A Novel Edge Cover based Graph Coloring Algorithm

Harish Patidar
Research Scholar
Department of Computer Science and Engineering
Sir Padampat Singhania University
Udaipur, India

Dr. Prasun Chakrabarti
Professor and Head
Department of Computer Science and Engineering
Sir Padampat Singhania University
Udaipur, India

***Abstract*—Graph Colouring Problem is a well-known NP-Hard problem. In Graph Colouring Problem (GCP) all vertices of any graph must be coloured in such a way that no two adjacent vertices are coloured with the same colour. In this paper, a new algorithm is proposed to solve the GCP. Proposed algorithm is based on finding vertex sets using edge cover method. In this paper implementation prospective of the algorithm is also discussed. Implemented algorithm is tested on various graph instances of DIMACS standards dataset. Algorithm execution time and a number of colours required to colour graph are compared with some other well-known Graph Colouring Algorithms. Variation in time complexity with reference to increasing in the number of vertices, a number of edges and an average degree of a graph are also discussed in this paper.**

***Keywords*—Graph Colouring Problem; Edge Cover; Independent Set; NP-Hard Problem**

I. Introduction

Graph Colouring Problem is used to the optimal solution of many real world practical applications like Time table scheduling [13], Air traffic flow management [29], Frequency assignment and Computer gaming. The graph colouring problem is defined as follows. Let G= (V, E) is a graph with |V| is a number of vertices and |E| is a number of edges, which connects vertices to each other. The edges are of the form (a, b) where a, b ∈ E. The problem of graph colouring is to assign a colour to each vertex a ∈ V such that a and b does not colour with the same colour.

Finding the optimum solution in optimum time is always the objective of researchers. In general colouring optimisation is the primary objective of graph colouring algorithms. But when it comes to a large graph where a number of vertices and number of edges are in large number, time complexity is more important than colouring optimisation. For example genetic algorithm with multipoint guided mutation algorithm (MSGCA) generate optimum chromatic number (5) for graph instance 4-Insertion_4, i.e. number of colours required to colour graph of 475 vertices and 1795 edges are five. But algorithm takes 1071 seconds to complete execution [8]. And proposed algorithm gives the same chromatic number and generates results in 0.41 second only.

Today, graph colouring algorithms are used for many internet applications, social media websites where graph size is very large. And user required fast results of their web access.

Rest of the paper is organised as follows: In section II, related work done by researchers in the field of graph colouring is discussed. In section III problem with the existing algorithm is highlighted. In section IV an algorithm is proposed to solve the problem highlighted in section III. In section V experimental results of proposed algorithm on DIMACS graph instances are shown. In section VI, results analysis is done on the bases of experimental results and results are also compared with some other well-known graph colouring algorithms. In Section VII, the conclusion of research work is discussed and future enhancement in proposed algorithm is also discussed.

II. Related Work and Background

There are already so many approaches to solving the GCP given by the researchers. These approaches are widely divided into two categories: (1) approximate [2], and (2) exact. The approximate approach does not give the best solution but can give a result with the large graphs. The algorithm developed by exact approach gives satisfactory results but most of the exact algorithms are not suitable for large graphs.

On the basis of an execution graph colouring algorithm can be sequential and parallel. There are number of algorithm like, Cuckoo optimisation algorithm [3], modified cuckoo optimisation algorithm [4], polynomial 3-SAT encoding algorithm [5], Ant colony optimisation algorithm [6], Mimetic algorithm [7], GA with multipoint guided mutation algorithm [8] many more are sequential graph colouring algorithm. On the other hand Parallel largest-log-degree-first (LLF) [9], Parallel smallest-log-degree-first (SLF) [9], a parallel algorithm based on BRS [10], parallel graph colouring on multi core CPUs [11] are a parallel algorithm. The parallel algorithm is more time efficient then sequential algorithm due to parallel execution of different iterations of the algorithm.

III. Problem Identification

The primary objective of graph colouring algorithm is to find the optimum chromatic number (number of colours required to colour all vertices of the graph), but when graph size is large and average vertex degree of a graph is high, the time complexity of the algorithm is more important than the chromatic number. For the large graphs algorithm execution time should be finite and optimum. In a review of different kinds of literature it has been found that most of the algorithms are not able to colour large graphs in optimum time.

IV. PROPOSED ALGORITHM

In this paper, edges cover based graph colouring algorithm is proposed. This proposed algorithm full fill the need of optimum time complexity for large graphs. This algorithm is based on finding an independent set (not a single connecting edge between vertices) of vertices using edges cover technique. The algorithm is able to give results for all kinds of graph instances successfully. Execution time is also optimum for large graphs.

A. Edges Cover Technique

Edge cover technique is a selection of vertices of any graph in such a manner that all edges of the graph will be covered. The remaining vertices set is called independent set. There should be minimum vertices in edge cover vertices set, to get maximum independent set.

$$V_{(EC)} + V_{(I)} = V \tag{1}$$

where,
$V_{(EC)}$ is set of Edge cover vertices.
$V_{(I)}$ is set of Independent vertices in the graph.
V is set of all vertices of the graph.

B. Edge Cover Graph Coloring Algorithm

Proposed Edges cover graph colouring algorithm works in an iterative manner. Each iteration gives a single set of vertices. This set contains vertices independent to each other, so that each vertex of the set can assign a single colour. The behaviour of iteration depends on a number of sets. For the large graph it is difficult to predict a number of sets. Figure 1 shows algorithm flow and different iterations.

Proposed algorithm takes the graph instance as input in the form of adjacency edge list. The algorithm generates a certain number of vertices sets as an output each set of vertices can be coloured with the same colour.

C. Complexity Analysis of Algorithm

Proposed graph colouring algorithm is NP-hard in nature. So it is hard to determine the complexity hypothetically. The complexity of algorithm depends on a number of independent sets. A number of independent sets are unpredictable. Proposed algorithm works on iterations. All iterations have three parts where maximum execution time is required.

First: when the degree of vertices is calculated. Equation (2) shows the complexity of calculating the degree of vertices in determining the single independent set.

$$|Nv|*|Ne| \tag{2}$$

where,

Nv is a number of vertices in vertex set.

Ne is a number of edges in edge set.

At the end of algorithm execution if algorithm generates total k independent sets then the total complexity of calculating the degree of all vertices in all iterations is shown in equation (3).

$$\sum_{i=1}^{k} (|Nvi|*|Nei|) \tag{3}$$

where,

Nvi is a number of vertices in vertex set while finding ith independent set.

Nei is Number of edges in edge set while finding ith independent set.

k is a number of independent sets

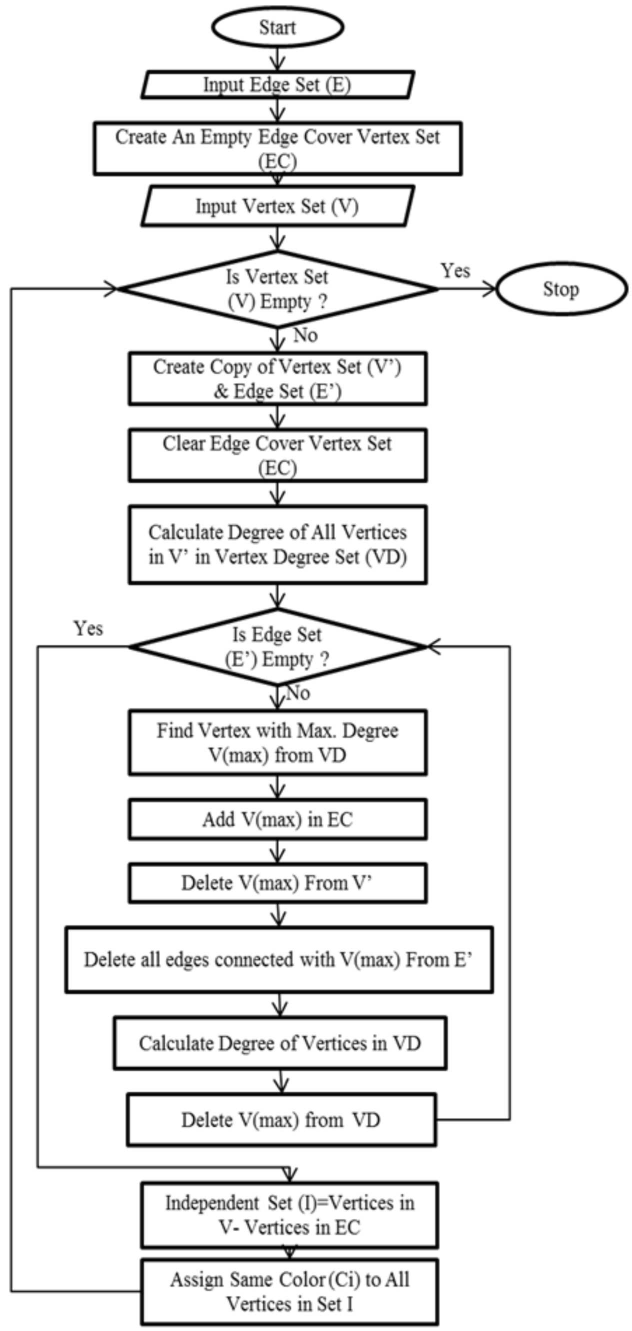

Fig. 1. Flow chart of algoritm

Second, time complexity in finding maximum degree vertices is shown by equation (4).

$$\sum_{i=1}^{k} (|Vi|*(|Vi|+1))/2 \tag{4}$$

where,

Vi is a number of vertices in vertex set while finding ith independent set.

k is Number of independent sets.

And third is when edge set required editing. Complexity to update edge set in all iteration of the algorithm can be evaluated by equation (5).

$$\sum_{i=1}^{k} (|\text{Vec i}|*|\text{Degree}(\text{Vmax})|*|\text{Eec}|) \quad (5)$$

where,

Veci is a number of vertices in edge cover set while finding ith independent set.

Degree(Vmax) is a degree of maximum degree vertex.

Eec is a number of edges connected to vertices available in edge cover set.

V. EXPERIMANTAL RESULTS

To evaluate the proposed algorithm DIMACS graph instances are used. DIMACS instances of graphs are introduced by scientists for graph colouring problem. Most of the graph colouring algorithms are tested on DIMACS graph instances. Some graphs of DIMACS are generated randomly by computer programs and some of them are results of real world applications.

Proposed algorithm is implemented in JAVA Programming language (jdk1.8.0_74). Eclipse JUNA Editor is used to write the program. Operating system Windows Server 2012 Standard 64-bit is used. Intel Pentium Dual CPU G640 @2.80Ghz with 2 GB RAM is used for implementation and result evaluation.

In this section of paper, test results on DIMACS graph instances are shown. Test results are shown in the tabular form. Each table contains graph Instance name, Number of vertices (V) in the graph, Number of edges connected to vertices (E), Number of coloured required to colour graph (K) which is generated by an algorithm, and Time (in Seconds) taken by the algorithm to execute.

A. DSJC Series Graphs Results

Table 1 shows the DSJC series of instances results. They are random graphs used in the paper by David S. Johnson.

TABLE I. DSJC GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
DSJC125.1	125	736	8	0.125
DSJC125.5	125	3891	25	0.797
DSJC125.9	125	6961	56	1.739
DSJC250.1	250	3218	12	0.673
DSJC250.5	250	15668	42	3.578
DSJC250.9	250	27897	94	13.932
DSJC500.1	500	12458	19	2.328
DSJC500.5	500	62624	73	41.642
DSJC500.9	500	224874	168	209.882

B. DSJRx Graphs Results

DSJRx graph instances are geometric random graphs with x nodes randomly distributed in the unit square. These graphs are used in a paper by David S. Johnson. Table 2 shows the proposed algorithm results.

TABLE II. DSJRX GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
DSJR500.1	500	3555	15	1.265
DSJR500.1c	500	121275	103	599.203
DSJR500.5	500	58862	197	493.005

C. Myciel Graphs Results

Myciel graphs are based on the Mycielski transformation and they are triangle free graphs. Table 3 show the results of myciel graphs on proposed algorithm.

TABLE III. MYCIEL GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
myciel3	11	20	4	0.016
myciel4	23	71	5	0.031
myciel5	47	236	6	0.094
myciel6	95	755	7	0.188
myciel7	191	2360	10	0.422

D. k-Insertion graphs and Full Insertion graphs results

k-insertion graphs and full insertion graphs are also tested on proposed algorithm. These graphs are a generalisation of myciel graphs with inserted nodes to increase graph size but not density. These instances are created by M. Caramia and P. Dell'Olmo. Table 4 shows the results of k-insertion graphs and full insertion graphs.

TABLE IV. K-INSERTION AND FULL INSERTION GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colors (K)	Time (s)
1-FullIns_3	30	100	4	0.032
1-FullIns_4	93	593	5	0.14
1-FullIns_5	282	3247	6	0.469
1-Insertions_4	67	232	5	0.063
1-Insertions_5	202	1227	6	0.219
1-Insertions_6	607	6337	7	0.953
2-FullIns_3	52	201	5	0.078
2-FullIns_4	212	1621	6	0.252
2-FullIns_5	852	12201	8	1.484
2-Insertions_3	37	72	4	0.016
2-Insertions_4	149	541	5	0.161
2-Insertions_5	597	3936	8	0.75
3-FullIns_3	80	346	6	0.078
3-FullIns_4	405	3524	8	0.594
3-FullIns_5	2030	33751	9	6.789
3-Insertions_3	56	110	4	0.031
3-Insertions_4	281	1046	5	0.219
3-Insertions_5	1406	9695	7	1.858
4-FullIns_3	114	541	8	0.187
4-FullIns_4	690	6650	9	0.985
4-FullIns_5	4146	77305	10	33.566
4-Insertions_3	79	156	4	0.078
4-Insertions_4	475	1795	5	0.406
5-FullIns_3	154	792	8	0.188
5-FullIns_4	1085	11395	10	1.422

E. Matrix Partitioning Problem Graphs Results

These graphs are generated by Matrix partitioning problem. Graphs from a matrix partitioning problem in the segmented columns approach to determine sparse Jacobian matrices. Table 5 shows the results of proposed algorithm on these graphs.

TABLE V. MATRIX PARTITIONING PROBLEM GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
ash331GPIA	662	4185	6	0.953
ash608GPIA	1216	7844	6	1.797
ash958GPIA	1916	12506	6	3.25

F. Register Allocation Problem Graphs Results

Proposed algorithm is also tested on graph instances generated by register allocation problem. Table 6 shows the results of register allocation problem generated graphs.

TABLE VI. REGISTER ALLOCATION PROBLEM GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
fpsol2.i.1	496	11654	65	1.954
fpsol2.i.2	451	8691	31	1.328
fpsol2.i.3	425	8688	31	1.297
inithx.i.1	864	18707	54	2.969
inithx.i.2	645	13979	31	2.062
inithx.i.3	621	13969	31	1.944
mulsol.i.1	197	3925	49	0.848
mulsol.i.2	188	3885	31	0.624
mulsol.i.3	184	3916	31	0.578
mulsol.i.4	185	3946	31	0.592
mulsol.i.5	186	3973	31	0.577
zeroin.i.1	211	4100	51	0.902
zeroin.i.2	211	3541	32	0.562
zeroin.i.3	206	3540	32	0.526

G. Latin Square Problem Graphs Results

The problem corresponds to assigning colours to the cells of an empty matrix such that there is no repetition of colours in each row/column of the matrix is called Latin Square Problem. Some graphs are generated by Latin square problem are also used to test the proposed algorithm. Table 7 shows the results of graphs generated by Latin square problem.

TABLE VII. LATIN SQUARE PROBLEM GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
qg.order100	10000	990000	128	20540.5
qg.order30	900	26100	40	17.441
qg.order40	1600	62400	60	96.171
qg.order60	3600	212400	82	978.151
latin_square_10	900	307350	152	1095.71

H. Leighton Graphs Results

Leighton graphs are generated by Leighton's graph covering theorem (Two finite graphs which have a common covering have a common finite covering). Leighton graphs results on proposed algorithm are shown in Table 8.

TABLE VIII. LEIGHTON GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
le450_15a	450	8168	23	1.817
le450_15b	450	8169	23	1.736
le450_15c	450	16680	33	3.69
le450_15d	450	16750	34	3.789
le450_25a	450	8260	33	1.907
le450_25b	450	8263	30	2
le450_25c	450	17343	39	4.063
le450_25d	450	17425	40	4.598
le450_5a	450	5714	11	1.11
le450_5b	450	5734	13	1.188
le450_5c	450	9803	9	1.143
le450_5d	450	9757	8	1.266

I. Miles Graphs Results

In miles graphs nodes are placed in space with two nodes connected if they are close enough. The nodes represent a set of United States cities. Proposed algorithm test results are shown in Table 9.

TABLE IX. MILES GRAPHS TEST RESULT

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
miles1000	128	6432	51	1.406
miles1500	128	10396	81	2.588
miles250	128	774	10	0.18
miles500	128	2340	26	0.422
miles750	128	4226	39	0.953

J. Queen Graphs Results

A queen graph is a graph on n^2 nodes, each corresponding to a square of the board. Two nodes are connected by an edge if the corresponding squares are in the same row, column, or diagonal. 13 different instances of queen problem are tested on proposed algorithm. The test result is shown in Table 10.

TABLE X. QUEEN PROBLEM GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
queen10_10	100	2940	17	0.437
queen11_11	121	3960	18	0.703
queen12_12	144	5192	19	0.859
queen13_13	169	6656	20	1.046
queen14_14	196	8372	21	1.375
queen15_15	225	10360	25	1.86
queen16_16	256	12640	27	2.221
queen5_5	25	320	7	0.094
queen6_6	36	580	10	0.125
queen7_7	49	952	12	0.203
queen8_12	96	2736	15	0.468

K. School Scheduling Graphs Results

School scheduling graphs are generated for scheduling the classes of school. Test results are shown in Table 11.

TABLE XII. SCHOOL SCHEDULING GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
school1	385	19095	43	4.682
school1_nsh	352	14612	40	2.924

L. Large Random Graph Result

Proposed algorithm is also tested on a random graph. This graph has 2000 vertices and 999836 edges. Table 12 shows the number of coloured and execution time of proposed algorithm.

TABLE XIII. RANDOME LARAGE GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
C2000.5	2000	999836	239	19091.7

M. Quasi-random coloring problem generated graphs results

Graph generated by Quasi-random colouring problem test results are shown in Table 13.

TABLE XIV. QUASI-RANDOM COLORING PROBLEM GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
flat1000_50_0	1000	245000	125	698.714
flat1000_60_0	1000	245830	125	697.875
flat1000_76_0	1000	246708	128	642.514
flat300_28_0	300	21695	45	5.954
R50_1g	50	108	5	0.047
R50_1gb	50	108	5	0.047
R50_5g	50	612	15	0.093
R50_5gb	50	612	15	0.124
R50_9g	50	1092	25	0.265
R50_9gb	50	1092	25	0.251
R75_1g	70	251	6	0.063
R75_1gb	70	251	6	0.078
R75_5g	75	1407	16	0.234
R75_5gb	75	1407	16	0.281
R75_9g	75	2513	39	0.577
R75_9gb	75	2513	39	0.593

N. Geometric Random Graphs Results

Geometric random graphs test result on proposed algorithm is shown in Table 14.

TABLE XV. GEOMETRIC RANDOM GRAPHS TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
r1000.1c	1000	485090	124	1220.47
r1000.5	1000	238267	411	2035.23
r250.5	250	14849	101	7.327

O. Geometric Graph with Bandwidth and Node Weights Graphs Results

In these graph instances bandwidth of each edge and weights of nodes are given. Proposed algorithm tested by ignoring edges bandwidth and nodes weight. Results of geometric graphs are shown in Table 15.

TABLE XVI. GEOMETRIC GRAPHS WITH BANDWIDTH AND NODE WEIGHT TEST RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
GEOM100	100	647	10	0.14
GEOM100a	100	1092	16	0.219
GEOM100b	100	1150	20	0.234
GEOM110	110	748	11	0.171
GEOM110a	110	1317	19	0.234
GEOM110b	110	1366	21	0.281
GEOM120	120	893	11	0.187
GEOM120a	120	1554	21	0.312
GEOM120b	120	1611	23	0.328
GEOM20	20	40	5	0.016
GEOM20a	20	57	6	0.031
GEOM20b	20	52	4	0.032
GEOM30	30	80	6	0.031
GEOM30a	30	111	7	0.046
GEOM30b	30	111	6	0.031
GEOM40	40	118	6	0.047
GEOM40a	40	186	8	0.062
GEOM40b	40	197	7	0.093
GEOM50	50	177	6	0.062
GEOM50a	50	288	11	0.078
GEOM50b	50	299	10	0.094
GEOM60	60	245	7	0.062
GEOM60a	60	399	11	0.093
GEOM60b	60	426	12	0.124
GEOM70	70	337	9	0.078
GEOM70a	70	529	12	0.125
GEOM70b	70	558	12	0.156
GEOM80	80	429	8	0.125
GEOM80a	80	692	14	0.156
GEOM80b	80	743	15	0.172
GEOM90	90	531	10	0.125
GEOM90a	90	879	16	0.234
GEOM90b	90	950	18	0.219

P. Book Graphs Results

Book graphs are created where each node represents a character. Two nodes are connected by an edge if the corresponding characters encounter each other in the book. Proposed algorithm test result of book graphs are shown in Table 16.

TABLE XVII. BOOK GRAPHS RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
anna	138	986	12	0.202
david	87	812	12	0.204
huck	74	602	11	0.109
jean	80	508	10	0.078

Q. Game graph results

Game graph representing the games played in a college football season can be represented by a graph where the nodes represent each college team. Two teams are connected by an edge if they played each other during the season. Test results of the game graph are shown in Table 17.

TABLE XVIII. GAME GRAPH RESULTS

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
games120	120	1276	9	0.281

VI. RESULT ANALYSIS

In this section certain facts are extracted from the test results of section 5. The time complexity of proposed algorithm is also compared with some well known graph colouring algorithms.

Proposed edge cover based graph colouring algorithm is tested on many large graphs. Table 18 shows graph instances with their execution time (in Seconds) and a number of colours required to colour graphs.

TABLE XIX. LARGE GRAPH INSTANCES

Instance	Vertices (V)	Edges (E)	Colours (K)	Time (s)
C2000.5	2000	999836	239	19091.7
qg.order100	10000	990000	128	20540.531
DSJC1000.9	1000	449449	307	4025.27
latin_square_10	900	307350	152	1095.714
wap03a	4730	286722	86	1100.153
wap04a	5231	294902	70	1158.958
DSJC1000.5	1000	249826	127	684.343
qg.order60	3600	212400	82	978.151
DSJC500.9	500	224874	168	209.882
wap02a	2464	111742	59	206.283
wap01a	2368	110871	59	188.199
wap08a	1870	104176	68	150.603
wap07a	1809	103368	65	149.708
DSJR500.1c	500	121275	103	102.53
DSJR500.5	500	58862	197	98.664
qg.order40	1600	62400	60	96.171

Implementation results of proposed edge cover based algorithm are compared with a well-known Ant-based algorithm for colouring graphs (ABAC) [13]. Table 19 shows the comparison results of both algorithms. The table also shows the results chromatic number (K) of both algorithms.

TABLE XX. COMPARISON OF PROPOSED ALGORITHM AND ANT-BASED ALGORITHM (ABCA)

Instance	Proposed		ABCA	
	K	*Time (s)*	*K*	*Time (s)*
2-Insertions_3	4	0.016	4	0.02
3-Insertions_3	4	0.031	4	0.07
1-Insertions_4	5	0.063	5	0.1
4-Insertions_3	4	0.078	4	0.17
mug88_25	4	0.078	4	0.16
mug88_1	5	0.062	4	0.17
1-FullIns_4	5	0.14	5	0.31
myciel6	7	0.188	7	0.56
mug100_25	4	0.125	4	0.35
mug100_1	4	0.078	4	0.25
4-FullIns_3	8	0.187	7	0.73
miles250	10	0.18	8	0.57
miles500	26	0.422	20	1.53
miles750	39	0.953	31	1.95
2-Insertions_4	5	0.161	5	0.74
5-FullIns_3	8	0.188	8	1.38
myciel7	10	0.422	8	2.49
1-Insertions_5	6	0.219	6	1.64
2-FullIns_4	6	0.252	6	2.03
3-Insertions_4	5	0.219	5	4.69
4-Insertions_4	5	0.406	5	12.9
2-Insertions_5	8	0.75	6	17.82
1-Insertions_6	7	0.953	7	18.6
4-FullIns_4	9	0.985	8	22.53
2-FullIns_5	8	1.484	7	29
5-FullIns_4	10	1.422	9	33.5
3-Insertions_5	7	1.858	6	36.68

Figure 2 shows the execution time of proposed and ABCA algorithm for different size of graphs. X axis is representing a number of vertices in graph and Y axis is representing execution time in seconds of the algorithm. Figure 2 is generated by the data available in Table 19. Figure 2 clearly shows that execution time of proposed algorithm is less then ABCA algorithm, especially for the large graphs.

Table 20 present the comparison of execution time (in seconds) and a chromatic number of proposed algorithm and Genetic algorithm with multipoint guided mutation algorithm (MSPGCA) [8].

Figure 3 generated from graph instances their execution time available in Table 20. It has been observed that proposed algorithm execution completed in optimum time.

In Table 21 Parallel genetic algorithm based on CUDA (PGACUDA) [13] is compared with proposed algorithm. Figure 4 shows execution time behaviour of both algorithms. By Figure 4 it is clear that for the larger graphs execution time of proposed algorithm is optimum compared to PGACUDA.

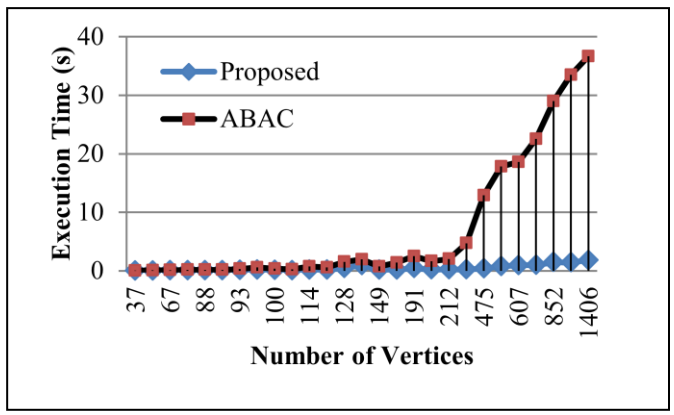

Fig. 2. Execution time comparison of proposed algorithm and ABAC algorithm

TABLE XXI. COMPARISON OF PROPOSED AND GENETIC ALGORITHM WITH MULTIPOINT GUIDED MUTATION ALGORITHM (MSPGCA)

Instance	Proposed		MSPGCA	
	K	*Time (s)*	*K*	*Time (s)*
mug88_25	4	0.08	4	15
myciel6	7	0.19	7	4
mug100_25	4	0.13	4	18
4-FullIns_3	8	0.19	7	2
miles750	39	0.95	31	69
2-Insertions_4	5	0.16	5	3
5-FullIns_3	8	0.19	8	3
myciel7	10	0.42	8	3

1-Insertions_5	6	0.22	5	148
2-FullIns_4	6	0.25	6	96
3-Insertions_4	5	0.22	5	6
4-Insertions_4	5	0.41	5	1071
2-FullIns_5	8	1.48	7	450

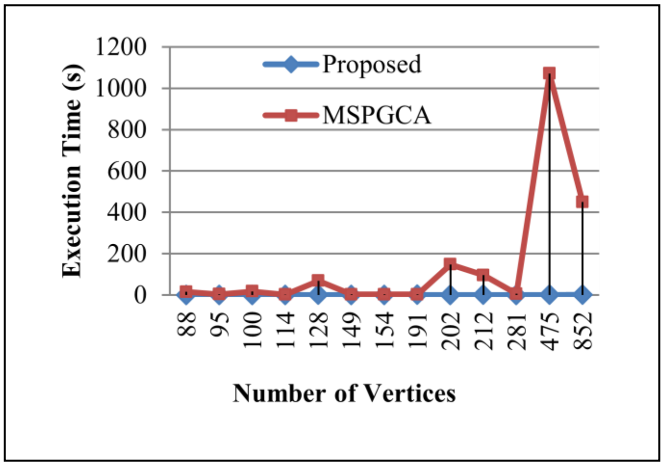

Fig. 3. Execution time comparison of proposed algorithm and MSPGCA algorithm

TABLE XXII. Comparison of Proposed and Parallel Genetic Algorithm Based on CUDA (PGACUDA)

Instance	Proposed		PGACUDA	
	K	*Time (s)*	*K*	*Time (s)*
2-Insertions_3	4	0.02	4	0.018
3-Insertions_3	4	0.03	4	0.043
1-Insertions_4	5	0.06	5	0.029
4-Insertions_3	4	0.08	4	0.013
mug88_25	4	0.08	4	0.063
mug88_1	5	0.06	4	0.059
1-FullIns_4	5	0.14	5	0.053
myciel6	7	0.19	7	0.174
mug100_25	4	0.13	4	0.084
mug100_1	4	0.08	4	0.085
4-FullIns_3	8	0.19	7	0.133
miles250	10	0.18	8	0.174
miles500	26	0.42	20	0.591
miles750	39	0.95	31	1.207
2-Insertions_4	5	0.16	5	0.151
5-FullIns_3	8	0.19	8	0.137
myciel7	10	0.42	8	0.496
1-Insertions_5	6	0.22	6	0.365
2-FullIns_4	6	0.25	6	0.313
3-Insertions_4	5	0.22	5	0.316
4-Insertions_4	5	0.41	5	0.947
2-Insertions_5	8	0.75	6	2.225
1-Insertions_6	7	0.95	7	3.495
4-FullIns_4	9	0.99	8	4.948
2-FullIns_5	8	1.48	7	8.475
5-FullIns_4	10	1.42	9	14.925
3-Insertions_5	7	1.86	6	20.419

Modified cuckoo optimisation algorithm (MCOACOL) [4] is modified algorithm of the cuckoo optimisation algorithm for graph colouring algorithm. Cuckoo optimisation well knows graph colouring algorithm based on cuckoo bard's behaviour. This paper also compared the results of MCOACOL algorithm to proposed algorithm results. Table 22 has the comparison proposed and MCOACOL algorithm. To analyse the Figure 5 it has been observed that time complexity of proposed algorithm is better than MCOACOL. The time complexity of proposed algorithm is highly expectable for the large graphs.

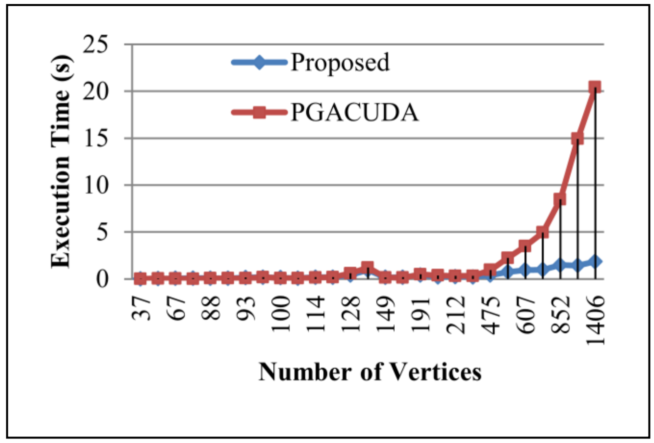

Fig. 4. Execution time comparison of proposed algorithm and PGACUDA algorithm

TABLE XXIII. Comparison of Proposed and Modified Cuckoo Opptimixation Algorithm (MCOACOL)

Instance	Proposed		MCOACOL	
	K	*Time (s)*	*K*	*Time (s)*
2-Insertions_3	4	0.02	4	0.4
3-Insertions_3	4	0.03	4	0.5
1-Insertions_4	5	0.06	5	0.5
4-Insertions_3	4	0.08	4	0.6
mug88_25	4	0.08	4	1.3
mug88_1	5	0.06	4	1.1
1-FullIns_4	5	0.14	5	0.5
myciel6	7	0.19	7	0.5
mug100_25	4	0.13	4	0.5
mug100_1	4	0.08	4	0.8
4-FullIns_3	8	0.19	7	0.7
miles250	10	0.18	8	1.1
miles500	26	0.42	20	1.2
miles750	39	0.95	31	1.5
2-Insertions_4	5	0.16	5	1.1
5-FullIns_3	8	0.19	9	0.5
myciel7	10	0.42	8	3.8
1-Insertions_5	6	0.22	6	1.2
2-FullIns_4	6	0.25	6	1.2
3-Insertions_4	5	0.22	5	2.1
4-Insertions_4	5	0.41	5	3.7
2-Insertions_5	8	0.75	6	6.5
1-Insertions_6	7	0.95	7	8.1
4-FullIns_4	9	0.99	8	7.7
2-FullIns_5	8	1.48	7	10.7
5-FullIns_4	10	1.42	9	28
3-Insertions_5	7	1.86	6	45

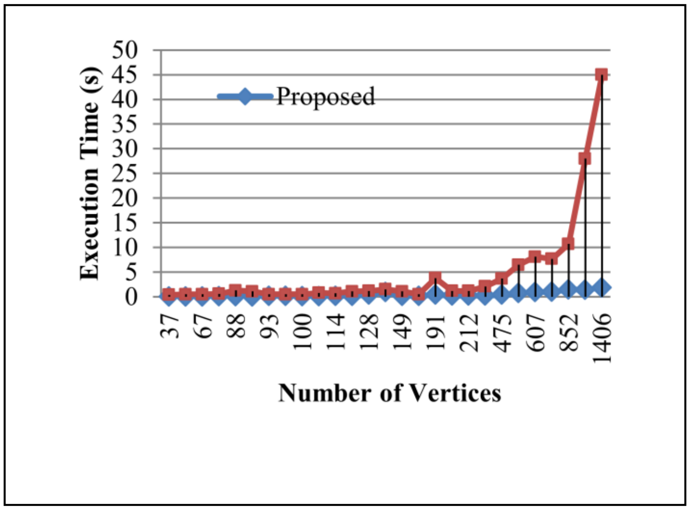

Fig. 5. Execution time comparison of proposed algorithm and MCOACOL algorithm

VII. Conclusion and Future Scope

Proposed edge cover based graph colouring algorithm is an exact graph colouring algorithm to solve the graph colouring problem. The algorithm is tested and evaluated on various categories of DIMACS graph instances. Results are also compared with some well-known graph colouring algorithms. Proposed edge cover based graph colouring algorithm is suitable for all size of graphs. Execution success rate is high of proposed algorithm. Execution time is optimum for large graphs. Proposed algorithm generates an optimum chromatic number for small and medium size graphs.

There are certain areas of an algorithm, like calculating the degree of vertices and calculating edge sets in iterations. Parallel execution can be applied to make algorithm more time efficient. The algorithm can also enhance to get the more optimum chromatic number for large graphs by adding some more iteration.

References

[1] B. Hussin, A. S. H. Basari, A. S. Shibghatullah, and S. A. Asmai, “Exam timetabling using graph colouring approach”, IEEE Conference on Open Systems, Langkawi, 25-28, pp.139-144, September, 2011.

[2] A. Gupta, and H. Patidar, “A survey on heuristic graph coloring algorithm”, International Journal for Scientific Research & Development, vol. 4, issue 04, pp. 297-301, 2016.

[3] S.P. Tiwari, K. K. Bansal, and T. Chauhan, “Survey paper on solving graph coloring problem”, International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, issue 2, pp. 29-31, February, 2014.

[4] S. Mahmoudia, S. Lotfiba, “Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem”, Elsevier, Applied Soft Computing 33, pp. 48–64, 2015.

[5] P. C. Sharma and N. S. Chaudhari, ”Polynomial 3-SAT encoding for K-colorability of graph", International Journal of Computer Applications, pp. 19-23, 2014.

[6] E. Salari and K. Eshghi, “An ACO algorithm for the graph coloring problem”, International Journal Contemp. Math Sciences, vol. 3, no.6, pp. 293-304, 2008.

[7] Z. Lü, and Jin-Kao Hao, “A memetic algorithm for graph coloring”, Elsevier, European Journal of Operational Research 203, pp. 241–250, 2010.

[8] B. Ray, A. J Pal, D. Bhattacharyya, and Tai-hoon Kim, “An efficient GA with multipoint guided mutation for graph coloring problems”, International Journal of Signal Processing, Image Processing and Pattern Recognition, vol. 3, No. 2, pp. 51-58, June, 2010.

[9] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, “Ordering heuristics for parallel graph coloring”, Report, National Science Foundation ACM, pp.166-177, 2014.

[10] G. M. Slota, S. Rajamanickam, and K. Madduri, “BFS and coloring-based parallel algorithms for strongly connected components and related problems”, IEEE 28th International Parallel & Distributed Processing Symposium, pp. 550-559, 2014.

[11] E. G. Boman, D. Bozda, U. Catalyurek, A. H. Gebremedhin, and F. Manne, “A scalable parallel graph coloring algorithm for distributed memory computers”, Lecture note in Computer Science 3648, pp. 241-251, August, 2005.

[12] B. Chen, Bo Chen, H. Liu, X. Zhang,” A fast parallel genetic algorithm for graph coloring problem based on CUDA”, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, China, pp. 145-148, 2015.

[13] T.N. Bui, T.H. Nguyen, C.M. Patel, K.-A.T. Phan, “An ant-based algorithm for coloring graphs”, Discrete Appl. Math. 156, pp. 190–200, 2008.