**Review on Graph Coloring Algorithms**

A project report submitted under the partial fulfillment of the requirements for the award of degree

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE AND ENGINEERING**

By

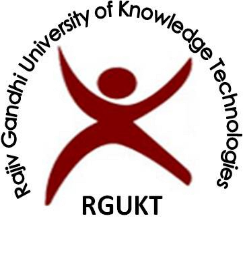
**Sravani Murakonda**

**(N091869)**

Under the Guidance of

**Mr. Krishna Kumar Singh**

Lecturer in Department of Computer Science and Engineering



**April 2015**

**RGUKT-NUZVID**

Nuzvid, Krishna, Andhra Pradesh – 521202

|  |  |
| --- | --- |
|  | *RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES*  (A.P. Government Act 18 of 2008)  RGUKT-NUZIVID  Nuzivid, Krishna, Andhra Pradesh – 521202.  Ph: 08656 – 235147; Telefax: 08656 – 235150 |
| ***Mr. Krishna Kumar Singh***  ***Department of Computer Science & Engineering***  **CERTIFICATE**  This is to certify that the project entitled “**Review on Graph Coloring Algorithms**” is a record of bonafide work carried out by **Sravani Murakonda (N091869)** under my guidance and supervision for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering during the academic session August 2014 – April 2015 at RGUKT Nuzivid. To the best of my knowledge, the results embodied in this dissertation work have not been submitted to any university or institute for the award of any degree or diploma.  ***Mr. Krishna Kumar Singh***  ***Department of Computer Science & Engineering*** | |
| |  |  | | --- | --- | |  | *RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES*  (A.P. Government Act 18 of 2008)  RGUKT-NUZIVID  Nuzivid, Krishna, Andhra Pradesh – 521202.  Ph: 08656 – 235147; Telefax: 08656 – 235150 | | **CERTIFICATE OF EXAMINATION**  This is to certify that the project entitled “**Review on Graph Coloring Algorithms**” is a record of bonafide work carried out by **Sravani Murakonda(N091869)** under my guidance and supervision for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering during the academic session August 2014 – April 2015 at RGUKT Nuzivid. To the best of my knowledge, the results embodied in this dissertation work have not been submitted to any university or institute for the award of any degree or diploma.  EXAMINER:  1.  2.  3.  Date: | | | |

|  |  |
| --- | --- |
|  | *RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES*  (A.P. Government Act 18 of 2008)  RGUKT-NUZIVID  Nuzivid, Krishna, Andhra Pradesh – 521202.  Ph: 08656 – 235147; Telefax: 08656 – 235150 |
| **CERTIFICATE OF PROJECT COMPLETION**  This is to certify that the project entitled “**Review on Graph Coloring Algorithms**” is a record of bonafide work carried out by **Sravani Murakonda (N091869)** under my guidance and supervision for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering during the academic session August 2014 – April 2015 at RGUKT Nuzivid. To the best of my knowledge, the results embodied in this dissertation work have not been submitted to any university or institute for the award of any degree or diploma.  Head of Department:  **Ms. D.V. Nagarjuna Devi M.Tech**  Lecturer in Computer Science &  Engineering  RGUKT - NUZVID  Project Guide:  Mr. Krishna Kumar Singh  Lecturer in Computer Science &  Engineering  RGUKT – NUZVID | |

#### DECLARATION

**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES**

(

A.P. Government Act 18 of

2008)

**RGUKT**

**-**

**NUZ**

**I**

**VID**

Nuzvid,

Krishna, Andhra Pradesh

–

521202.

Ph: 08656

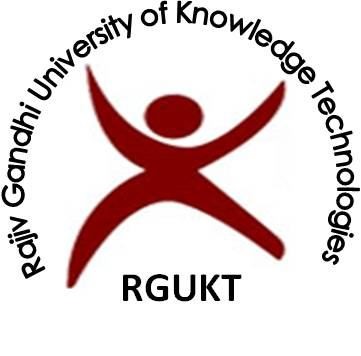
–

235147

; Telefax: 08656

–

235150



I,**Sravani Murakonda (N091869)** **,** hereby declare that the project report entitled “**Review on Graph Coloring Algorithms**” done by me under the guidance of **Mr. KRISHNA KUMAR SINGH** is submitted for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering during the academic session August 2014 – May 2015 at RGUKT- Nuzvid.

I also declare that this project is a result of my own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references.

The results embodied in this project report have not been submitted to any other university of institute for the award of any degree or diploma.

**Sravani Murakonda (N091869)**

**Place:** Nuzvid

**Date: 17/04/2015**

***Acknowledgements***

I am deeply indebted to my project guide, **Mr. Krishna Kumar Singh** for his valuable guidance, motivation and constant encouragement throughout the course of this thesis work. His help, advice and guidance have been a constant source of inspiration to me throughout the B. Tech course. I shall always cherish the time spent with him during the course of this work due to the invaluable knowledge gained in the field of reliability engineering.

I am very grateful to **Prof. RAJ REDDY (Chancellor, RGUKT), Prof. K. SATYANARAYANA (Vice Chancellor, RGUKT), and Prof. Hanumantha Rao (Director, RGUKT - Nuzvid)** for their endeavor and extended support throughout my education in the institute.

I express gratitude to **Mrs. D.V.NAGARJUNA DEVI (HOD of CSE) and other faculty members** for being source of inspiration, and constant encouragement which helped me in completing the project successfully.

My sincere thanks to all the **batch mates of 2009 CSE**, who have made my stay at RGUKT-Nuzvid, a memorable one.

Finally, I express gratitude to our parents for supporting me in every walk of life.

# ABSTRACT

To study and present the already existing graph coloring algorithms and analyzing the heuristics, properties and time complexities of different algorithms like Greedy coloring, Sequential coloring, Largest Degree coloring, Incidence Degree Coloring, Saturation Degree Coloring, Independent Set Coloring and Graph coloring by Domination Vertex Covering(DOM). Comparing the output chromatic number(c(G)) given by the algorithms for the various benchmark graphs. To discuss different applications of graph coloring in real time problems and their implementation.

Contents

[ABSTRACT vii](#_Toc416950317)

[Chapter](#_Toc416950329) 1 [INTRODUCTION 3](#_Toc416950319)

[Chapter](#_Toc416950329) 2 [Reviewed Algorithms 5](#_Toc416950321)

[2.1 Greedy Algorithm 5](#_Toc416950322)

[2 .2 Sequential Coloring 7](#_Toc416950323)

[2 .3 Saturation Degree Ordering 9](#_Toc416950324)

[2 .4 Incidence Degree Ordering 11](#_Toc416950325)

[2 .5 Largest Degree Ordering 13](#_Toc416950326)

[2 .6 Independent Set 15](#_Toc416950327)

[2.7 Domination Vertex Covering 16](#_Toc416950328)

[Chapter 3](#_Toc416950329) [Computational Results 19](#_Toc416950330)

[3.1 DIMAC Graphs 19](#_Toc416950331)

[3.2 Clique Graphs 20](#_Toc416950332)

[Chapter 4](#_Toc416950333) [Comparison of Literature Work 22](#_Toc416950334)

[4.1 Comparison Table 22](#_Toc416950335)

[Chapter 5](#_Toc416950336) [Applications of Graph Coloring 24](#_Toc416950337)

[5.1 Scheduling Exams: 24](#_Toc416950338)

[5.2 Register Allocation 25](#_Toc416950339)

[5.3 Sudoku 25](#_Toc416950340)

[5.4Traffic Signal Design 26](#_Toc416950341)

[5.5 8 Queen Problem: 26](#_Toc416950342)

[Chapter 6](#_Toc416950343) [Conclusion 27](#_Toc416950344)

[REFERENCES 28](#_Toc416950345)

**TABLE OF FIGURES**

[Figure 2.1 5](#_Toc393273608)

[Figure 2.2 8](#_Toc393273609)

[Figure 2.3](#_Toc393273610) 10

[Figure 2.4](#_Toc393273611) 12

[Figure 2.5](#_Toc393273612) 14

[Figure 2.6](#_Toc393273613) 16

[Figure 2.7](#_Toc393273619) 18

[Figure 3.1 24](#_Toc393273613)

**TABLE OF TABLES**

Table 3.1 19-20

Table 3.2 20-21

Table 4.1 22-23

# Chapter 1

# INTRODUCTION

Graph coloring is defined as coloring the nodes of a graph with the minimum number of colors without any two adjacent nodes having the same color. The coloring of a graph G=(V,E) is a function f: V -> C, such that for all {*v,w*}ϵ E: f(*v*) ≠ f(*w*). In other words, adjacent vertices are not assigned the same color[2]. The problem that arises is the coloring of a graph provided that no adjacent vertices have the same color. The Chromatic number of a graph G, **ᵡ(G)** : Minimum size of C such that there is a vertex coloring to C.

Graph coloring is one of the most useful models in graph theory. By using graph coloring many real world problems can be modelled and thus having a fast and robust algorithm to generate optimal coloring is highly desirable. Since the nature of graph coloring is in NP-complete no general purpose technique is known and new approaches are trade-off between accuracy and time complexity. Although, graph coloring is known as a NPhard [8], but it is an important problem, as it has a wide variety of scope in real world applications[7] .

For example, the linked list needs two colors and so does the binary search tree and to color a map it is already proved that the maximum chromatic number is 4. Graph coloring has wide applications such as: estimation of sparse Jacobins, bandwidth allocation, frequency matching, analysis of biological and archaeological data, pattern matching, printed circuit board testing I-coloring, Printed circuit board testing-II clique, registering allocation and many other applications.

Graph coloring is often used for solving scheduling problems of the following type. We are given a set E of jobs (all of them have the same processing time) which have to be performed by some agents with some machines. The constraints to be taken into account are that for each job j in E there is a subset of jobs which cannot be performed at the same time as j because they have to be performed either by the same agent or by the same machine. This type of problem arises in the so-called school scheduling problem where teachers are agents, classes are machines, and lectures are jobs[3].

One of other graph coloring applications has been seen in register allocation, the concept proposed by Magno *etel*. The goal of register allocation is to allocate a finite number of machine registers to an unbounded number of temporary variables such that temporary variables with interfering live ranges are assigned different registers. Most approaches to register allocation have been based on graph coloring. The graph coloring problem can be stated formally as follows: given a graph *G* and a positive integer K, assign a color to each vertex of G, using at most Kcolors, such that no two adjacent vertices receive the same color. We can map a program to a graph in which each vertex represents a temporary variable and edges connect temporaries whose live ranges interfere. We can then use a coloring algorithm to perform register allocation by representing colors with machine registers.

Unfortunately until now we have not found in the literature very good algorithms for coloring the vertices of a graph in a reasonable amount of computation time. Here, we investigated some of the heuristic graph coloring algorithms like Greedy algorithm, Sequential Algorithm, Largest Degree Ordering, Saturation Degree Ordering, Incidence Degree Ordering[4] and Independent Set based Algorithm. Using all these algorithms we have taken the output chromatic number **ᵡ(G)** for the given bench mark Graph G and compared the result for different algorithms.

However, in real life applications variables and incompatibility constraints can evolve with time. Thus, we have to deal with dynamic graphs. Few work has been devoted to deal with the dynamic graph coloring problem [9] [10]. The main drawback of these techniques is that they are efficient for specific dynamic graph instances and they show bad performance with other graphs.

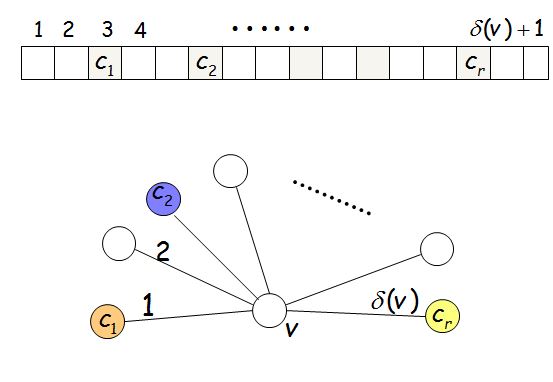
# Chapter 2

# Reviewed Algorithms

In a graph, every vertex is connected to other vertices either directly or indirectly, the directly connected vertices are neighbors of the vertex. There are two types of neighbors, first are visited neighbors and second are not-visited neighbors. Initially we cannot define how many colors will be used to fill the given graph. The algorithm maintains a list of colors, K. It uses a color from the list K and efficiently fills the graph using less number of colors. We illustrate different graph coloring algorithms by taking examples, and conduct experiments to see how the results compare with other graph coloring algorithms for given bench mark graphs.

**Color palette of a node v**

Suppose r neighbors of v has picked colors then color palette gives the number of available colors to color node v followed in Figure 2.1



**Figure: 2.1**

## 2.1 Greedy Algorithm

A greedy algorithm repeatedly executes a procedure which tries to maximize the return based on examining local conditions, with the hope that the outcome will lead to a desired outcome for the global problem. In some cases such a strategy is guaranteed to offer optimal solutions, and in some

other cases it may provide a compromise that produces acceptable approximations. Typically, the greedy algorithms employ simple strategies that are simple to implement and require minimal amount of resources [12].

The basic ideas of greedy algorithm is to begin with a certain initial value and go step by step, measuring by a certain optimization , each step makes sure to obtain local optimal solution value. Each step considers one data by local optimal solution value until the condition satisfies. If the next data and local optimal solution value are connected and their optimal solution values have no feasible solution, these data should neglect.

In general, greedy algorithms have five pillars:

1. A candidate set, from which a solution is created
2. A selection function, which chooses the best candidate to be added to the solution
3. A feasibility function, that is used to determine if a candidate can be used to contribute to a solution.
4. An objective function, which assigns a value to a solution, or a partial solution.
5. A solution function, which will indicate when we have discovered a complete solution

Greedy algorithms produce good solutions on some mathematical problems, but not on others. Most problems for which they work well have two properties:

**Greedy Choice Property:**

We can make whatever choice seems best at the moment and then solve the sub-problems that arise later. The choice made by a greedy algorithm may depend on choices made so far but not on future choices or all the solutions to the sub-problem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage, and may reconsider the previous stage's algorithmic path to solution.

**Optimal Substructure**

"A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the sub-problems."[1] Said differently, a problem has optimal substructure if the best next move always leads to the optimal solution. An example of 'non-optimal substructure' would be a situation where capturing a queen in chess (good next move) will eventually leads to the loss of the game (bad overall move) [11].

**Greedy Coloring (G, n)**

1 **Input:** Graph G(V,E)

2 **Algorithm:** **for** *i* = 1 **to** *n* **do**

3 Select vϵ V(G)

4 Color the vertex with a color min index and different from its neighbor’s color.

5 **Output:** number of colors required to color the graph G(V,E).

There are many heuristic sequential techniques for coloring a graph. One of them is the Greedy Graph Coloring. This technique focuses on carefully picking the next vertex to be colored. In this heuristic algorithm, once a vertex is colored, its color never changes. The following are the different types of heuristics based on greedy algorithm and degree of the vertex (i.e. number of nodes adjacent to the vertex).Each node keeps a color palette of size. . ( is degree of vertex v).

## 2 .2 Sequential Coloring



In Sequential coloring, we use greedy algorithm and we pick the vertex that is to be colored next in one of the vertices permutations of the graph.

**Sequential Coloring (G, n)**

**Input:** Graph G(V,E)

**Algorithm:**

1. Initialize Adj[n][n], Deg[n]
2. for i=1 to n do
3. Update Deg[i]
4. For j=1 to n do
5. Update Adj[i][j] &Adj[j][i]
6. Update InDeg[n] of the graph
7. for i=1 to n do
8. Color the node in the order of vertex number
9. Remove the node from the graph G’[V,E]
10. Update the adjacency matrix Adj[n][n] and degree array Deg[n]
11. return #colors

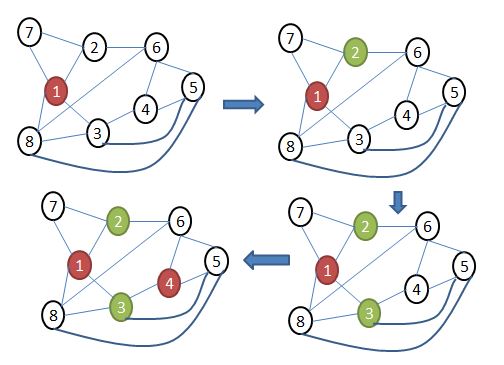
**Output:** Number of colors required to color the graph G(V,E) in O(n^2) time complexity

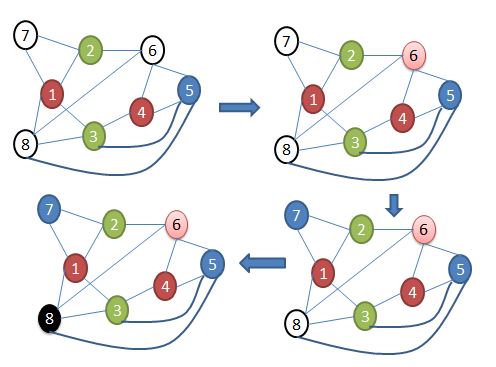
**Example:** Shown in Figure 2.2

**Input:** G(V,E)

**Order of Vertices:** 1, 2,3,4,5,6,7,8

**Output for the example graph:** Chromatic Number is 5.



**Figure:2.2**

## 2 .3 Saturation Degree Ordering

Let G be a simple graph and C a partial coloration of G vertices. We define the saturation degree of a vertex as the number of different colors to which it is adjacent (colored vertices).

**Saturation Degree Coloring (G, n)**

**Input:** Graph G(V,E)

**Algorithm:**

1. Initialize Adj[n][n], Deg[n],SaDeg[n]
2. for i=1 to n do
3. Update Deg[i]
4. For j=1 to n do
5. Update Adj[i][j] &Adj[j][i]
6. Update SaDeg[n] of the graph
7. Sort the nodes based on degree of the vertex in descending order
8. for i=1 to n do
9. Color the node with maximum saturation degree with least available index color and if ambiguity is there then color the node with maximum degree
10. Remove the node from the graph G’(V,E)
11. Update the adjacency matrix Adj[n][n] and degree array D[n]
12. Find the maximum saturation degree vertex in the residual graph G’(V,E)
13. return #colors

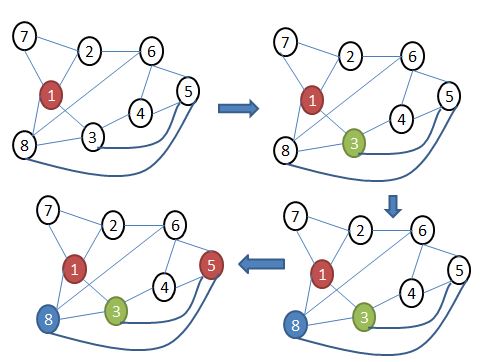
**Output:** number of colors required to color the graph G(V,E) in O(n^2) time complexity

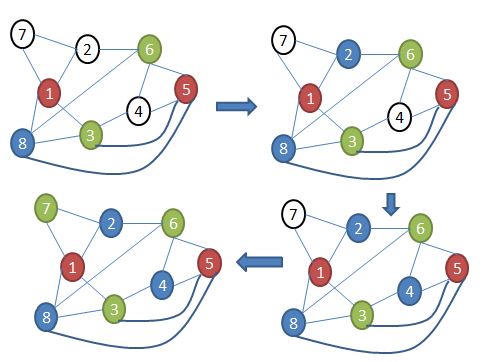
**Example:** Shown in Figure 2.3

**Input:** G=(V,E)

**Order of Vertices:** 1,3,8,5,6,2,4,7

**Output for the example graph:** Chromatic number is 3.



 **Figure: 2.3**

## 2 .4 Incidence Degree Ordering

Let G be a simple graph and C a partial coloration of G vertices. We define the incidence degree of a vertex as the number of colored neighbors to which it is adjacent.

**Incidence Degree Coloring (G, n)**

**Input:** Graph G(V,E)

**Algorithm:**

1. Initialize Adj[n][n], Deg[n],InDeg[n]
2. for i=1 to n do
3. Update Deg[i]
4. For j=1 to n do
5. Update Adj[i][j] &Adj[j][i]
6. Update InDeg[n] of the graph
7. Sort the nodes based on incidence degree of the vertex in descending order
8. for i=1 to n do
9. Color the node with maximum incidence degree with least availables index color and if ambiguity is there then color the node with maximum degree
10. Remove the node from the graph G’(V,E)
11. Update the adjacency matrix Adj[n][n] and degree array D[n]
12. Find the maximum incidence degree vertex in the residual graph G’(V,E)
13. return #colors

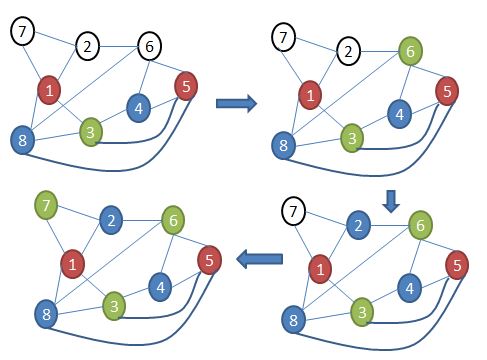
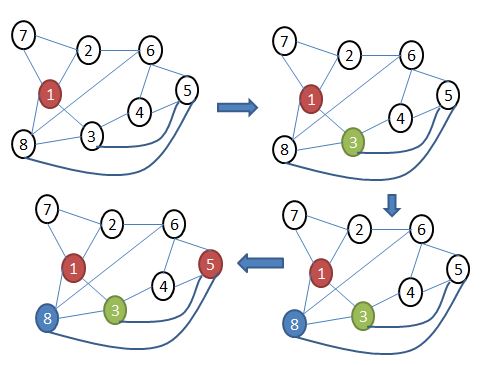
**Output:** number of colors required to color the graph G(V,E) in O(n^2) time complexity.

**Example:** Shown in Figure 2.4

**Input:** G=(V,E)

**Order of vertices:** 1,3,8,5,4,6,2,7

**Output for the example graph:** Chromatic number is 3.

 **Figure: 2.4**

## 2 .5 Largest Degree Ordering

Let G be a simple graph and C a partial coloration of G vertices. We define the largest degree of a vertex as the number of neighbors to which it is adjacent.

**Largest Degree Coloring (G, n)**

**Input:** Graph G(V,E)

**Algorithm:**

1. Initialize Adj[n][n], Deg[n],LaDeg[n]
2. for i=1 to n do
3. Update Deg[i]
4. For j=1 to n do
5. Update Adj[i][j] &Adj[j][i]
6. Update LaDeg[n] of the graph
7. Sort the nodes based on degree of the vertex in descending order
8. for i=1 to n do
9. Color the node with maximum degree with least available index color and if ambiguity is there then color the node with maximum saturation degree
10. Remove the node from the graph G’(V,E)
11. Update the adjacency matrix Adj[n][n] and degree array D[n]
12. Find the maximum degree vertex in the residual graph G’(V,E)
13. return #colors

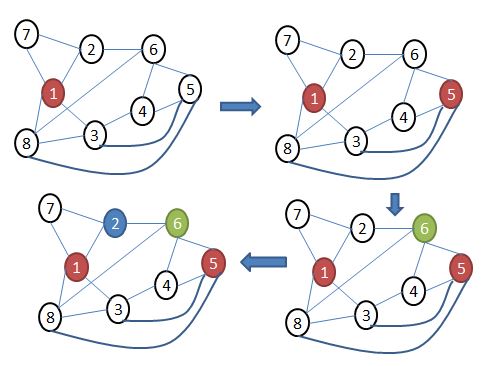
**Output:** No.of colors required to color the graph G(V,E) in O(n^2) time complexity.

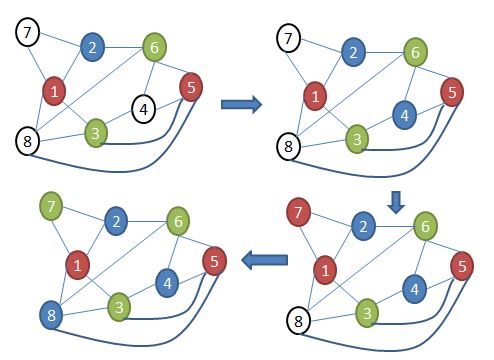
**Example:** Shown in Figure 2.5

**Input:** G=(V,E)

**Order of vertices:** 1, 5,6,2,3,4,7,8

**Output for the example graph:** Chromatic number is 3.





**Figure: 2.5**

## 2 .6 Independent Set

Let G be a simple graph and C a partial coloration of G vertices. In this it finds largest independent set in the remaining uncolored graph and color it with a new color until all the vertices are colored.

**Independent Set Coloring (G, n)**

**Input: Graph G(V,E)**

**Algorithm:**

1. Initialize Adj[n][n], Deg[n],InSet[n][n]
2. for i=1 to n do
3. Update Deg[i]
4. For j=1 to n do
5. Update Adj[i][j] &Adj[j][i]
6. Update InSet[n][n] of the graph
7. While(G(V,E)!=null) do
8. if(i is not belongs to any list I1,I2…..Im)
9. select all the nodes not connected to i and to other nodes in the selected list with each other
10. Remove the nodes of independent set from the graph G’(V,E)
11. Update the adjacency matrix Adj[n][n]
12. Color each independent set I1,I2.….In with different color
13. return #colors

**Output:** No. of colors required to color the graph G(V,E) in O(n^3) time complexity

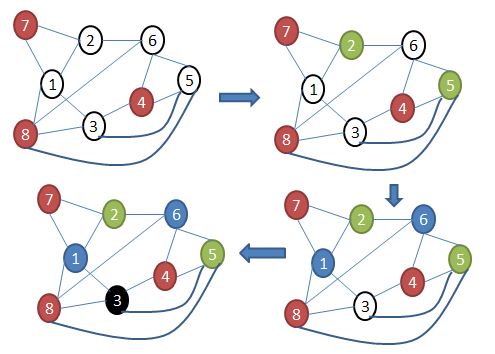
**Example:** Shown in Figure 2.6

**Input:** G(V,E)

**Order of Vertices:**

1. I1={8,4,7}
2. I2={2,5}
3. I3={1,6}
4. I4={3}

**Output:** Chromatic number is 4.



**Figure: 2.6**

## 2.7 Domination Vertex Covering

**Definition 1:** A node “A” in an incompatibility graph covers some other node “B” in the graph if all of the following are satisfied:

1) Node “A” and node “B” have no common edge.

2) Node “A” has edges with all the nodes that node “B” has edges with.

3) Node “A” has at least one more edge than node “B”.

When two nodes have a covering then both the nodes can be colored with the same color.

**Definition 2:** If conditions 1) and 2) for coverings are satisfied and node “A” has the same number of edges as node “B”, then it is called a pseudo-covering.

**Theorem 1:** If any node “A” in a graph covers any other node “B” in the graph, node “B” can be removed from the graph, and in a pseudo-covering any one of the nodes “A”or “B” can be removed.

**Definition3:** A complete graph is one in which all pairs of vertices are connected.

In a complete graph total edges =n\*(n-1)/2, where total edges is the sum of all the edges in a graph. In a complete graph no covering or pseudo coverings can be found and all nodes must have unique color.

**Definition4:** A non-reducible graph is a graph that is not complete and has no covered or pseudo-covered node(s).

**Theorem 2:** If a graph is reducible and can be reduced to a complete graph by successive removing of all its covered and pseudo-covered nodes, then Algorithm DOM finds the coloring with the minimum number of colors (the exact coloring).

**Domination Vertex Covering (G,n):**

**Input:** Graph G(V,E)

**Algorithm:**

1. Initialize Adj[n][n], Deg[n],Buc[n][n/4]
2. for i=1 to n do
3. Update Deg[i]
4. For j=1 to n do
5. Update Adj[i][j] &Adj[j][i]
6. for i=1 to N do
7. If(node i is not colored)
8. for j=1 to N do
9. if (node j is not colored and i!=j and i,j are not connected)
10. if(i covers/pseudo covers j)
11. Then remove j and keep it in i th bucket
12. if (there is any complete sub graphs as a component in the residual graph G’(V,E) )
13. Color the complete graph and its covered and pseudo covered nodes
14. else
15. Remove a node n randomly from the residual graph G’(V,E) and continue till all the nodes are colored
16. Make ith node and its covered nodes as colored
17. return #colors

**Output:** number of colors required to color the graph G(V,E) in O(n^3) time complexity.

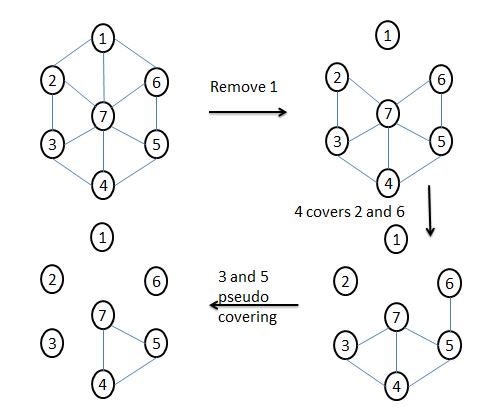
**Example:** Process is shown in Figure 2.7

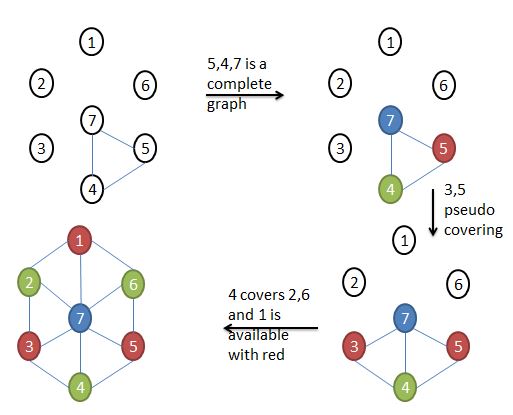
**Input:** G(V,E)

**Order of Vertices:**

1. Random Nodes={1}
2. Covered Nodes Node 4={2,6}
3. Pseudo Covered Node {3,5}
4. Complete Graph of size 3 {4,5,7}

**Output:** Chromatic number is 3





**Figure: 2.7**

# Chapter 3

# Computational Results

Computational Results shows the efficiency of the algorithms in finding graph coloring for the given graph. In this chapter we have given computational results of different graphs like benchmark graphs for graph coloring i.e DIMAC graphs and benchmark graphs for Clique problem.

## 3.1 DIMAC Graphs

DIMAC graphs are benchmark graphs for graph coloring problems. The Colored results in Table 3.1 showing the related algorithm is giving bet solution among all algorithms for the particular graph.

**Table 3.1**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **No.of vertices** | **No.of Edges** | **#colors** | **LDO** | **IDO** | **Ind Set** | **Domination Vertex Covering**  **(#colors)** | |
| **DIMAC Graphs** |  |  |  | **#colors** | **#colors** | **#colors** | **LDO** | **IDO** |
| abb313GPIA.col | 1557 | 65390 | ? | 12 | 15 | 18 | 13 | 15 |
| ash331GPIA | 662 | 4185 | ? | 6 | 8 | 10 | 6 | 7 |
| ash608GPIA | 1216 | 7844 | ? | 6 | 8 | 10 | 6 | 9 |
| ash958GPIA | 1916 | 12506 | ? | 7 | 10 | 10 | 7 | 7 |
| DSJC1000.1 | 1000 | 49629 | 20 | 28 | 30 | 29 | 47 | 48 |
| DSJC1000.5 | 1000 | 499652 | 83 | 121 | 127 | 121 | 121 | 127 |
| DSJC1000.5 | 1000 | 249826 | 85 | 121 | 127 | 121 | 121 | 127 |
| DSJC1000.9 | 1000 | 449449 | 223 | 300 | 300 | 313 | 306 | 317 |
| DSJC250.1 | 250 | 3218 | 8 | 11 | 11 | 11 | 11 | 13 |
| DSJC250.5 | 250 | 15668 | 28 | 39 | 41 | 41 | 39 | 41 |
| DSJC250.9 | 250 | 27897 | 72 | 89 | 95 | 93 | 95 | 95 |
| DSJC500.1 | 500 | 12458 | 12 | 18 | 20 | 18 | 17 | 18 |
| DSJC500.5 | 500 | 125248 | 48 | 69 | 71 | 71 | 69 | 71 |
| DSJC500.5 | 500 | 62624 | 48 | 69 | 71 | 71 | 69 | 71 |
| DSJC500.9 | 500 | 224874 | 126 | 166 | 177 | 169 | 166 | 177 |
| DSJR500.1 | 500 | 3555 | 12 | 18 | 20 | 13 | 13 | 13 |
| DSJR500.1c | 500 | 121275 | 84 | 96 | 103 | 100 | 93 | 94 |
| DSJR500.5 | 500 | 58862 | 122 | 136 | 129 | 134 | 136 | 129 |
| latin\_square\_10.col | 900 | 307350 | ? | 148 | 184 | 213 | 148 | 184 |
| le450\_15a.col | 450 | 8168 | 15 | 18 | 19 | 18 | 18 | 17 |
| le450\_15b.col | 450 | 8169 | 15 | 18 | 18 | 18 | 18 | 18 |
| le450\_15c.col | 450 | 16680 | 15 | 26 | 27 | 26 | 26 | 27 |
| le450\_15d.col | 450 | 16750 | 15 | 26 | 26 | 26 | 26 | 26 |
| le450\_25b.col | 450 | 8263 | 25 | 25 | 25 | 25 | 25 | 25 |
| le450\_25c.col | 450 | 17343 | 25 | 31 | 31 | 29 | 30 | 30 |
| le450\_25d.col | 450 | 17425 | 25 | 31 | 29 | 30 | 31 | 31 |
| le450\_5a.col | 450 | 5714 | 5 | 12 | 11 | 11 | 11 | 11 |
| le450\_5b.col | 450 | 5734 | 5 | 15 | 15 | 12 | 11 | 11 |
| le450\_5c.col | 450 | 9803 | 5 | 10 | 12 | 12 | 10 | 13 |
| le450\_5d.col | 450 | 9757 | 5 | 10 | 14 | 14 | 11 | 12 |

## 3.2 Clique Graphs

Clique graphs are benchmark graphs for finding Clique in a graph and graph coloring problems. The Colored results in Figure 3.2 showing the related algorithm is giving best solution among all algorithms for the particular graph.

**Table 3.2**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **No.of Vertices** | **No.of Edges** | **LDO(#colors)** | | | **IDO(#colors)** | | | **Ind-Set** | **DOM(#colors)** | |
| **Least Color** | **Random** | **Half Random** | **Least Color** | **Random** | **Half Random** | **LDO** | **IDO** |
| brock2001\_3.cq.b | 200 | 12048 | 39 | 39 | 39 | 43 | 47 | 45 | 42 | 41 | 41 |
| brock2001\_4.cq.b | 200 | 13089 | 44 | 46 | 45 | 45 | 52 | 48 | 46 | 44 | 49 |
| brock4001\_1.cq.b | 400 | 59723 | 92 | 95 | 92 | 98 | 104 | 98 | 98 | 92 | 98 |
| brock4001\_2.cq.b | 400 | 59786 | 96 | 98 | 97 | 101 | 104 | 98 | 100 | 96 | 101 |
| brock4001\_3.cq.b | 400 | 59681 | 95 | 98 | 94 | 100 | 108 | 103 | 97 | 95 | 100 |
| brock4001\_4.cq.b | 400 | 59765 | 95 | 98 | 98 | 101 | 105 | 99 | 96 | 96 | 101 |
| brock8001\_1.cq.b | 800 |  | 139 | 145 | 139 | 140 | 154 | 145 | 142 | 139 | 140 |
| brock8001\_2.cq.b | 800 |  | 138 | 145 | 141 | 143 | 154 | 145 | 142 | 138 | 143 |
| brock8001\_3.cq.b | 800 |  | 137 | 143 | 137 | 143 | 156 | 143 | 141 | 137 | 143 |
| brock8001\_4.cq.b | 800 |  | 138 | 149 | 144 | 142 | 152 | 143 | 142 | 138 | 142 |
| C-fat200-1.clq | 200 | 3235 | 13 | 13 | 13 | 15 | 12 | 15 | 14 | 15 | 13 |
| C-fat200-2.clq | 200 | 3235 | 24 | 24 | 24 | 24 | 25 | 24 | 24 | 24 | 24 |
| C-fat200-5.clq | 200 | 8473 | 82 | 82 | 82 | 84 | 84 | 84 | 84 | 84 | 84 |
| C-fat500-1.clq | 500 | 4459 | 18 | 18 | 18 | 14 | 15 | 16 | 14 | 17 | 14 |
| C-fat500-10.clq | 500 | 46627 | 127 | 127 | 127 | 126 | 126 | 126 | 126 | 127 | 126 |
| C-fat500-2.clq | 500 | 9139 | 33 | 33 | 33 | 26 | 27 | 26 | 26 | 33 | 26 |
| C-fat500-5.clq | 500 | 23191 | 78 | 79 | 79 | 64 | 66 | 64 | 64 | 78 | 64 |
| gen200\_p0.9\_44.b | 200 | 17910 | 68 | 72 | 69 | 62 | 70 | 66 | 68 | 66 | 60 |
| gen200\_p0.9\_55.b | 200 | 17910 | 71 | 74 | 73 | 73 | 78 | 74 | 74 | 71 | 72 |
| gen400-p0.9-55 | 400 |  | 120 | 128 | 126 | 94 | 104 | 101 | 129 | 122 | 87 |
| gen400-p0.9-65 | 400 |  | 130 | 136 | 130 | 87 | 96 | 91 | 129 | 131 | 84 |
| gen400-p0.9-75 | 400 |  | 129 | 138 | 139 | 101 | 115 | 101 | 128 | 132 | 101 |
| hamming6-4.clq | 64 | 704 | 8 | 8 | 8 | 9 | 9 | 10 | 8 | 8 | 9 |
| keller4.clq.b | 171 | 9435 | 32 | 32 | 32 | 35 | 34 | 31 | 37 | 31 | 31 |
| Keller5.clq.b | 776 |  | 107 | 116 | 106 | 118 | 123 | 114 | 175 | 123 | 108 |
| keller6.clq.b | 3361 |  | 260 | 312 | 285 | 380 | 400 | 350 | 781 | 380 | 260 |
| MANN\_a27.clq | 378 | 70551 | 141 | 141 | 141 | 144 | 144 | 144 | 144 | 126 | 126 |
| MANN\_a45.clq | 1035 |  | 372 | 372 | 372 | 375 | 375 | 375 | 375 | 338 | 338 |
| r200.5 | 200 | 10036 | 34 | 35 | 34 | 36 | 35 | 36 | 35 | 34 | 36 |
| r400.5 | 400 | 40061 | 57 | 58 | 57 | 59 | 58 | 59 | 58 | 57 | 59 |
| r500.5 | 500 | 62161 | 68 | 70 | 68 | 69 | 68 | 69 | 69 | 68 | 69 |

# Chapter 4

# Comparison of Literature Work

Comparison of Literature work mentions different research papers and its proposed algorithms. It mainly discuss about the history of graph coloring algorithms and their drawbacks in performance on different graphs.

## 4.1 Comparison Table

Comparison table compares the performance of different algorithms on given input graphs. Figure 4.1 is showing the performance problems of different graphs in the related research paper and the authors and publishing year of the paper.

**Table 4.1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Year** | **Author** | **Methodology** | **Remark on Performance** |
| 1967 | D. J. A Welsh and M. B. Powel | LDO[15] | Output is not optimal |
| 1979 | Daniel Brelaz | Greedy Algorithm[3] | Doesn’t depend on past or future problems |
| 1979 | Daniel Brelaz | SDO[3] | Output is not Optimal always |
| 1983 | T. Coleman and J. More | IDO[16] | Output is not optimal always |
| 1986 | N. Alon, L. Babai, and A.Itai | Independent Set[13] | Finding Independent set is NPC |
| 1999 | Marek Perkowski, Rahul Malvi, Stan Grygiel, Mike Burns, and Alan Mishchenko | Domination Covering for Graph Coloring[18] | Gives optimal solution for some type graphs only |
| 2000 | Dimitris A. Fotakis, Spiridon D. Likothanassis, and Stamatis  K. Stefanakos | Evolutionary Annealing Approach | Wont give better results if instance size increases |
| 2006 | Dr. Hussein Al-Omari and Khair Eddin Sabri | LDO with IDO[3] | Output is not optimal, but it gives better output than LDO |
| 2006 | Dr. Hussein Al-Omari and Khair Eddin Sabri | SDO with IDO[3] | Output is not optimal, but it gives better output than SDO |
| 2011 | Qinghua Wu and Jin-Kao Hao | EXTRACOL[19] | Gives optimal solution for some graphs only and independent set extraction is also a NPC |
| 2011 | Linda Ouerfelli, Hend Bouziri | DSATUR | Not giving optimal solution for dense graph |
| 2012 | Hilal ALMARA’BEH, Amjad SULEIMAN | LDO[20],SDO[20],Min-Max[20] | Giving better output for Sparse Graphs |

# Chapter 5

# Applications of Graph Coloring

In the real time applications graph coloring solves many problems. Some of the applications of graph coloring are mentioned below as follows

* Scheduling Final Exams
* Traffic signal design
* Sudoku
* 8 Queen Problem
* Register allocation
* VLSI channel routing
* Testing printed circuit boards (Garey, Johnson, & Hing)

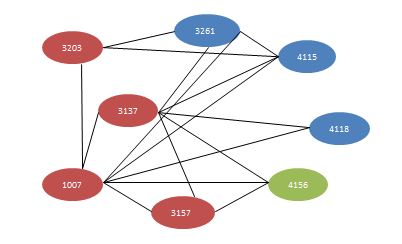
## 5.1 Scheduling Exams:

Final Exam Example: As shown in Figure 5.1 suppose if we want to schedule some final exams for CS courses with following course numbers:1007, 3137, 3157, 3203, 3261, 4115, 4118, 4156.The procedure and constraints to be followed for scheduling are as follows

* Suppose also that there are no students in common taking

the following pairs of courses:

* + 1007‐3137
  + 1007‐3157, 3137‐3157
  + 1007‐3203
  + 1007‐3261, 3137‐3261, 3203‐3261
  + 1007‐4115, 3137‐4115, 3203‐4115, 3261‐4115
  + 1007‐4118, 3137‐4118
  + 1007‐4156, 3137‐4156, 3157‐4156
* How many exam slots are necessary to schedule exams?



**Figure: 5.1**

* Convert problem into a graph coloring problem.
* Courses are represented by vertices.
* Two vertices are connected with an edge if the corresponding courses have a student in common.
* One way to do this is to put edges down where students mutually excluded and then compute the complementary graph and chromatic number is equal to 3 i.e. three slots are needed to complete the exams without collisions .

**Output:** Chromatic number is 3.Each color represents a single slot.

## 5.2 Register Allocation

Register Allocation for the storage of data is very critical in scheduling processes. In this we need to allocate the registers to the variables such that there will not be any conflicts in retrieving the data.

**Register input Steps:**

1. *L1:*  *li a, 0*
2. *loop: addi b, a, 1*
3. *L2:add c, c, b*
4. *L3:muli a, b, 2*
5. *L4: blt a, n, loop*
6. *L5:print c*

**Label**  **Live-i**

1. *L1*  *c, n*
2. *loop*  *a, c, n*
3. *L2*  *b, c, n*
4. *L3*  *b, c, n*
5. *L4*  *a, c, n*
6. *L5*  *c*

**Process:**

1. Make a node for each register

2. Are V1 and V2 ever live together, if so keep an edge between both the vertices.

3. Do the coloring for the graph.

## 5.3 Sudoku

Sudoku is puzzle game where we need solve the puzzle such that any common number will not be there in the same row or column or 3x3 grid. The procedure and the constraints that we are following to solve this is as follows

* Each cell is a vertex
* Each integer label is a “color”
* A vertex is adjacent to another vertex if one of the following hold:
  + Same row
  + Same column
  + Same 3x3 grid
* Vertex-coloring solves Sudoku

## 5.4Traffic Signal Design

In Traffic Signal Design the goal is to reduce the waiting time for cars before they get green signal. So for reducing that time we can solve this by using graph coloring by representing the problem as graph as follows

* At an intersection of roads, we want to install traffic signal lights which will periodically switch between green and red.
* This problem can be modeled as a coloring problem.
* Each path that crosses the intersection is a node. If two paths intersect each other, there is an edge connecting them. Each color represents a time slot at which the path gets a green light.

## 5.5 8 Queen Problem:

In 8 Queen problem we need place 8 queens on a chess board so that the main constraints to follow are as follows

* No two Queens should meet in a column
* No two Queens should meet in a row
* No two Queens should meet diagonally

To solve this using graph we need convert this into graph and the constraints followed are as follows

* Each cell is a vertex
* Each integer label is a “color”
* A vertex is adjacent to another vertex if one of the following hold:
* Same row
* Same column
* Same Diagonal Element

After that Vertex-coloring for the output graph solves Sudoku with 9 different colors.

# Chapter 6

# Conclusion

Using all the mentioned algorithms we can find the chromatic number of the given graph and the solution is around optimal solution with polynomial time complexity using all the algorithms and its properties. In future by using different heuristics dynamically and applying artificial intelligence we can make the efficient algorithm, finds best solution around optimal solution with in polynomial time complexity.

# REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms,* Third Edition. MIT Press/McGraw-HillHigher Education, 2009.

[2] Narsingh Deo, *Graph Theory with Application to Engineering and Computer Science*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

[3] Daniel Brelaz, *New Methods to Color the Vertices of a Graph*,, ACM New York, Vol 22 Issue4,1979.

[4] Dr. Hussein Al-Omari and Khair Eddin Sabri, *New Graph Coloring Algorithms*, Computer Science Department, Applied Science University, Amman, Jordan, 2006.

[5] Desislava Kukova, *Chromatic Number* , 30 Car Osvoboditel STR, ap. 7, 9000 Varna, Bulgaria, desi\_kukova@abv.bg

[6] Marco Chiarandini and Thomas Stutzle, *An Analysis of Heuristics for Vertex Coloring,* University of Southern Denmark, Campusvej 55,Odense, Denmark,2010.

[7] J. Randall-Brown, “Chromatic scheduling and the chromatic number problems,” Management Science 19(4), Part l, pp. 456-463, 1972.

[8] P. Cheeseman, B. Kenefsky, and W. Taylor, Where the really hard problems are, In J.Mylopoulos and R. Reiter (Eds.), Proceedings of 2th International Joint Conference on AI (IJCAI-91), Volume 1, pp. 331–337., 1991.

[9] S. Vishwanathan, Randomized online graph coloring, Journal of Algorithms,1992.

[10] A. Miller, Online graph colouring, Canadian Under graduate Mathematics Conference, 2004.

[11] [http://en.wikipedia.orglwiki/Greedy\_algorithm]

[12] [http://www.cse.ohiostate.edul-gurari/course/cis680/cis680ChI7.html#QQ 1-49-1 07]

[13] N. Alon, L. Babai, and A.Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, J. Algorithms , 1986.

[14] P. Briggs. Register allocation via graph coloring. PhD thesis, Rice University , 1992.

[15] D. J. A Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to time tabling problems. The computer Journal, 1967.

[16] T. Coleman and J. More. Estimation of sparse Jacobian matrices and graph coloring problems. SIAM J. Numer. Anal., 1983.

[17] Vlastimil Chytry. Sudoku Game solution based on Graph Theory and Suitable for School Mathematics. 2014

[18] Marek Perkowski, Rahul Malvi, Stan Grygiel, Mike Burns, and Alan Mishchenko. Graph Coloring Algorithms for Fast Evaluation of Curtis Decompositions, Portlan University, USA, mperkows@ee.pdx.edu.2010.

[19] Qinghua Wu and Jin-Kao Hao, Coloring Large Graphs based on Independent Set Extraction, LERIA,France ,2011.

[20] Hilal ALMARA’BEH, Amjad SULEIMAN, Heuristic Algorithm for Graph Coloring Based On Maximum Independent Set, *Journal of Applied Computer Science & Mathematics, no. 13 (6) , Suceava*, Saudi Arabia,2012.