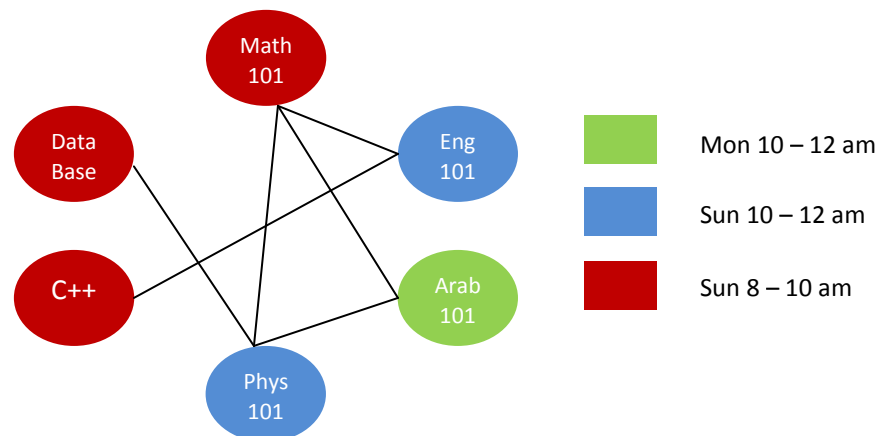


Figure 1.8 an indirect unweighted graph  $G$  represents exam schedule in a particular university.



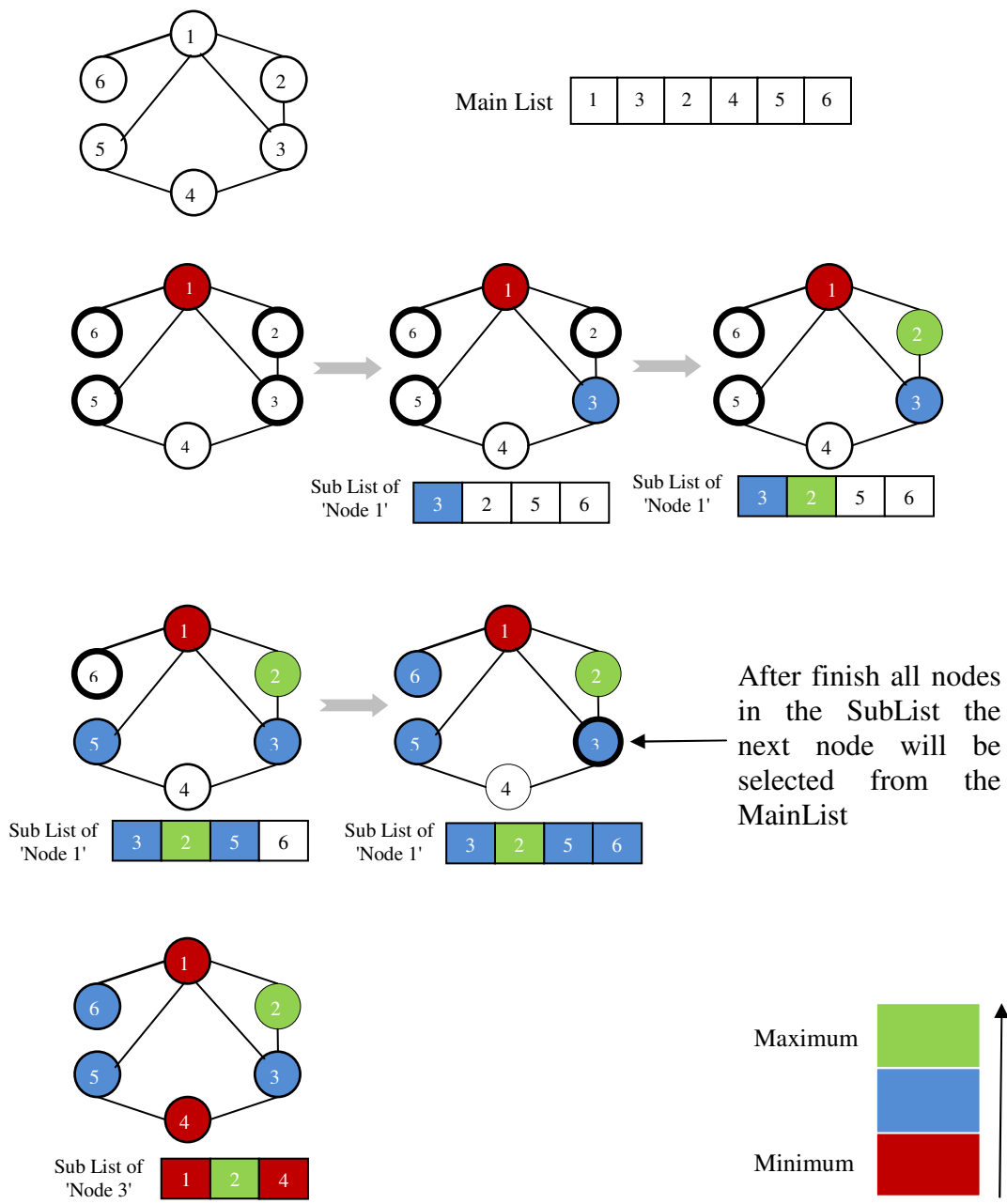
Important notes on the previous Figure:

1. Each node represents a specific course.
2. Each color represents an exam time slot.
3. Edges between pairs of nodes indicate that there are some common students registered in both courses, so they have both exams, and thus both exams cannot be scheduled at the same time.
4. According to graph theory, no two adjacent nodes should have the same color, which means no two exams that have common students should be held at the same time.
5. A node with larger degree represents a course in which students are registered to many other courses

By applying the largest degree algorithm on the previous graph, the coloring process will start by sorting the courses in decreasing order according to the number of adjacent nodes (degree); if two nodes have the same degree, then choose the one with the largest ID, so the courses will be sorted as follow Math 101, Phys 101, Eng 101, Arab 101, Data Base, C++. The previous list will be saved in a list, for example (MainList), after that the first course in the MainList (Math 101) will be selected to be colored first, it will be colored with the minimum available color (Red). Next courses that are adjacent to the selected node will be sorted in decreasing order and will be saved in another list, for example (SubList)( Phys 101, Eng 101, Arab 101), and then courses in this list will be colored one by one by choosing the minimum available color for each course. After coloring all courses in the SubList, the algorithm will pick the next course form the MainList (Phys 101). And so on. Following is a description of the algorithm:

1. Sort nodes based on degree in descending order.
2. Select the first node in the list; color the node with smallest available color (use colors, 1, 2, 3....)

3. List the neighbors of the selected node.
4. Sort the neighbors of the selected node in descending order based on degree, if two or more nodes have the same degree, then the one with the largest ID is ordered first.
5. Color the neighbors of the selected node, starting with the first node in the list, for each node; check all its neighbors which have already been colored. Color the node with the smallest available color.
6. When all neighbors of the selected node have been colored; go to the next node in the main list of nodes.
7. Go to step 3.
8. Stop when all nodes have been colored. See Figure1.9.



**Figure 1.9 Graph coloring example using the largest degree algorithm**

Generally, graphs can be classified under two general categories, the first type is special graphs where the maximum number of colors needed for these graphs is determined directly according to the type of graph. For example, the 5-coloring graphs need 5 colors at most to color the graph, while the 4-coloring planer graphs need 4 colors, and

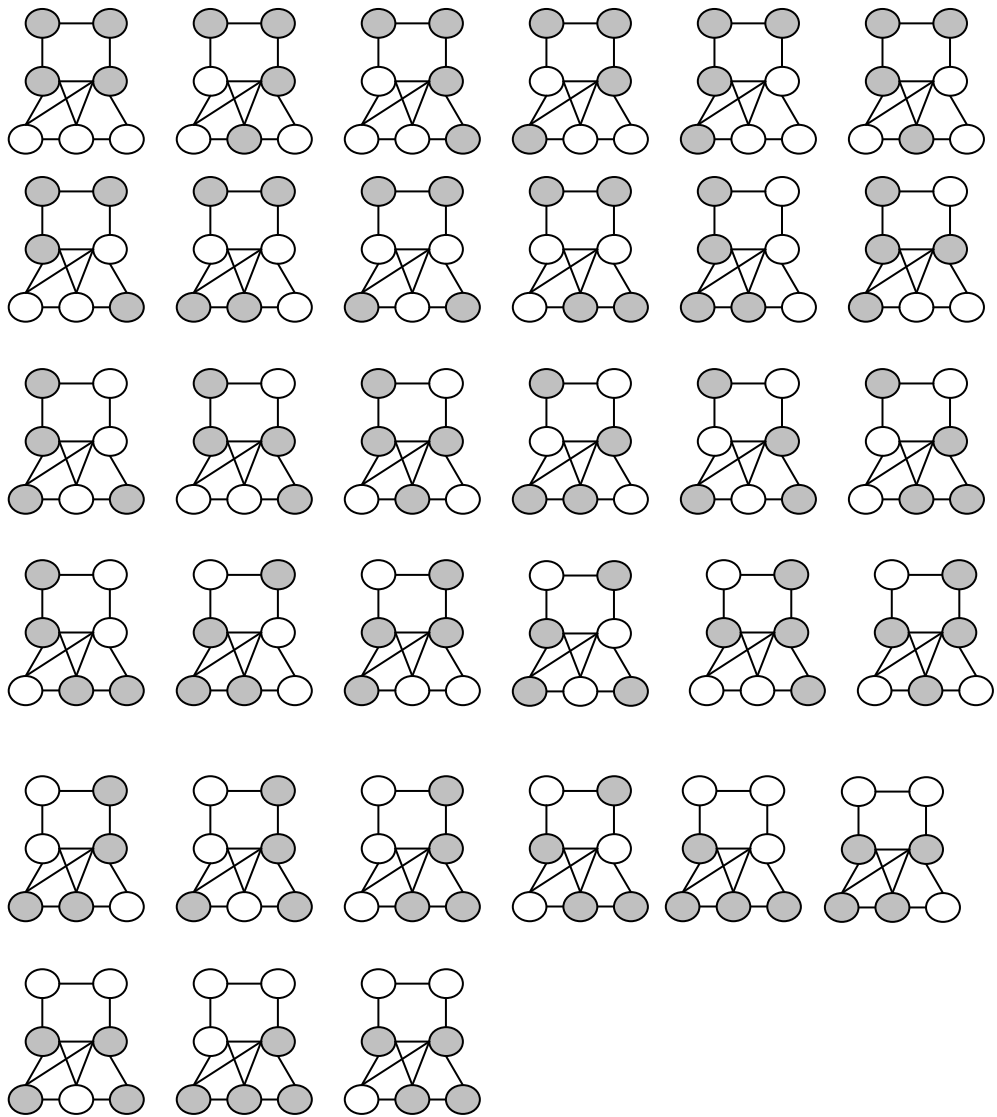
so on. The second type is the general purpose graphs, this type of graphs doesn't have any rules that constraints the connection between its nodes. One way to know the minimum number of colors needed to color these graphs is based on the maximum clique(s) in the graph.

The largest degree algorithm has been shown to succeed in coloring the special graphs with the minimum number of colors [38]. The algorithm will color one of the famous 5-coloring graphs with 4 colors. The algorithm will also color all 4-coloring planer graphs with exactly 4-colors. Graphs, known as 6-triangulation graphs will be colored with 5 colors. Such graphs are known to be 5-colorables.

The first problem addressed in this thesis is the extent of the success of the largest degree algorithm as well as the modified algorithm in coloring the general purpose graphs with the minimum number of colors.

### **1.2.2 CLIQUE DETECTION PROBLEM**

The problem of finding the maximum clique in an indirect graph is considered one of the NP-complete problems. This means that there is no known algorithm for solving this problem in a polynomial time, except for those which have been developed for specialized graphs such as the planer graphs or perfect graphs where the problem can be solved in a polynomial time. Solving this problem can also be done in polynomial time if  $k$  is constant, where  $k$  is the number of nodes in the clique; in this case all subgraphs of at least  $k$  nodes will be checked whether it form a clique or not. This method is called the brute force algorithm See Figure 1.10 [21].



**Figure 1.10 Finding 4-clique in a graph of 7 vertices using the brute force algorithm. This required checking all the 4 nodes combinations in order to determine if any combination compose a clique or not. Number of checks is equal to 35.**

In this thesis, we will develop a measure to evaluate the level of proximity of coloring algorithms to the maximum clique detection algorithm. The measure will show if the coloring algorithm colors the nodes of the maximum clique first, and if not, it will show the level of deviation from the clique.

### **1.2.3 THE MODIFIED LARGEST DEGREE ALGORITHM**

As mentioned above, the largest degree algorithm starts by sorting the nodes in decreasing order, then it assigns the minimum available color to the first node in the list, then it lists the neighbors of the selected node and sort them in decreasing order based on the degree, next it colors the neighbors of the selected node starting with the first node in the list, for each node, it checks all its neighbors which have already been colored, and then it colors the node with the smallest available color.

The problem arises when two or more nodes have the same degree; the question is "Which node should be chosen first?" The largest degree algorithm assumes that the node which has a greater ID number must be chosen first, this assumption has no impact on the performance of the graph coloring process.

In this thesis we provide a new criteria that will allow the algorithm to choose the next node more intelligently, which will have an impact on the performance of the algorithm.

### **1.3 OBJECTIVES OF THE STUDY**

The major objectives of this thesis are:

1. Evaluating the performance of the largest degree algorithm [38]
2. Developing an improvement version of the same algorithm.
3. Comparing the performance of the largest degree algorithm with the modified algorithm in terms of number of colors needed to perform the coloring process.
4. Measuring the proximity of the largest degree algorithm (the original and the modified) from the maximum clique delectability.

## 1.4 SCOPE AND SIGNIFICANCE OF THE STUDY

This study will address a special kind of problems which is categorized under the NP-completeness problem (Hard problems). Until now, no better algorithm than an exponential time algorithm for such problems is known, although many heuristic algorithms give us reasonably good approximations of the optimal solutions. NP-completeness is thought as a class in which if one of such problems is solved in polynomial time, all instances of NP-complete problems are reducible to a polynomial time problem. Then, it is expected to be a remarkable contribution to modern science field.

The study will focus on analyzing the behavior of the underlined coloring algorithm in order to get some results about its ability to follow the clique track while coloring the vertices, and then we can determine the effectiveness of this algorithm in finding the minimum number of colors needed to coloring the graph; by demonstrating this factor we can make use of it in modeling many practical problems. Managing social networks over the internet is one of the challenging problems that can be modeled using the graph coloring and the maximum clique, the group which contains the maximum number of people and each one in this group knows all the participants can be modeled as the maximum clique, and the groups that contain fewer participants and each one knows all members can be modeled as cliques. Then we can manipulate these cliques in such social networks in a rather smooth manner [7].

## CHAPTER TWO

### LITERATURE SURVEY

---

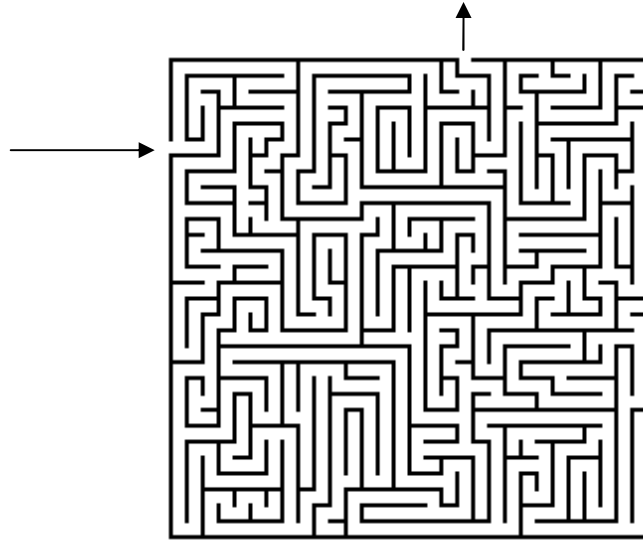
In this chapter we present a brief description of what has been done by other scientists in this area of research. We introduce some methods addressing the graph coloring problem such as the backtracking method, the local search method and the greedy method.

#### **2.1 GRAPH COLORING USING THE EXHAUSTIVE SEARCH**

##### **(BACKTRACKING)**

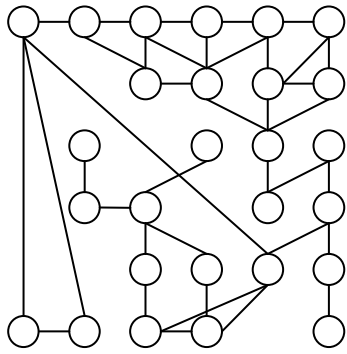
One of the most well known methods in graph coloring is the exhaustive search method used for solving the maze problem; see Figure 2.1. The process starts by choosing particular path, if you reach a dead point, you need to backward one step to the last joint and choose another path; if you reach the required destination then you solve the maze, else if you continue to go backward till you reach the start point, then the maze is unsolvable.



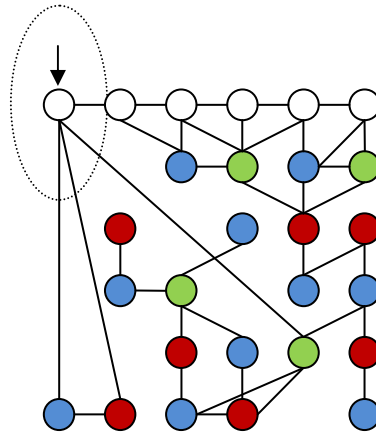


**Figure 2.1 solving the maze using the backtracking method**

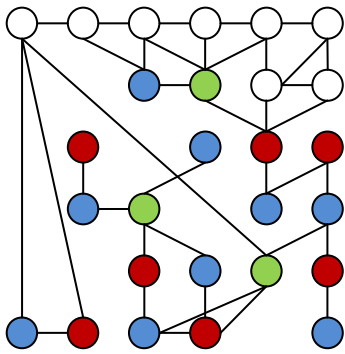
This is very similar to the graph coloring problem, See Figure 2.2. The exhaustive search algorithm is very trivial but at the same time it is very costly, simply it is enumerating all possible choices to color the graph with a minimum of  $k$  colors. The process starts by coloring the nodes one by one with initial value of  $k = 2$ ; if you reach a situation where you can't continue, which means it is impossible to color the next node without incrementing  $k$  by one, you have to go backward one step and retry with another choice. If you continue to go backward and retry all the choices till you reach the starting node, then the graph can't be colored with  $k$  colors, and we should increment  $k$  by one.



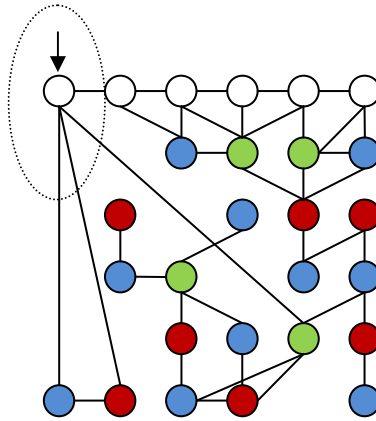
A-Using the exhaustive search to color indirect graph of 27 nodes with only 3 colors. Nodes will be colored line by line from bottom to up.



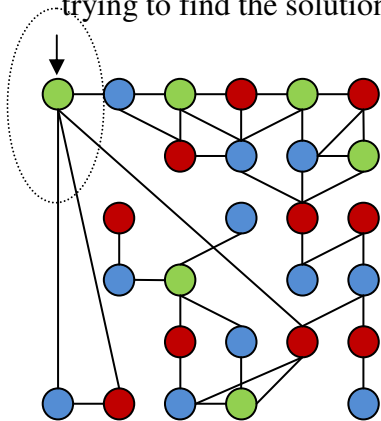
B –the problem arises when attempting to color the node which is in the circle. In order to solve this problem we have to backward a step and retry another coloring choice.



C-Backward two steps, trying to find the solution.



D-Another coloring has been chosen but the problem still exist.



E-Finally; and after (65448) steps the solution was found.

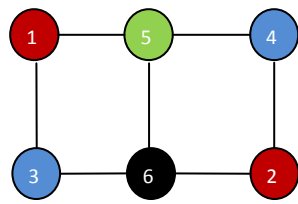
Figure 2.2 The exhaustive search algorithm for solving the graph coloring

The main drawback of this method is its complexity; this method will be run in exponential time ( $k^n$ ), which means that this algorithm is applicable to small graphs. Real world applications such as social networks are relatively large graphs and the time complexity is prohibitive when using exhaustive coloring methods [13].

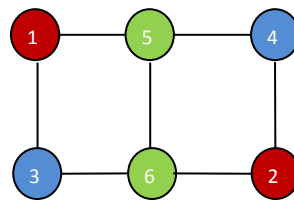
## **2.2 LOCAL SEARCH METHOD**

Local search algorithms are used in solving many NP-complete problems such as the traveling sales man problem and the course scheduling problem as well as the graph coloring problem. Local search algorithms are generally divided into two categories, the first is used to optimize the results which are generated by other algorithms. The second category consists of standalone algorithms which are considered more complex.

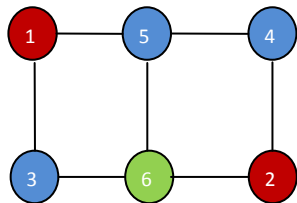
Figure 2.3 presents an example of the min-conflicts algorithm [41]. This algorithm starts by a valid coloring which is generated from applying a particular algorithm, and then attempts to reduce the number of colors in this graph. This reduction will lead to a number of conflicts that should be removed by the algorithm. At the end of this algorithm a better coloring could be produced.



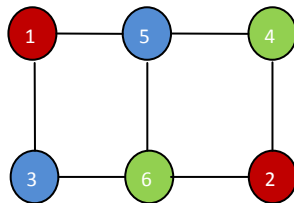
A-Coloring resulted from applying a particular algorithm.



B-Trying to reduce the number of colors by coloring the node number 6 with green instead of black. This will cause the first conflict between node 5 and node 6.



C-Trying to resolve this conflict by coloring the node 5 with blue. This will cause the second conflict between node 5 and node 4.



D-Assigning the green color to node 4 will resolve the conflict and producing a better coloring than the previous one (which is in A).

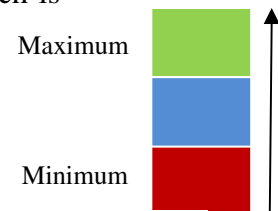


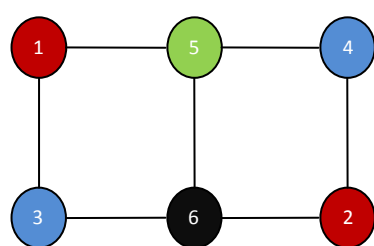
Figure 2.3 The min-conflicts algorithm.

## 2.3 GREEDY COLORING

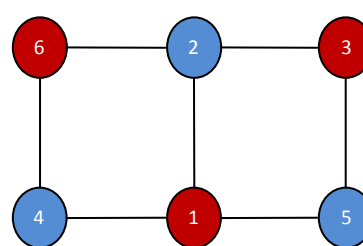
A greedy algorithm is any algorithm that follows the problem solving heuristic of making a local optimal choice at each stage [23]. In general, greedy algorithms don't use the minimum number of colors; however greedy algorithms manage to go around the NP problem, by coloring the graph with few colors in a considerable time.

Practically, the greedy algorithm takes every node in turn in some predefined order and tries to color this node with one of the already consumed colors; if it's not possible to use any of the consumed colors it will assign a new color to the node. The predefined order particularly important for the coloring process. Next, we address two types of this “predefined order” policy, the first is regarding the nodes, and the second is regarding to the colors.

Visiting the nodes in different orders will produce different coloring scheme, with different chromatic numbers. For example let  $G$  be an indirect graph of six nodes (See Figure 2.4.a), and let the order of visiting the nodes be (1, 2, 3, 4, 5, 6). In this example, 4 colors are required to complete the coloring of the graph. Let the second order be (6, 5, 4, 3, 2, 1); this order will reduce the number of required colors from 4 colors to only 2. See Figure 2.4.b. Note that number inside the node doesn't represent value of node rather it indicates the order of this node in visiting.



A-Bad order of nodes produces a costly coloring.



B-Good order of nodes reduces the number of colors.

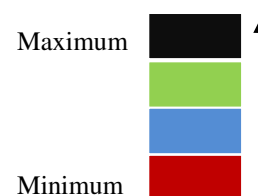
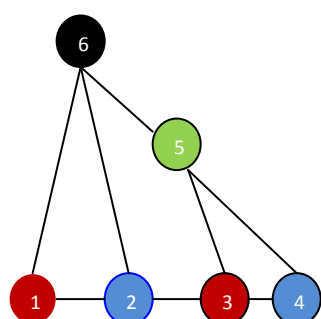


Figure 2.4 Predefined orders of visiting graph nodes

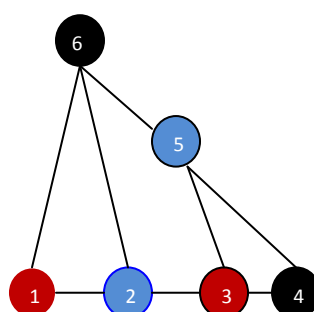
The previous example has illustrated the importance of choosing the best order for nodes. Many algorithms have been proposed for this purpose, and the algorithm in this thesis is one of these algorithms.

The second type of predefined order is regarding to the colors, the importance here is on how to find the best criteria for choosing the next color in order to reduce the number of colors.

To clarify the idea let  $G$  be an indirect graph of six nodes, (See Figure 2.5.a), and let us consider the minimum available color to be the criteria that will be used to choose the next color. The process starts by coloring node 1 with red which is the minimum color, then it colors node 2 with blue which is the minimum available color after the red color has been taken, after that the available color is the red which is assigned to node 3, node 4 is colored with the available color which is the blue, then node 5 is colored with green because there is no available colors, finally node 6 is colored with another new color which is the black. As a result we need four colors to complete the process.



A- This coloring requires four colors.



B- This coloring requires three colors.

Figure 2.5 Predefined orders of colors

However in the second graph (See Figure 2.5.b) the strategy of choosing the next color is different. The new strategy states that there is no conditions of choosing the next color as long as the next color is legal for coloring. As a result of the new strategy node 4 is colored with black instead of blue (which is the minimum available according to the previous strategy), and the number of required colors is reduced to three instead of four.

## CHAPTER THREE

### THE MODIFIED GRAPH COLORING ALGORITHM

---

In chapter one we gave a brief description of the largest degree algorithm. In this chapter we will introduce the modified algorithm in more details. This chapter will be divided into three sections; the first section introduces the concept of degree saturation; the second section explains how the modified algorithm proceeds to color the graph; the third section explains how the modified algorithm helps in solving the maximum clique problem.

#### 3.1 SATURATION DEGREE

In 1979 the scientist Brelaz introduced a new algorithm for graph coloring [11]. That algorithm basically depends on the Saturation Degree (SD) of every node. The term Saturation Degree refers to the number of differently colored nodes adjacent to a particular node. See Figure 3.1 for illustration.

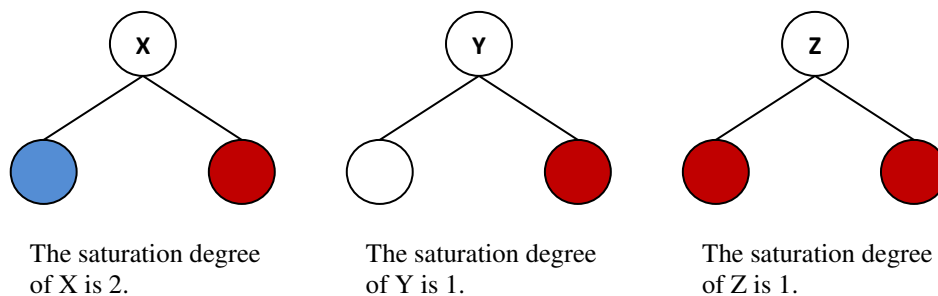


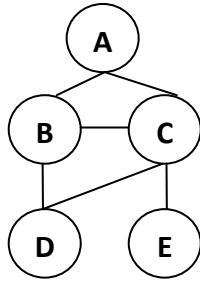
Figure 3.1 Saturation Degree



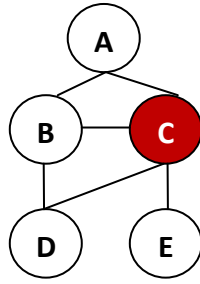
According to the SD approach nodes with larger SD will be colored first. Following is a description of the algorithm, based on the concept of SD:

1. Arrange the nodes by decreasing order of degrees.
2. Color a node of maximal degree with color 1.
3. Choose a node with a maximal saturation degree. If there is equality, choose any node of maximal degree in the uncolored subgraph.
4. Color the chosen node with the least possible (lowest numbered) color.
5. If all the nodes are colored, stop. Otherwise, go back to 3.

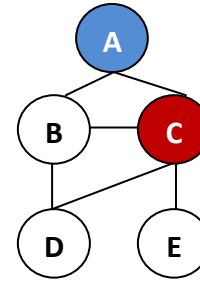
To clarify the idea of this algorithm, See Figure 3.2 which explains the algorithm step by step.



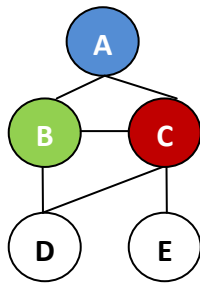
C should be colored first because it has the largest degree



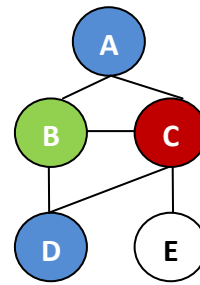
A, B, D, E, and H have the same saturation degree which is equal to 1. . Select node A for coloring (minimum ID number) and assign the color blue to node A



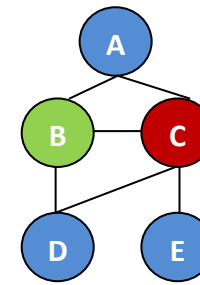
B has a saturation degree of 2, which is the maximum. B will be colored with the minimum available color (Green)



D has a saturation degree of 2, which is the maximum. D will be colored with the minimum available color (Blue)



Finally E will be colored with the minimum available color (Blue) .



Graph is colored

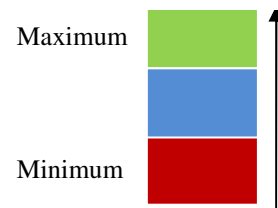


Figure 3.2 the SD approach

According to [11], the SD is exact for the bipartite graphs (special type graph), it also produces a good coloring quality for the general purpose graphs and it could be run in  $O(n^3)$ .

### 3.2 GRAPH COLORING USING THE MODIFIED ALGORITHM

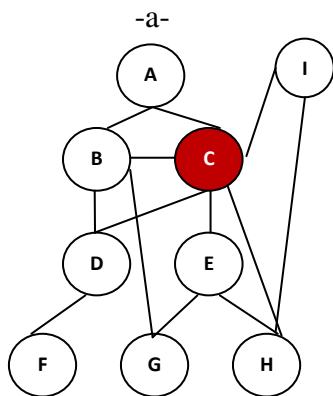
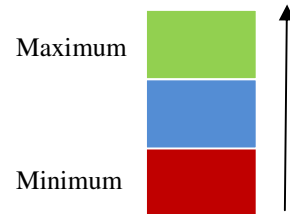
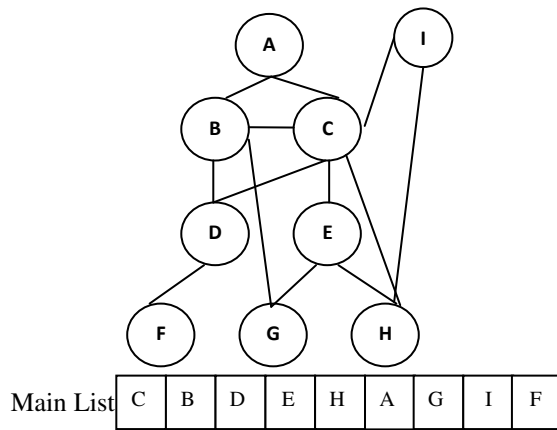
The need for this modification arises when two or more nodes have the same degree, the original algorithm chooses the one with the larger (or lowest) ID, which has no impact on the coloring quality. The modified version uses the SD concept for determining the next node to be colored.

The algorithm states that if two or more nodes have the same degree, then the node which has fewer available colors should be colored first. In other words if two or more nodes have the same degree, then the node which has larger saturation degree will be colored first. Following is a description of the algorithm:

1. Sort nodes based on degree in descending order.
2. Select the first node in the list; Color the node with smallest available color (use colors, 1, 2, 3....).
3. List the neighbors of the selected node.
4. Sort the neighbors of the selected node in descending order based on degree, if two nodes have the same degree, choose the node which has a greater saturation degree.
5. Color the neighbors of the selected node, starting with the first node in the list. For each node; check all its neighbors which have already been colored. Color the node with the smallest available color.
6. When all neighbors of the selected node have been colored; go to the next node in the main list of nodes.
7. Go to step 3.
8. Stop when all nodes have been colored.

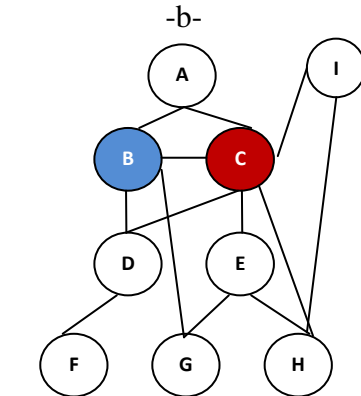
Figure 3.3 shows an example of the modified algorithm, it starts by ordering the nodes in the main list in a decreasing order based on the degree of each node. The first node in the list (C) is selected to be colored with the minimum available color (Red). Then the neighbors of (C) are sorted in a sublist according to the degree of each neighbor. See Figure 3.3.a. The first node in the sublist (B) is selected to be colored with the minimum available color (Blue), then the sublist and the main list are reordered based on the degree and the saturation degree. Note that the sublist puts D before E and H, despite of having the same degree; that's because D has a greater saturation degree than E and H. See Figure 3.3.b. in the next step, D is colored with a new color (Green), this color is chosen because Red and Blue were taken, and so the minimum available is the green color. See Figure 3.3.c.

Figures 3.3.d-g are repetitions for the previous steps; in 3.3.g the algorithm has colored all of the sublist nodes, which are related to node C and it will proceed with the next node in the main list. The next node in the main list is B which is already colored. Figure 3.3.h represents the sublist for B, it includes C, D, A, and G, all of which are colored except G which will be colored with red. Figure 3.3.i represents the next node in the main list (D), which is already colored. The last Figure 3.3.j represents the last step of the algorithm; which colors the last node in the sublist.



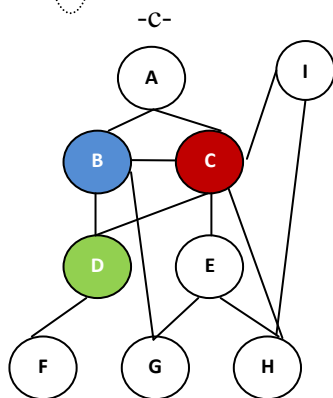
Sub List of 'Node C'

B	D	E	H	A	I
4	3	3	3	2	2
1	1	1	1	1	1



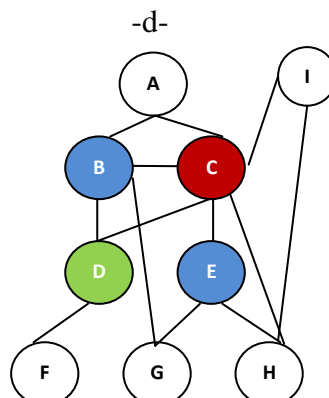
Sub List of 'Node C'

B	D	E	H	A	I
4	3	3	3	2	2
1	2	1	1	2	1



Sub List of 'Node C'

B	D	E	H	A	I
4	3	3	3	2	2
2	2	1	1	2	1



Sub List of 'Node C'

B	D	E	H	A	I
4	3	3	3	2	2
2	2	1	2	2	2

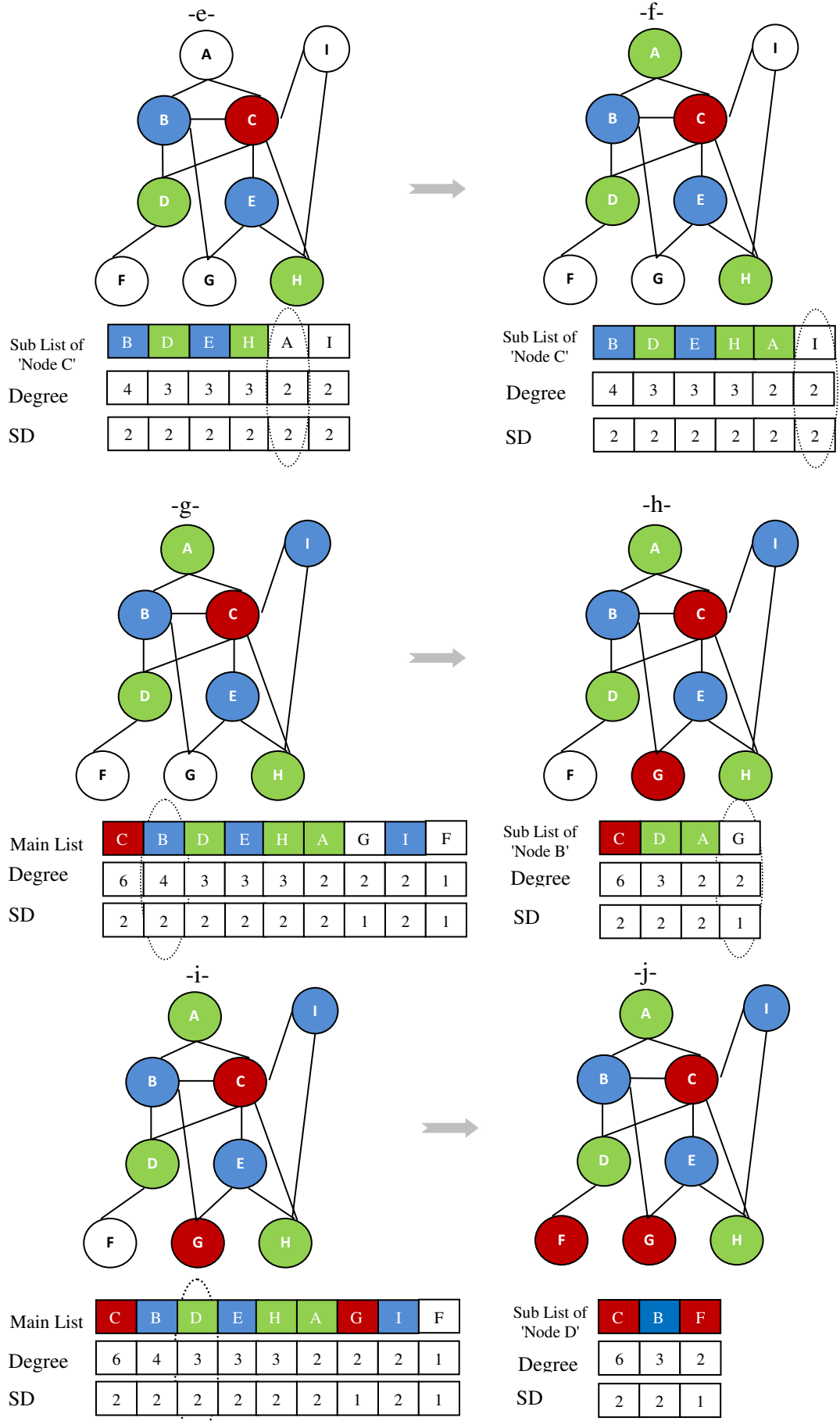


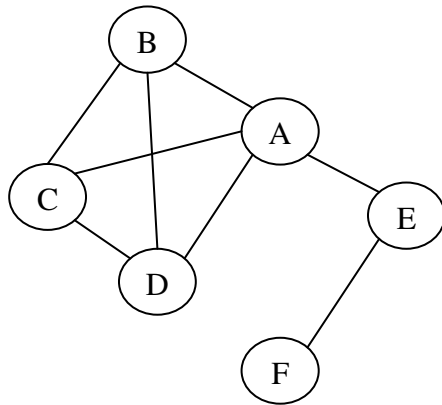
Figure 3.3 Graph Coloring using the modified algorithm

### **3.3 CLIQUE DETECTION USING THE MODEFIED ALGORITHM**

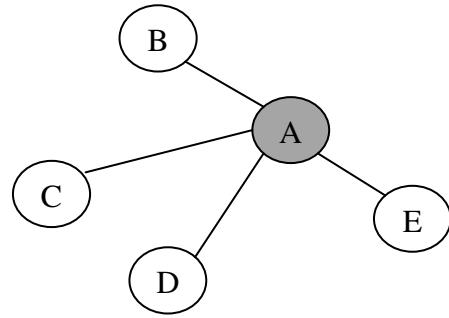
The second problem addressed in this thesis is the maximum clique detection; in the preceding chapters we have defined this concept. In this section we will explain how we can benefit from the modified algorithm in solving the maximum clique detection problem.

#### **3.3.1 GRAPH PARTITIONING:**

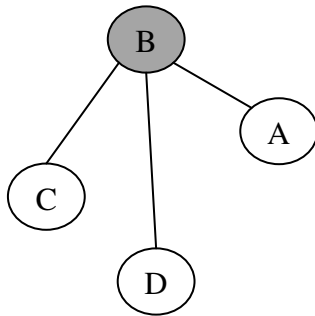
Graph will be partitioned into smaller subgraphs, each one is composed of a node (later will be called the leader node) and all of its adjacent nodes. For example let  $G$  be an indirect graph, and let  $N$  be the set of nodes in the graph, Applying the partitioning process on  $G$  produces  $N$  subgraphs. Following is graphical illustration of the partitioning procedure.



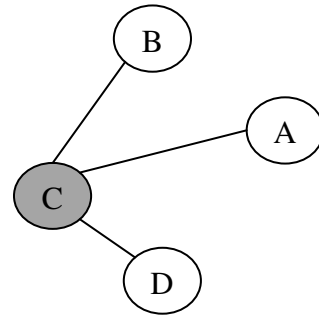
The original graph



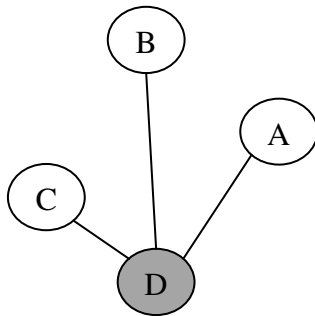
A is the leader node,



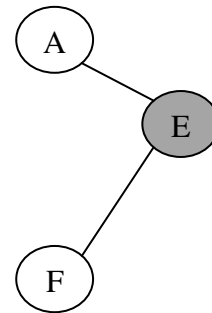
B is the leader node



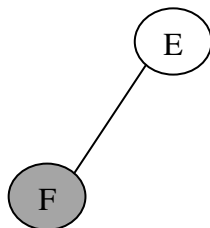
C is the leader node



D is the leader node



E is the leader node



F is the leader node

Figure 3.4 Graph Partitioning



The previous diagram shows how the indirect graph  $G$  which is composed of six nodes produces six subgraphs of the described type, that is each subgraph consists of only from the leader node and its neighbors.

### **3.3.2 CLIQUE FINDING**

After partitioning the indirect graph into the  $N$  subgraphs of the described type, we will apply the coloring algorithm on each subgraph. We begin with the subgraph whose leading node has the largest degree; then we move to the next subgraph with the next highest degree and so on, within each subgraph, we traverse the nodes starting from the node with the largest degree and moving down to lower degree nodes.

When this algorithm was applied to special graphs [38], the algorithm produced minimum number of colors. For general purpose graphs, we propose a new measure to evaluate the efficiency of the algorithm. This measure depends on how closely the algorithm colors the nodes of the cliques in the graph, starting with the largest clique first.

The purpose of applying this algorithm is not to detect the exact clique in each subgraph; rather the purpose is to analyze the improved algorithm in terms of how closely the algorithm follows the maximum clique-based coloring of a graph.

In other words the clique of each subgraph will be known in advance using a particular algorithm called the brute force algorithm. After applying both algorithms on the same graph we will do a comparison between the two results. Upon this comparison we will discover how closely our improved algorithm follows the maximum clique.

The clique of each subgraph will be fully recognized and detected using a brute force algorithm. When we apply our algorithm to the same graph, we will trace the coloring path taken by the algorithm and test it against the cliques of the graph. A complete matching exists if the algorithm colors all the nodes of the clique before it moves to another subgraph. A partial matching exists when the algorithm colors nodes from the clique and then moves to another set of nodes before it returns back to color the rest of the clique nodes. The efficiency of the algorithm will be determined by the rate of matching between the set of colored nodes and the nodes of the clique and by the distance between the nodes colored outside the clique and the nodes of the clique.

If the clique has  $N$  nodes, and the algorithm colors  $M$  nodes first where  $M \leq N$  then the rate of convergence is given by  $\rho = M/N$ . The deviation rate between the clique and node ( $\gamma_i$ ) colored outside the clique is given by  $\delta = [R(\gamma_i) - N] / R(\gamma_i)$ , where  $N$  is the size of the clique and  $R(\gamma_i)$  is the order of coloring node ( $\gamma_i$ ).

For example, assume that a clique in a 7 nodes graph has 4 nodes {1, 2, 3, 4}, ( $N=4$ ) and the algorithm colors the graph in the following sequence [1, 2, 4, 6, 7, 3, 5]. The convergence rate  $\rho = [3/4] = 0.75$ . And the deviation rate between node 3 (colored outside the clique sequence) is given by  $\delta = (6-4)/6 = 0.33$ .

The average deviation rate is given by  $\Delta = \sum_{i=1}^N \delta_i$

The computation of the rate of convergence and the rate of deviation will not add to the complexity of the algorithm except for record keeping.

The implementation of the algorithm proceeds as follows:

1. Let the list ColorsOrderList= $\emptyset$ .

2. Start the coloring process by coloring the leader node with the minimum color. Then add it to ColorsOrderList.
3. Sort the neighbors of the leader node in a decreasing order based on the degree of each neighbor, if two nodes have the same degree choose the node which has a greater saturation degree.
4. Select the first node in the list, color it with the minimum available color, and then add it to ColorsOrderList.
5. Repeat step 4 until all nodes are colored.
6. At the end compare ColorsOrderList with CliqueList and report the results. See Figure 3.5

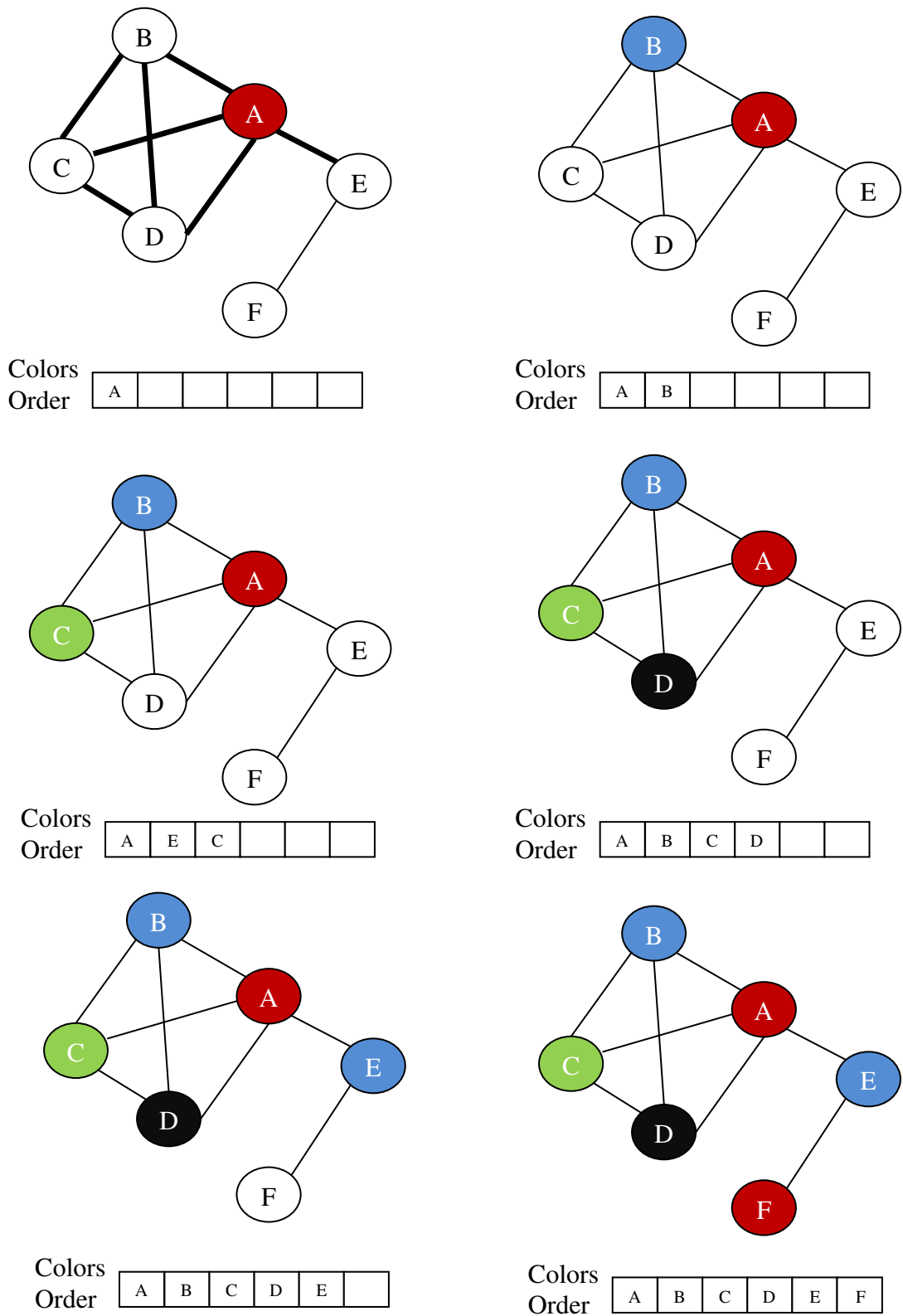


Figure 3.5 Recoding the coloring track using the ColorsOrderList

### 3.3.3 CLIQUE FINDING EVALUATION

Evaluating the performance of the algorithm in detecting the clique depends on the number of the nodes which are related to the same clique and which have been colored before going outside the clique track, and the greater the number of colored nodes which belong to the clique the better is the algorithm.

In other words, in order to check the performance of this algorithm we have to do a comparison between the nodes in ColorsOrderList with the others in the CliqueList and then report the results. For example the previous Figure produces the following ColorsOrderList {A, B, C, D, E, F}, while the CliqueList is {A, B, C, D}, by comparing each node in ColorsOrderList with all nodes in CliqueList, we have founded that our algorithm has succeeded in detecting all nodes in the clique with the following convergence rate :

$$\rho = M/N = 4/4 = 100\%.$$

The second part of the evaluation is to count how many nodes are colored before returning back to the clique nodes, i.e., the rate of deviation from the main course of the clique ( $\delta$ ). For example let Clique1 be the first detected clique in the first subgraph and let the CliqueList that represents this clique be {A, B, C, D} and let the ColorsOrderList be {A, B, E, F, Q, R, C, D}. In this example,  $\rho=0.5$ ; the algorithm succeeds in coloring the first two nodes A and B which belong to clique1, then it colors E, F, Q and R before returning back to C and D which belong to clique1. The distance of the first two nodes in the clique A and B is equal to zero because they are colored within the clique order, while the distances of node C and D are 2 and 3 respectively. The deviation rate for

nodes C and D is given by  $\delta_1 = 2/7$  and  $\delta_2 = 3/8$  respectively. The average deviation rate is  $\Delta = (0.29 + 0.38)/2 = 0.33$ .

The convergence and deviation rates can be combined with one general index, used to measure the overall efficiency of the algorithm. We call this index the Clique Conformance Index (CCI) which is defined by:  $CCI = \rho/\Delta$ . In the above example,  $CCI = 0.5/0.33 = 1.5$ . The larger the CCI, the better is the algorithm. CCI accounts for odd cases, such as when the convergence rate is very high say 0.9 (only node is colored outside the clique). But the distance of this node is very large, say 0.9. Then CCI is  $0.9/0.9 = 1$  (lowest index). However, if the node distance is small, say 0.1, then the CCI index is  $0.9/0.1 = 9$ .

In order to simplify the process, results will only be reported for the first subgraph which contains the node with the largest degree. Chapter 4 explains the implementation in more details.

## CHAPTER FOUR

### IMPLEMENTATION

---

This algorithm has been implemented using JAVA programming language; this chapter discusses the implementation details such as the data structure used to represent the graph and the methodology used in generating graphs with different density. Also this chapter provides explanation of some other important classes and methods.

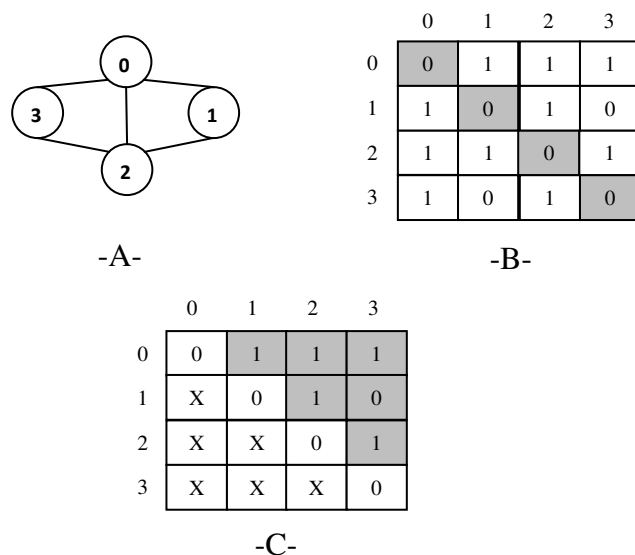
#### 4.1 GRAPH GENERATION

This is the first stage in the implementation process. In this stage a random indirect graph with a variable size and density will be created, but before that we must firstly choose the most appropriate data structure to represent the graph.

Graphs will be represented using the Adjacency Matrix which is a two dimensional matrix of zeros and ones used to represent which nodes of a graph are adjacent to which other nodes. A length of dimensions of adjacency matrix is equal to the number of nodes in the represented graph. For example a matrix of 5\*5 is representing a graph of 5 nodes.

The adjacency matrix will be filled up with random values of zeros and ones, so that every entry has the potential to be assigned to zero or to one. The value of zero means that there is no edge between the node which has a value identical to the row number and the other node which has a value identical to the column number. While the value

of one means that there is an edge between both nodes. Below is an example of graph representation using the adjacency matrix. Note that the diagonal entries will be forced to be zeros, since our graphs have no loops.

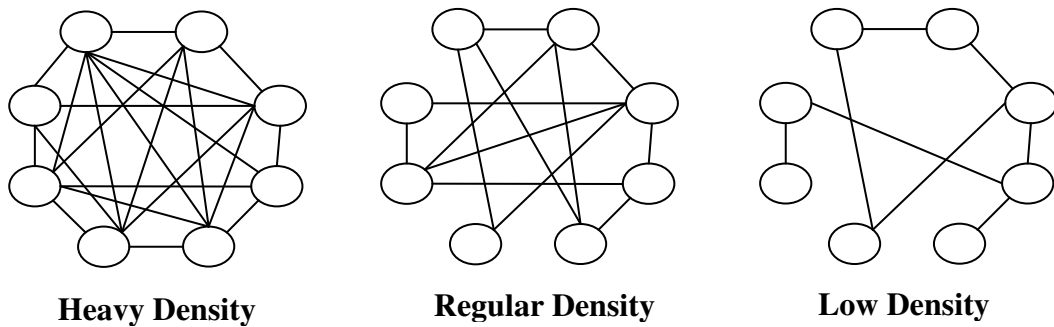


**Figure 4.1 Indirect graph and its corresponding adjacency matrix**

In the last Figure it's clear that the adjacency matrix is a symmetric matrix which means that cells in the upper right are the same as the cells in the lower left, so in order to reduce the processing time and the space required, only one of the two sides will be filled up with values, while the second half will be ignored. See Figure 4.1.C

Size of the graph must be variable which means the user can enter any size he/she wants, Density of the graph should also be variable, our algorithm will deal with three types of densities; Heavy density, Regular density and Low density. Density here is defined as the probability of a pair of nodes being connected, so that graphs with a heavy density will have more edges than graphs with regular or low density, because the probabilistic to find to connected nodes is greater. See Figure 4.2





**Figure 4.2 Heavy density graph, Regular density graph and Low density graph.**

```

1 public class GraphGenerater {
2     public static void main(String args[]) throws IOException {
3         int NumberOfNodes = 1000;
4         double HeavyDensity = .75;
5         double RegularDensity = .5;
6         double LowDensity = .25;
7
8         int NumberOfEdges_H = 0;
9         int NumberOfEdges_R = 0;
10        int NumberOfEdges_L = 0;
11
12        int [][] HeavyMatrix = new int [NumberOfNodes][NumberOfNodes]
13        int [][] RegularMatrix = new int [NumberOfNodes][NumberOfNodes]
14        int [][] LowMatrix = new int [NumberOfNodes][NumberOfNodes]
15
16        for (int i = 0 ; i < NumberOfNodes ; i++){
17            for (int j = 0 ; j < NumberOfNodes ; j ){
18                HeavyMatrix [i][j] = -1;
19                RegularMatrix [i][j] = -1;
20                LowMatrix [i][j]= -1;
21            }
22        }
23    }
24 }

```

**Figure 4.3 GraphGenerater class (main method - declaration part)**

The previous figure shows a section of code from GraphGenerater class, (lines 4-6) contain the declaration of three double variables, and its assignments to specific values; these variables are important to create variant types of graphs. Lines 7-9 contain a

declaration of another three variables for counting the number of edges in each graph. (Lines 10-12) declare three matrices with different names, the purpose of these metrics is for representing graphs in the computer system, HeavyMatrix holds the heavy density graphs, the RegularMatrix holds the regular density graphs and the LowMatrix holds the low density graphs. The rest of this code (Lines 14-18) represents assigning the adjacency matrixes to initial values.

After declaring and initializing the required variables, the next step is to fill up the adjacency matrices with random values of zeros and ones depending on the density type. This can be done by calling the method CreateGraph. See Figure 4.4 That shows three calling statement , each one sends a three different parameters.

```

1 HeavyMatrix = CreateGraph(HeavyDensity , NumberOfNodes , HeavyMatrix);
2 RegularMatrix=CreateGraph(RegularDensity,NumberOfNodes,RegularMatrix);
3 LowMatrix = CreateGraph(LowDensity , NumberOfNodes , LowMatrix);

```

**Figure 4.4 CreateGraph calling statements**

The previous three statements call the same method but with different parameters, the body of CreateGraph method is in the following diagram. See Figure 4.5.

```

1 public static int [][] CreateGraph (double Density , int NumberOfNodes , int
2 AdjMatrix [][]){
3 for (int row = 0; row < NumberOfNodes; row++) {
4     for (int col = row+1 ; col< NumberOfNodes ; col++){
5         if (row == col){
6             AdjMatrix [row][col]= 0;
7             continue;}
8         double rand = Math.random();
9         if (rand <= Density ){ rand = 1 ;}
10        else{ rand = 0 ;}
11        if (AdjMatrix [row][col] == -1){

```

```

12     AdjMatrix [row][col]= (int) rand;}
13 }
14 }
15 return AdjMatrix ;
16 }

```

**Figure 4.5 CreateGraph body**

As previously said we don't need to fill up all the cells in the graph, the only needed cells that on the upper right part of the matrix. That's why the second loop always restarted from the value row+1 in line 4.

Lines 4-6 include if statement, the function of this if statement is to prevent occurrence of loops in the graph, so that all of the cells in the matrix diagonal are assigned to zero. Line 10 uses Math.random() method to generate random value in range of 0-.99, this value is stored in variable rand, in the next line the variable rand is compared with the parameter Density, if the rand value is less than or equal to the Density value then rand will be assigned to zero, else rand will be assigned to one . Finally the value of rand will be assigned to a cell in the adjacency matrix.

Returning to the main method, Figure 4.6 shows a part of code contains three calls for the WriteInBenchMark method which is responsible for writing the contents of the adjacency matrixes in form of bench marks. The body of this method is represented in Figure 4.7.

```

1 WriteInBenchMark (NumberOfNodes,NumberOfEdges_H,
2 HeavyDensity, HeavyMatrix, "c:\Heavy.txt" );

```

```

3 WriteInBenchMark (NumberOfNodes,NumberOfEdges_R,
4 RegularDensity,RegularMatrix,"c:\\Regular.txt");
5 WriteInBenchMark (NumberOfNodes,NumberOfEdges_L,LowDensity
6 LowMatrix,"c:\\Low.txt");

```

Figure 4.6 WriteInBenchMark Method calls

```

1 public static void WriteInBenchMark (int NumberOfNodes, int throws
2 NumberOfEdges, double Density, int AdjMatrix [][], String BenchName )
3 IOException {
4 FileWriter File_Heavy = new FileWriter (BenchName);
5 BufferedWriter Out = new BufferedWriter(File_Heavy);
6 Out.write("This Benchmark is Created by Abdel Mutaleb Alzoubi");
7 Out.newLine();
8 Out.write("for the purpose of developing coloring graph algorithm");
9 Out.newLine();
10 Out.write("the number of nodes in this graph is: " + NumberOfNodes );
11 Out.newLine();
12 Out.write("the number of edges in this graph is: " + (NumberOfEdges ) );
13 Out.newLine();
14 Out.write("nodes in this graph are connected to each other by " + Density +
15 "% of the total number of nodes");
16 Out.newLine();
17 Out.write("Stop");
18 Out.newLine();
19 Out.write("N "+ NumberOfNodes);
20 Out.newLine();
21 Out.write("E "+ (NumberOfEdges ));
22 Out.newLine();
23 Out.write("Start");
24 Out.newLine();
25 for (int i = 0 ; i < NumberOfNodes ; i ++ ){
26 for (int j = i+1 ; j < NumberOfNodes ; j++){
27 if (AdjMatrix [i][j]== 1 ){
28 Out.write("e "+(i+1) + " " + (j+1) );
29 Out.newLine();}
30 }
31 }
32 Out.close();
33 }
34 }

```

Figure 4.7 WriteInBenchMark body

As a result of running the previous method three benchmarks will be created, each one consists of a particular number of nodes and edges, these benchmarks can be stored on the hard drive for any future need. See Figure 4.8 which shows a screen shot for a benchmark.

```

11-11-77-100.txt - Notepad
File Edit Format View Help
This Benchmark is Created by Abdel Mutaleb Alzoubi
for the purpose of developing coloring graph algorithm
the number of nodes in this graph is: 25
the number of edges in this graph is: 224
vertices in this graph are connected to each other by 0.75% of the total number of vertexes
Stop
N 25
E 224
Start
e 1 2
e 1 3
e 1 4
e 1 6
e 1 8
e 1 10
e 1 15
e 1 16
e 1 18
e 1 19
e 1 20
e 1 22
e 1 23
e 1 24
e 2 3
e 2 4
e 2 5
e 2 6
  
```

Figure 4.8 Screenshot for a benchmark

## 4.2 NODE CALSS

Node.java is the main building unit of the program; it contains important attributes and methods that will be referenced so many times in the program. Table 4.1 lists the attributes and brief description for each of them.

Table 4.1 Node.java Attributes

Attribute Name	Description
Int Value	Stores the value of the node; it could be string, double, char, or any other data type. This program deals with integers so the Value of the node should be stored in integer data type.

LinkedHashSet Adjacents	This list stores the neighbors of a node; it stores the value of each neighbor.
Int NumberOfAdjacents	Store the number of neighbors of a particular node.
Node Next	Point to the next Node in the backtracking method
Node Previous	Point to the previous Node in the backtracking method
int Color	Colors are expressed as integers; the minimum color is the one which has a minimum value. Value (-1) indicates that the node is uncolored.
List PossibleColors	Stores the colors that are legal for a particular node. It stores values in the range of 0 to ColorCount.
int ColorCount	Stores the index of the last visited value of the PossibleColors list. Used in the backtracking method
int Dsatur	Stores the Saturation Degree of a particular node.
LinkedHashSet UnColoredAdjs	Stores the neighbors of the node which are uncolored, it stores the value of each neighbor.

Methods in this class perform important operations; Table 4.2 lists methods name and a brief description about the work of each one.

Table 4.2 Node.java Methods

Method Name	Description
public Node (int x)	A constructor of the class node, it will assign the value of x to the node's value, and it will assign a default values to lists Adjacents and PossibleColors.

Node next()	Returns the next node to a particular node.
AddConnection (int x)	Adds the value of x to the Adjacents list which stores the neighbors of the node.
ColorNode(int x )	Assigns the color x to the node.
int nextColor()	Returns the most appropriate color, by referring to the list PossibleColors with index equal to the ColorCount. Or returning -1 if all attempts have failed to color the node, which means ColorCount is equal to PossibleColors.size()
boolean isValidColor(Graph graph, int color)	Checks whether a particular color is appropriate to be assigned to a node or not.
computePossibleColors (Graph graph, int k)	Computes the possible color for each node in the graph.
ComputeDsatur(Graph graph)	Computes the Saturation Degree for the node.

### 4.3 GRAPH CLASS

This is the main class in the program; it contains only two attributes, a constructor and one method .See Table 4.3 and Table 4.4.

**Table 4.3 Graph.java Attributes**

Attribute Name	Description
int [][] AdjMatrix	The adjacency matrix which represents the graph.
Node [] GraphNodes	The matrix which contains graph nodes.

**Table 4.4 Graph.java Methods**

Method Name	Description
public Graph()	The default constructor, it initializes the AdjMatrix and the GraphNodes
AddEdge (int x, int y)	The purpose of this method is to adjust the adjacency matrix by inserting ones in the corresponding indices. Add the node to the array GraphNodes[], and then calling the method node.AddConnection();

### 4.3 FIXEDVALUES CLASS

FixedValues.java class contains the constants attributes; each attribute has only one unchangeable value in all parts of the program. For example, the NumberOfNodes is assigned to a specific value which represents the actual number of nodes, this value is needed to be constant wherever it is used. See Table 4.5.

**Table 4.5 FixedValues.java Attributes**

Attribute Name	Description
static int NumberOfNodes ;	Represents the actual number of nodes in the graph.
static String FileLocation ;	Represents the physical location of the benchmark on the hard drive.
static int Uncolored;	Specifies the value -1 to distinguish the uncolored nodes.



## 4.4 GRAPHREADER CLASS

The main function of this class is to read the benchmark which is stored in the hard drive, and then convert it to an adjacency matrix representing the actual graph, see Figure 4.9. Line 24 contains a call for the graph.AddEdge method, the purpose of this method is to adjust the adjacency matrix by inserting ones in the corresponding indices. Add the node to the array GraphNodes[], and then calling the method node.AddConnection().

```
1 import java.util.*;
2 import java.io.*;
3 public class GraphReader {
4     public static Graph ReadGraph () throws FileNotFoundException, IOException
5     {
6         BufferedReader R = new BufferedReader(new FileReader(new File
7         (FixedValues.FileLocation)));
8         String line = R.readLine();
9         while(line.charAt(0) != 'N') {line = R.readLine();}
10        StringTokenizer token = new StringTokenizer(line, " ");
11        token.nextToken();
12        FixedValues.NumberOfNodes=Integer.parseInt(token.nextToken().trim());
13        line = R.readLine();
14        line = R.readLine();
15        line = R.readLine();
16        Graph graph = new Graph();
17        while(line != null) {
18            token = new StringTokenizer(line, " ");
19            token.nextToken();
20            int x = Integer.parseInt(token.nextToken().trim());
21            int y = Integer.parseInt(token.nextToken().trim());
22            x--;
23            y--;
24            graph.AddEdge(x, y);
25            line = R.readLine();
26        }
27        return graph;

```

```
28 }
29 }
```

Figure 4.9 GraphReeader.java

## 4.5 LARGESTDGREE CLASS

This class represents the implementation of the largest degree algorithm, it starts by reading the benchmark, then it finds the first clique using the exhaustive search method, next it starts the coloring process using the largest degree algorithm, after that it checks the probability of finding the first clique in the first subgraph using the same algorithm, and finally it shows the results.

This class contains three main methods; ColorNode, ColorSubNOdes, and CCI\_Dev\_Conv . It also makes use of another method from another class, the name of this method is compare and it's written in the DegreeComparator class. This class is originally implemented from the Comparator interface.

Starting our explanation with the compare method, the main function of this method is to order the nodes of a collection basing on the degree of each node. This method will be called implicitly when a new node enters to the collection, the new node will be put in the right entry according to its degree. See Figure 4.10.

```
1 public static class DegreeComparator implements Comparator
2 {
3     public int compare(Object o1, Object o2)
4     {
5         Node v1 = (Node)o1;
6         Node v2 = (Node)o2;
7         if(v1.NumberOfAdjacents <= v2.NumberOfAdjacents)
8         {
9             return 1;
```

```

10 }
11 else if (v1.NumberOfAdjacents > v2.NumberOfAdjacents)
12 {
13 return -1;
14 }
15 else return 0 ;
16 }
17 }

```

**Figure 4.10 Compare method**

The main function of the ColorNode method is to choose the minimum color for a particular node. If the selected color exceeds the current maximum color then it will be saved in a global variable as a new maximum color. See Figure 4.11.

```

1 public static void ColorNode (Node node , Graph graph , LinkedHashSet
2 ColorsOrder){
3 for(int x = 0 ; ; x++ )
4 {
5 if(node.isValidColor(graph, x))
6 {
7 node.ColorNode(x);
8 ColorsOrder.add(node.Value );
9 if(x > MaxColor)
10 {
11 MaxColor = x;
12 }
13 break;
14 }
15 }
16 }

```

**Figure 4.11 ColorNode method**

After coloring the node which has the maximum degree, the program must color the adjacent nodes of this node. This is the main function of the ColorSubNOdes method. It orders the adjacent nodes in decreasing order basing on the degree of each adjacent using the compare method, and then it starts the coloring in the same previous manner.

CCI\_Dev\_Conv is the last method in this class. This method is responsible for calculating the Clique Conformance Index, Deviation Rate and Convergence rate. This method has three parameters; array CliqueArr which contains clique elements , ColorsOrderArr which saves the order of coloring and CCIArr [] which will store the CCI value for each node in the graph. See figure 4.12.

The if statement in Line 7 compares value of ColorOrderArr[j] with value of Cliquearr[i] if they are identical and J is less than clique length then the deviation rate and the distance for that node in index I will be assign to zero. Which means that if the LDC or LDSC algorithms colors the clique nodes in order before color any node out of the clique then the deviation rate and the distance for those nodes will be zeros. Otherwise the above values will be calculated according to the previously explained equations.

```

1 public static CCI [ ] CCI_Dev_Conv ( CCI CCIArr [ ], int CliqueArr
2 , int ColorsOrderArr [ ] )
3 int Nodes_Out_Of_Order = 0 ;
4 float Total_Deviation = 0 ;
5     for (int i = 0 ; i <CliqueArr.length ; i++){
6         for (int j = 0 ; j < ColorsOrderArr.length ; j++){
7             if (ColorsOrderArr [j] == CliqueArr[i]){
8                 if(j<CliqueArr.length){
9                     if(CCIArr[i] != null){
10                        CCIArr[i].value = CliqueArr[i];
11                        CCIArr[i].distance = 0 ;
12                        CCIArr[i].deviation = 0;} break ;}
13             Else{
14                 if(CCIArr[i] != null){
15                     CCIArr[i].value = CliqueArr[i];
16                     CCIArr[i].distance = j-CliqueArr.length + 1 ;
17                     CCIArr[i].deviation = CCIArr[i].distance/(j + 1);
18                     Nodes_Out_Of_Order ++;
19                     Total_Deviation = Total_Deviation + CCIArr[i].deviation ;

```

```

20 }}}}
21     double Deviation_Rate = Total_Deviation / Nodes_Out_Of_Order ;
22     return CCIArr ;}

```

Figure 4.12 CCI\_Dev\_Conv Method

## 4.6 MODEFIEDLARGESTDEGREE CLASS

This class represents the new modified algorithm that resulted from this research; the implementation of this algorithm is similar to the previous algorithm except some differences. The main difference is in the compare method, nodes are ordered according to their degrees but if two or more nodes have the same degree, then the program must compare these nodes according to its saturation degree. See Figure 4.13.

```

1  public static class DegreeAndDsaturComparator implements Comparator{
2  public int compare(Object o1, Object o2){
3  Node v1 = (Node)o1;
4  Node v2 = (Node)o2;
5  if (v1.NumberOfAdjacents == v2.NumberOfAdjacents){
6  if(v1.Dsatur <= v2.Dsatur){ return 1;}
7  else if (v1.Dsatur > v2.Dsatur){ return -1;}
8  else return 0 ;
9  }
10 else if(v1.NumberOfAdjacents < v2.NumberOfAdjacents){
11 return 1;}
12 else if (v1.NumberOfAdjacents > v2.NumberOfAdjacents){
13 return -1;}
14 else return 0;
15 }
16 }

```

Figure 4.13 Compare method

Another important difference is the need for calculating the saturation degree for the adjacent node of the colored node whenever a node is colored; this required a new arrangement for the nodes in the main list whenever a node is colored.

## CHAPTER FIVE

### COMPLEXITY AND PERFORMANCE ANALYSIS

---

This chapter presents analysis for the complexity of the largest degree algorithm (LDC) and for the modified largest degree algorithm (LDSC), and it will also compare the performance of both algorithms using several experiments. We will also generate results for an exhaustive algorithm which detects the maximum clique in a randomly generated graph.

## 5.1 COMPLEXITY ANALYSIS

In the context of time complexity, LDC algorithm is considered faster than the LDSC algorithm. LDC algorithm can be run in  $O(n^2)$  in its worst case, while LDSC needs  $O(n^3)$  in the worst case.

### 5.1.1 COMPLEXITY OF THE LDC ALGORITHM

1. Assume the largest degree  $d = d_1$ ; and that node  $v_1$  has degree  $K_1$

1.1. The first step assigns the smallest color, say  $c_1$  to node  $v_1$ . The total number of steps required to color all the nodes in the neighbor list of  $v_1$  is

$$1+2+3+ \dots + d_1 = (d_1^2 + d_1)/2 = O(d_1^2)$$

1.2. Repeat the coloring procedure for the next node  $v_2$  with degree  $d_2$ . The number of steps required to color all the nodes adjacent to node  $v_2$  is

$$1+2+3+ \dots + d_2 = (d_2^2 + d_2)/2 = O(d_2^2)$$

2. In general, the number of steps required to color all the nodes in the neighbor list of any node  $v_i$  with degree  $d_i$  is

$$(d_i^2 + d_i)/2 = O(d_i^2)$$

3. Let the average degree of nodes be  $\mu$ . Then the average number of steps required to color the neighbors of node  $v_i$  with degree  $\rho$  is  $O(\mu^2)$



- Repeat the coloring procedure in steps 1 and 2 until all nodes are colored.
- Since each coloring step colors on the average  $\rho$  nodes, the coloring procedure will be repeated on the average  $(n/\rho)$ , where  $n$  is the number of nodes.
- The total number of coloring steps required to color all nodes, on the average is

$$O((n/\mu) \cdot (\mu^2)) = O(n \cdot \mu).$$

The complexity equation (above) can be expressed as

$$\sum_{i=1}^n \mu, \text{ where } \mu = \left( \sum_{i=1}^n d_i \right) / n.$$

### 5.1.2 COMPLEXITY OF LDSC ALGORITHM

The major difference between LDSC and LDC is that whenever a new node is colored the algorithm will compute the saturation degree for all the neighbors of this node. Practically this new change requires additional inner loop to recalculate the saturation degree for every node. In term of time complexity the new change will require additional  $(n)$  steps for each node. In other words, the time complexity for the new algorithm is  $O(n^3)$  in the worst case.

## 5.2 PERFORMANCE ANALYSIS

The algorithms presented in this thesis (LDC and LDSC) have been tested on a set of benchmarks which are based on randomly generated graphs. Three categories of graphs are used in the benchmarks: heavy, medium, and low density graphs. The benchmarks include different graph sizes with the number of nodes ranging from 25 nodes (small graphs) to 1000 nodes (large graph) as shown in Table 5.1, 5.2 and 5.3.

**Table 5.1 First Experiment**

LDSC				LDC				Density	Size of the clique	Number of Nodes
CCI = $\rho/\Delta$	Average Deviation	Convergence Rate $\rho$	Number of Colors	CCI = $\rho/\Delta$	Average Deviation	Convergence Rate $\rho$	Number of Colors			
4	0	1.00	5	$\infty$	0	1.00	5	L	4	25
1.24	.64	.8	7	1.29	.61	0.80	8	R	5	25
9	0.00	1.00	11	9.00	0.307	0.89	11	H	9	25
1	.5	.5	7	.87	.57	.5	8	L	4	50
2.08	.40	.83	11	1.83	.45	.83	11	R	6	50
1.8	0.30	0.54	18	1.45	0.37	0.54	19	H	11	50
1.24	.6	0.75	11	1.24	.6	0.75	12	L	4	100
.83	.51	.42	20	.77	.55	.42	21	R	7	100

1.1	0.76	.85	28	1.37	.57	.78	30	H	14	100
.66	.6	0.40	20	.66	.6	0.40	21	L	5	200
.71	.7	.5	34	.68	.73	.5	36	R	8	200
.98	.67	.66	54	1.05	.56	.6	54	H	15	200
.67	.59	.4	37	.62	.63	.4	40	L	5	500
.67	.82	.55	72	.66	.83	.55	73	R	9	500
.579	.63	.36	118	.571	.64	.36	124	H	19	500
.17	.82	0.14	64	.17	.82	.14	67	L	7	1000
.387	.774	.3	126	.385	.777	.3	129	R	10	1000
.58	.62	.36	212	.58	.62	.36	220	H	22	1000

**Table 5.2 Second Experiment**

LDSC				LDC				Density	Size of the clique	Number of Nodes
CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Number of Colors	CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Number of Colors			
4	0	1	5.00	2.24	.33	.75	5.00	L	4.00	25.00
5	0	1	7.00	4.79	.16	.8	7.00	R	5.00	25.00
10	0	1	10.00	.88	.1	.88	11.00	H	10.00	25.00
2.24	.33	.75	7.00	2.24	.33	.75	7.00	L	4.00	50.00
.800	.749	.6	10.00	.781	.767	.6	11.00	R	5.00	50.00
1.53	.47	.72	18.00	1.61	.39	.63	19.00	H	11.00	50.00
1.02	.78	0.80	11.00	1.03	.77	.8	11.00	L	5.00	100.00
.732	.585	.42	21.00	.737	.58	.42	22.00	R	7.00	100.00

1.65	.43	.714	31.00	1.45	.49	.714	33.00	H	14.00	100.00
1	.75	.75	19.00	.96	.777	0.75	20.00	L	4.00	200.00
.96	.77	.75	35.00	.93	.806	.75	36.00	R	8.00	200.00
1.10	.58	.64	56.00	1.07	.6	.64	57.00	H	17.00	200.00
.48	.82	0.40	37.00	.48	.82	0.40	38.00	L	5.00	500.00
.68	.81	.55	70.00	.67	.82	.55	73.00	R	9.00	500.00
.73	.64	.47	117.00	.72	.65	.47	120.00	H	21.00	500.00
.48	.7	0.40	65.00	.58	.67	0.40	65.00	L	5.00	1000.00
.45	.80	.36	126.00	.45	.79	.36	130.00	R	11.00	1000.00
.52	.76	.4	214.00	.49	.70	.35	216.00	H	20.00	1000.00

**Table 5.3 Third Experiment**

LDSC				LDC				Density	Size of the clique	Number of Nodes
CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Number of Colors	CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Number of Colors			
4	0.00	1.00	5.00	$\infty$	0.00	1.00	5.00	L	4.00	25.00
5	0.00	1.00	6.00	$\infty$	0	1	6	R	5.00	25.00
9	0.00	1.00	11.00	$\infty$	0	1	11	H	9.00	25.00
4	0.00	1	7	$\infty$	0.00	1	7	L	4.00	50.00
5	0	1	13	$\infty$	0.00	1	13	R	5.00	50.00
1.8	.46	0.84	19	1.52	.55	0.84	20	H	13.00	50.00
.78	.63	0.50	11	.78	.63	0.50	13	L	4.00	100.00
1.13	.54	.62	20	1.08	.57	0.62	21	R	7.00	100.00
1.37	.51	.71	33	1.35	.47	.64	34	H	14.00	100.00
.75	.79	.6	18	.73	.81	.6	20	L	5.00	200.00
.78	.73	.57	34	.73	.77	.57	36	R	8.00	200.00
1.1	.58	.64	56	1.07	.60	.64	57	H	17.00	200.00
.48	.82	.4	37	.48	.82	.40	38	L	5.00	500.00
.29	.74	.22	70	.29	.75	.22	74	R	9.00	500.00
.47	.63	.30	120	.511	.68	.35	126	H	20.00	500.00
.56	.70	.40	65	.58	.67	.40	65	L	5.00	1000.00
.29	.60	.18	126	.29	.6	.18	128	R	11.00	1000.00
.61	.7	.43	215	.61	.7	.43	219	H	20.00	1000.00

Each experiment contains several runs for graphs with different densities and sizes. For example, the first three rows correspond to graphs with 25 nodes and low density (L), regular density (R) and high density (H). Low density graphs correspond to graphs where the probability of two nodes being connected is 0.25. The probabilities for regular and high density graphs are 0.5 and 0.75 respectively.

Each row of the Table shows the following parameters:

1. The number of nodes: This is given as an input to the program.
2. Size of the largest clique: Generated by the program.
3. Density: (L, R, H): given as input to the program in the form of a probability.
4. Number of colors: The number of colors used to color the graph; generated by the program.
5. Convergence rate ( $\rho$ ): Computed by the program =  $M/N$ ; where  $M$  is the number of nodes colored while in the clique; and  $N$  is the size of the clique.
6. Average deviation ( $\Delta$ ): Computed by the program;  $\Delta = \sum_{i=1}^N \delta_i / N$ , where  $\delta$  is the deviation of each node, measured as  $\delta = [R(\gamma_i) - N] / R(\gamma_i)$ , where  $N$  is the size of the clique and  $R(\gamma_i)$  is the order of coloring node  $R(\gamma_i)$ .
7. Clique conformance index  $CCI = \rho/\Delta$ ; generated by the program.

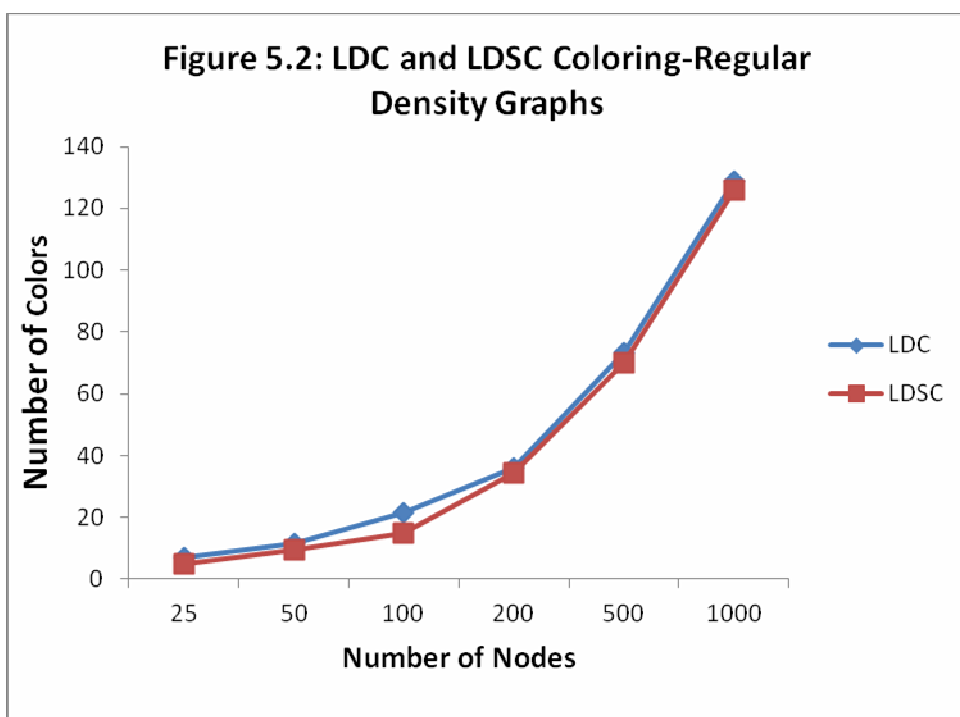
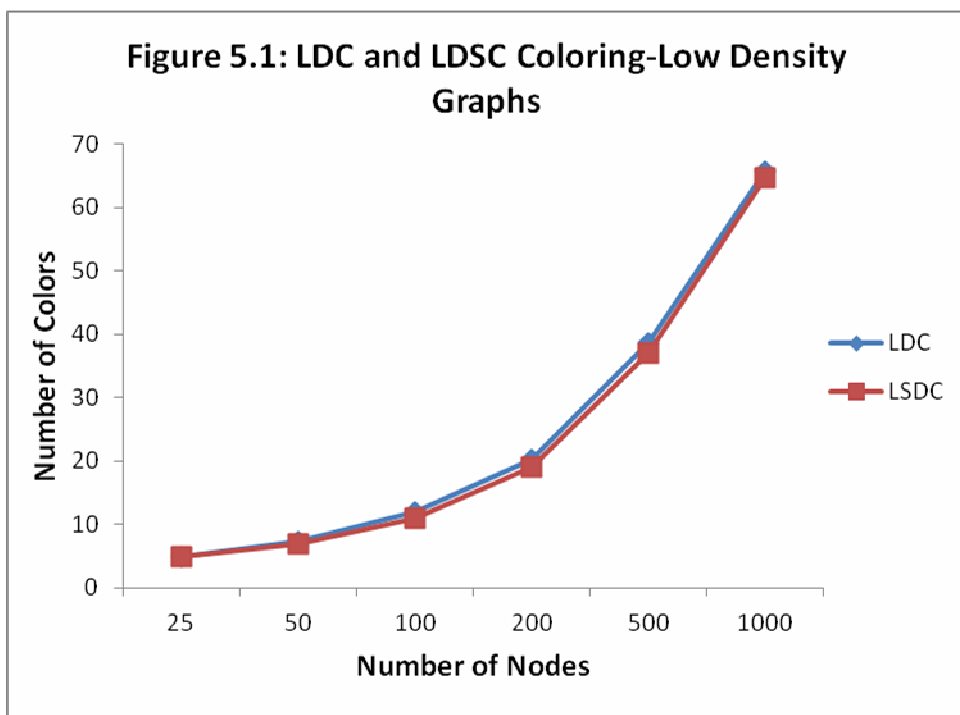
These parameters are shown for the LDC and LDSC algorithms.

Table 5.4 shows the results averaged for all three experiments. The results converge for all three experiments. Since the graphs are generated randomly, we decided to take the average results for several runs. We ran several experiments and the results remain very close for all runs.

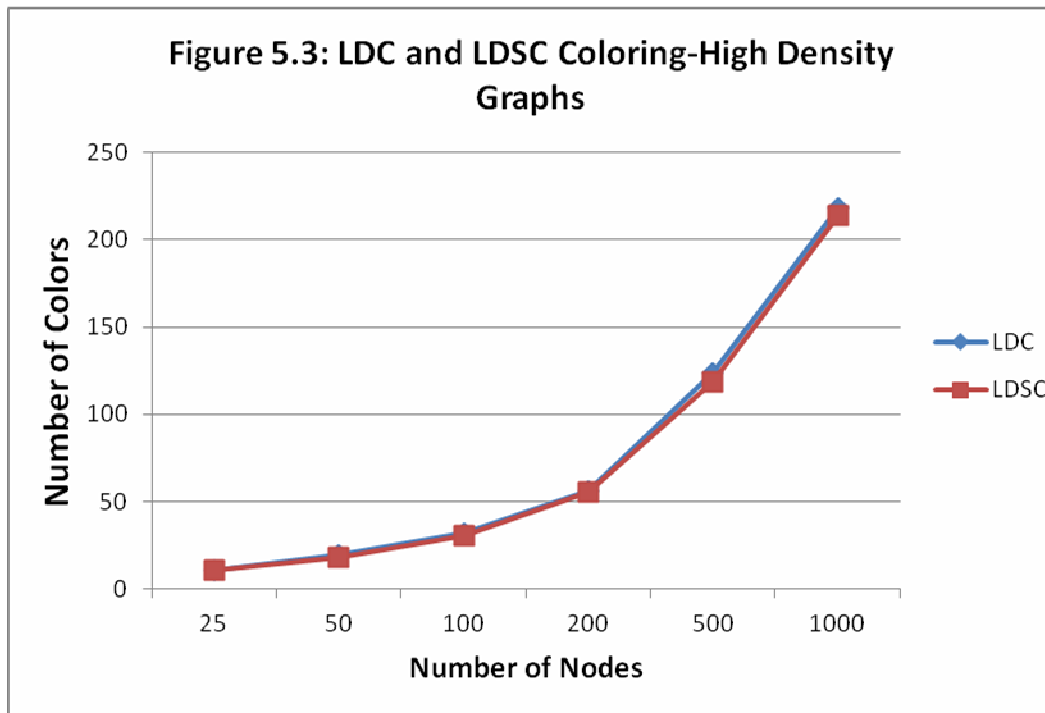
**Table 5.4: Average results for three experiments**

New Largest Degree				Largest Degree				Low Density	
CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Coloring	CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Coloring	Size of the clique	Number of Nodes
4.00	0.00	1.00	5.00	3.41	0.11	0.92	5	4	25
2.41	0.28	0.75	7.00	2.37	0.30	0.75	7	4	50
1.01	0.67	0.68	11.00	1.02	0.67	0.68	12	4	100
0.80	0.71	0.58	19.00	0.78	0.73	0.58	20	5	200
0.54	0.74	0.40	37.00	0.53	0.76	0.59	39	5	500
0.40	0.74	0.31	64.67	0.44	0.72	0.31	66	6	1000
New Largest Degree				Largest Degree				Regular Density	
CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Coloring	CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Coloring	Size of the clique	Number of Nodes
3.75	0.21	0.93	4.79	3.69	0.26	0.87	7.00	5	25
2.63	0.38	0.81	9.70	2.54	0.41	0.81	11.67	5	50
0.90	0.55	0.49	15.10	0.86	0.57	0.49	21.33	7	100
0.82	0.73	0.61	34.33	0.78	0.77	0.61	36.00	8	200
0.55	0.79	0.44	39.42	0.51	0.80	0.44	73.33	9	500
0.41	0.72	0.28	126.00	0.38	0.72	0.28	129.00	11	1000
LDSC				LDC				High Density	
CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Coloring	CCI = $\rho/\Delta$	Average Deviation $\Delta$	Convergence Rate $\rho$	Coloring	Size of the clique	Number of Nodes
9.33	0.00	1.00	11	6.29	0.14	0.92	11	9	25
1.7	0.41	0.70	18	1.53	0.44	0.67	19	12	50
1.33	0.57	0.76	31	1.39	0.51	0.71	32	14	100
1.06	0.61	0.65	55	1.06	0.59	0.70	56	16	200
0.59	0.61	0.38	118	0.72	0.66	0.39	123	20	500
0.57	0.69	0.40	214	0.56	0.67	0.38	218	21	1000

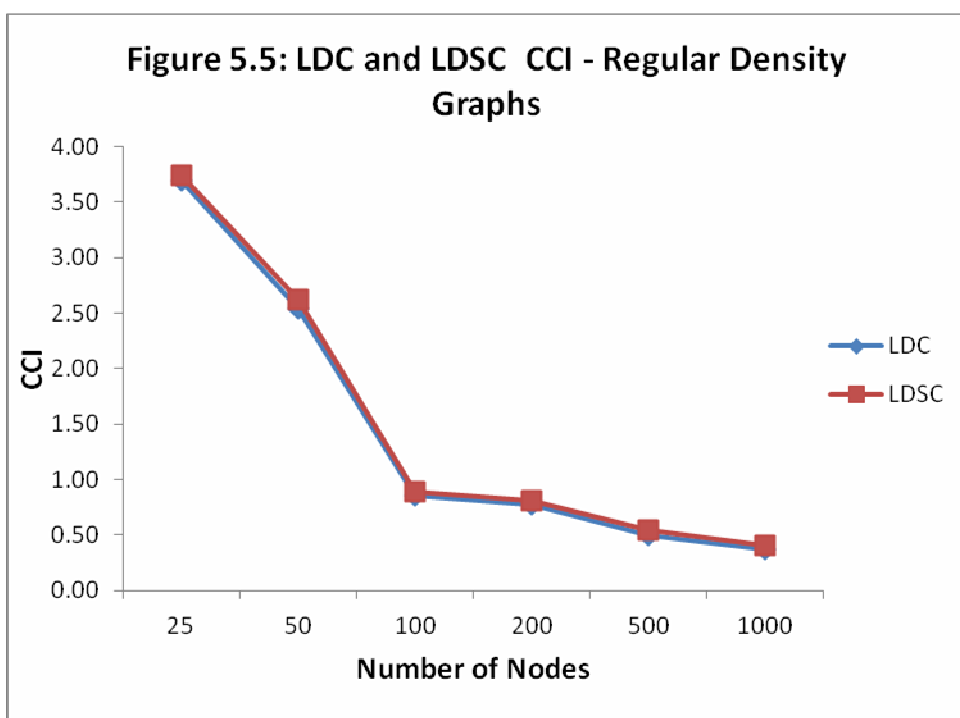
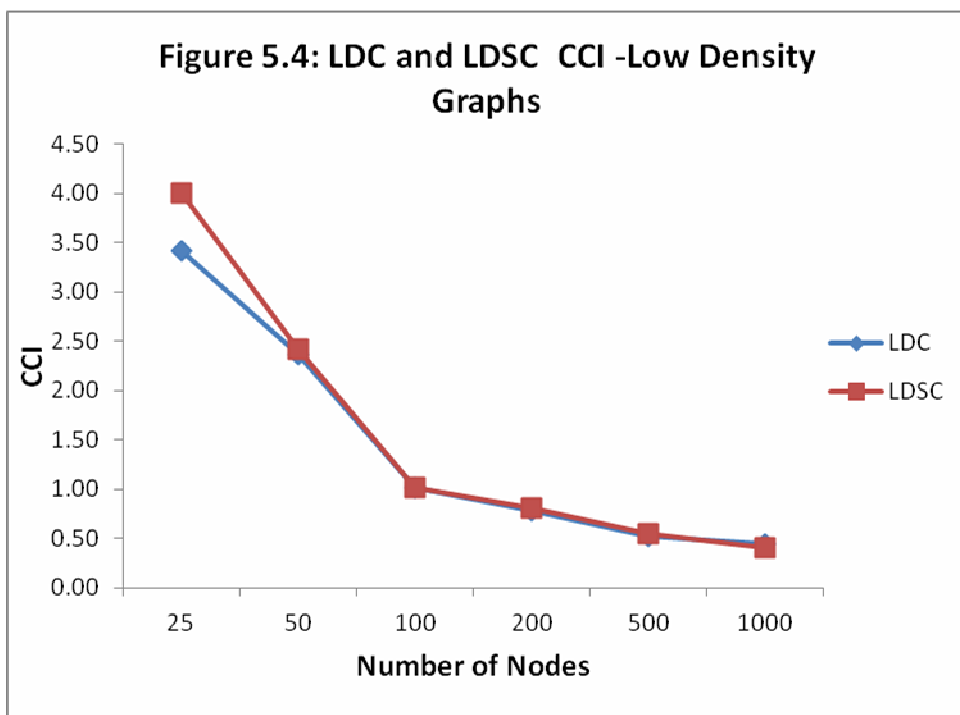
Figures 5.1, 5.2 and 5.3 show the coloring provided by LDC and LDSC algorithms for low, regular and high density graphs. LDSC produces slightly lower number of colors than LDC.







Figures 5.4, 5.5 and 5.6 show the clique conformance index for both LDC and LDSC algorithms. Both algorithms observe similar behavior. For lower number of nodes, both algorithms conform more closely to the maximum clique in the graphs. The larger the graph, the more the algorithms deviate from the maximum clique.



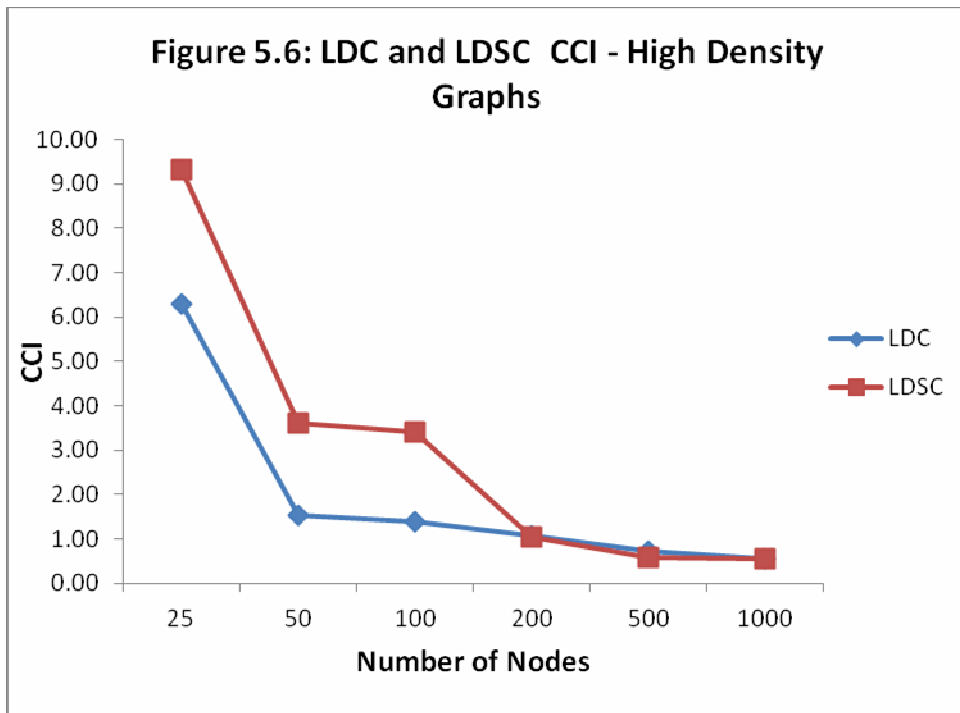
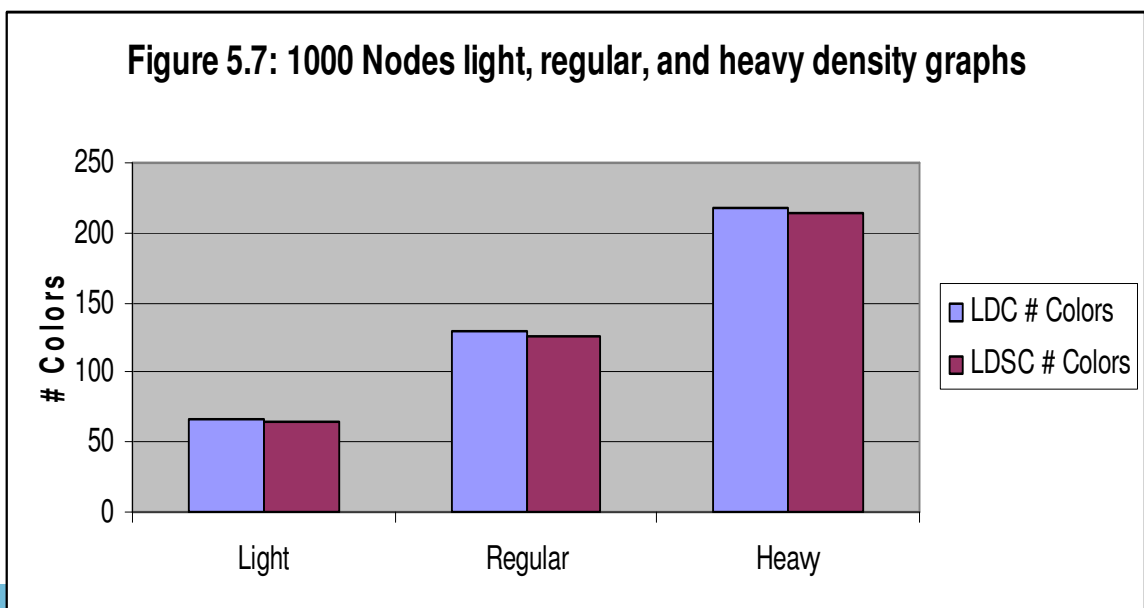


Figure 5.7 shows the performance of both algorithms for large number of nodes, high density graphs. The density of the graph does not have a significant impact on the performance of the algorithms.



We finally compare the coloring algorithms (LDC and LDSC) with backtracking exhaustive search coloring algorithm.

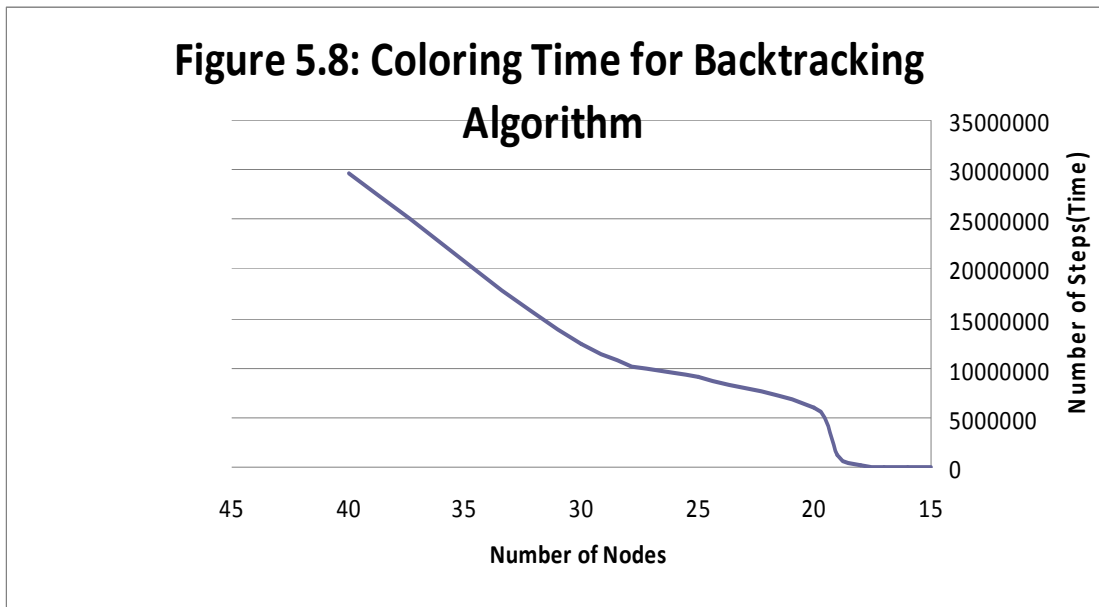
Table 5.5 shows the results for small size graphs (number of nodes less than 50). We could not run the backtracking algorithm for larger graphs, because the execution time is prohibitive. For 40 nodes, the program ran for more than 3 hours and the time increase is exponential with the increase of the number of nodes. For the small size graphs we ran, the performance of the LDC and LDSC in terms of the number of colors is comparable. This indicates that the conformance to the maximum clique leads to lower number of used colors.

**Table 5.5: Comparing LDC and LDSC with Backtracking Algorithm**

Back Tracking (Optimal)			LDSC		LDC		Density	Number of Nodes
Number of Steps	Time	Number of Colors	Time in Seconds	Number of Colors	Time in Seconds	Number of Colors		
1238	2 Seconds	4	Less than 1	4	Less than 1	4	L	15
8874	4 Seconds	4	Less than 1	4	Less than 1	4	L	16
47947	26 Seconds	3	Less than 1	3	Less than 1	3	L	17
150704	1.14 minutes	3	Less than 1	3	Less than 1	3	L	18
1,182,112	11.38 minutes	4	Less than 1	4	Less than 1	4	L	19
5931701	40 minutes	4	Less than 1	4	Less than 1	4	L	20
9,076,729	1 Hour	4	Less than 1	4	Less than 1	4	L	25
12,452,861	1.3 Hour	4	Less than 1	4	Less than 1	4	L	30
29,712,003	3 Hours	5	Less than 1	5	Less than 1	5	L	40

Figures 5.1 – 5.6 show that both LDC and LDSC conform closely to the maximum clique for small graphs. For 50 node graphs, the CCI performance index is 3, 4, and 6 for low, regular and heavy density graphs respectively. Hence, the algorithms managed to color the graphs with the same number of colors as the backtracking exhaustive algorithm. Albeit, the time used to produce the colors is significantly lower than the

time required by the backtracking algorithm, as shown in Figure 5.8. The time for LDC and LDSC is not shown since it took few seconds for the algorithms to finish coloring the graph.



## CHAPTER SIX

### CONCLUSIONS AND FUTURE WORKS

---

In this chapter, we present and discuss the conclusions of our work, and ideas for future work.

#### 6.1 SUMMERY

First of all the researcher has introduced a brief review for some important concepts related to the subject of the thesis such as, graphs, graph coloring, clique, P, NP, and NP-Complete Problems. After that he has clarified the problems that have been addressed by the thesis in chapter 1. In chapter 2 the researcher has presented a brief description of what have been done by other scientists in this area of research. We have introduced some methods addressing the graph coloring problem such that the backtracking method, the local search method and the greedy method. In chapter 3 he has explained the modified coloring algorithm. At the beginning of the chapter he clarified the term "Degree Saturation", and then he explained the modified coloring algorithm. After that he explained the clique detection process using the modified coloring algorithm. The implementation of this algorithm is illustrated in chapter 4, in this chapter the researcher discussed the random generation of the graphs, together with their possible properties, types, and the data structure used in the generation, he also went through the main java classes used in our implementation. The complexity analysis, the performance analysis and the experiments results, are discussed in chapter 5.

## 6.2 CONCLUSIONS

The graph coloring problem and the maximum clique problem are essential problems in graph theory, and they arise in many real life applications. These two problems have received a lot of attention by the scientists, not only for their importance in the real life applications, but also for their theoretical sides.

Unfortunately solving these problems is very complex, and the proposed algorithms for this purpose are able to solve only the small graphs with up to 80 nodes, on the other hand and in order to module the real life application we need to graphs of hundreds or thousands of nodes.

This thesis presents a new algorithm for solving these hard problems. The new algorithm runs in a considerable time and it shows a competitive results for both of special purpose graphs as well as the general random graphs. It also introduces a new measure for the deviation of the algorithms from maximum clique based coloring. We summarize the main achievement and contributions of this thesis by the following:

1. A modified algorithm (LDSC) of the maximum degree algorithm (LDC) is introduced, where the LDSC used the idea of saturation degree to better determine the next node to be colored when two or more nodes have the same degree. It also offers a detection of the maximum clique in the underlying graphs.
2. New performance measure is given, and used to measure the performance of both LDC and LDSC. This measure use the ideas of convergence rate  $\rho$ , deviation rate  $\delta$ , average deviation  $\Delta$ , and the clique conformance index (CCI).

3. A performance checking mechanism is introduced and used in our experimental work.

### **6.3 FUTURE WORK**

We would like to suggest some interesting issues and ideas that could not be satisfied because of time limitation and they will help as improvement of this work.

- Work on new algorithms to produce better CCI.
- Study other algorithms to measure and compare the CCI.



## REFERENCES

1. Akkoyunlu, E. V.-6. (1973). The enumeration of maximal cliques of large graphs. *SIAM J. Compute* , 1-6.
2. Allen, M., Kumaran, G., & Liu, T. (2002). A combined algorithm for graph-coloring in register allocation. *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, (pp. 100–111).
3. Appel, K., & Haken, W. (1977). Every planar map is four colorable. *Part I. Discharging, Illinois J. Math. 21* , (pp. 429-490).
4. Appel, K., & Haken, W. (1977). Every planar map is four colorable. *Part II. Reducibility, Illinois J. Math. 21.* , (pp.491--567).
5. Babel, L., & Tinhofer, G. (1990). A branch and bound algorithm for the maximal clique problem. *ZOR-Methods and models of operation research* , 207-217.
6. Balas, E., & Yu, C. S. (1986). Finding a maximum clique in an arbitrary graph. *SIAM J. Comput* , Vol 14: 1054-1068.
7. Balasundaram, B. e. (2009). Clique Relaxations in Social Network Analysis: TheMaximum k-plex Problem.
8. Bednarek, A., & Taulbee, O. (1966). On maximal chains Roum. *Math. Pres et Appl* , Vol 11: 23-25.
9. Biggs, N. L., Lloyd, E. K., & Wilson, R. J. (1986). *Graph Theory*. Oxford.
10. Bonner, R. (1964). On some clustering technique. *IBM J. Res. Develop* , Vol. 8: 22-32.
11. Brelaz, D. (1979). New methods to color the vertices of a graph. *ACM* .

12. Bron, K., & Kerbosh, J. (1973). Finding all cliques of an undirected graph. *Commun. ACM* , Vol. 16: 575-577.
13. Caramia, M., & Dell’Olmo, P. (2001). Iterative Coloring Extension of a Maximum Clique. *Naval Research Logistics (NRL)* .
14. Coleman, T. e. (1983). Estimation of sparse jacobian matrices and graph coloring problems, . *SIAM Journal on Numerical Analysis* .
15. De Werra, D. (n.d.). An introduction to timetabling . *European Journal of Operational Research* , 151–162.
16. Desle, J. F., & Hakimi, S. L. (1970). On maximum internally stable set of a graph. *fourth annual Princeton conference on information science and systems*, (pp. Vol. 4: 459-462).
17. Dorian, P. (1969). A note on the detection of cliques in valued graphs. *Sociometry* , Vol 32: 237-244.
18. Downey, R. G., & Fellows, M. R. (1995). Fixed-parameter tractability and completeness,. *Theoretical Computer Science* , 141 (1–2).
19. Gamst, A. (1986). Some lower bounds for a class of frequency assignment problems. . *IEEE Transactions of Vehicular Technology* , 8–14.
20. Gebremedhin, A. (1999). Parallel Graph Coloring. *UNIVERSITY M Thesis University of Bergen Norway Spring*.
21. Harary, F., & Ross, I. C. (1957). A procedure for clique detection using the group matrix. Vol. 20: 205-215.
22. Jensen, T. R., & Toft, B. (1994). *Graph Coloring Problems*. USA: John Wiley & Sons.

23. Johnson, Aragon, McGeoch, & Schevon. (1991). Optimization by Simulated Annealing: An Experimental Evaluation; Part {II}, Graph Coloring and Number Partitioning". *Operations Research* , vol. 39, 378-406.
24. Johnson, D. S., Mehrotra, A., & Trick, M. (Ithaca, New York, USA). Proceedings of the Computational Symposium on Graph Coloring and its Generalizations. 2002.
25. Kempe, B. A. (1879). On the geographical problem of the four colors. *Amer. J. Math.*, 2 , 193-200.
26. Kopf, R., & Ruhe, G. (1987). A computational study of the weighted independent set problem for general graphs. Vol. 12:167-180.
27. Loukakis, E., & Tsouros, C. (1981). A depth first search algorithm to generate the family of maximal independent sets of graph lexicographically. Vol. 27:249-266.
28. Luce, R., & Perry, D. (1949). A method of matrix analysis of group structure. 14(2): 95-116.
29. Maghout, K. (1959). Sur la determination des nombres de stabilite et du nombre chromatique d'un graph,. Vol. 248: 2522-2523.
30. Magnus, M. H. (1991). Frugal methods for the independent set and graph coloring problems. *PhD thesis, The State University of New Jersey, New Brunswick, New Jersey.*
31. Malkawi, M. e. (Mar 23-26, 2009). A graph-Coloring –Based Navigational Algorithm for Personnel Safety in Nuclear Applications. *Proceeding of the 6th International Symposium on Mechatronics and its Applications (ISMA09)*, . Sharjah, UAE,.

32. Malkawi, M. e. (1990). Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm. *Journal of Parallel & Distributed Systems* , vol. 10.
33. Malkawi, M. e. (1992). Distributed Algorithms for Edge Coloring of Graphs. *Conference on Parallel and Distributed Computing Systems*.
34. Malkawi, M. e. (1987). Distributed Algorithms for Load Balancing in Very Large Homogeneous Systems. *Fall Joint Computer Conference*.
35. Malkawi, M. e. (1990). Graph Coloring Based Distributed Load Balancing Algorithm and Its Performance Evaluation. *4th Annual Symposium on Parallel Processing* .
36. Malkawi, M. e. (1989). System Theory Modeling and Performance Analysis of a Distributed Load Balancing Algorithm. *32nd Midwest Symp. on Circuits and Systems* .
37. Malkawi, M., & Haj Hasan, M. (2010). Coloring of Triangle Free Girth Graphs with Minimal k-Colors. *International Journal on Computers and Network Security* , Vol. 2, No. 5.
38. Malkawi, M., Hajhasan, M., & Hajhasan, O. (2008). New exam scheduling algorithm using graph coloring. *International Arab Journal for IT , (IAJIT)*.
39. Marcus, P. M. (1964). Derivation of maximal compatibles using Boolean algebra . *IBM J. Res. Develop* , , Vol. 8: 537-538.
40. Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992.). A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* , 161–205.
41. Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* , 58(1–3):161–205, .

42. Paull, M. C., & Unger, S. H. (1959). Minimizing the number of states in incompletely specified sequential switching function . Vol. EC-8: 356-367.
43. Robson, J. M. (1989). Algorithm for maximum independent set . *J.Algorithms* , Vol. 7: 425-440.
44. Robson, J. M. (2001). Finding a maximum independent set in time  $O(2^{n/4})$ .
45. Tarjan, R. E. (1972). Finding the maximum clique. *Technical Report* , 72-123.
46. Tomita, A., Tanaka, A., & Takahashi, H. (1988). *The worst-case time complexity for finding all the cliques*. UEC-TR-C5.