An Efficient GA with Multipoint Guided Mutation for Graph Coloring Problems

Biman Ray[1], Anindya J Pal[1], Debnath Bhattacharyya[2], and Tai-hoon Kim[2,*]
[1]*Heritage Institute of Technology*
Kolkata 700107, India
bimray@gmail.com, anindyajp@yahoo.com
[2]*Department of Multimedia*
Hannam University
Daejeon, Korea
debnathb@gmail.com, taihoonn@empal.com

Abstract

Proper coloring of the vertices of a graph with minimum number of colors has always been of great interest of researchers in the field of soft computing. Genetic Algorithm (GA) and its application as the solution method to the Graph Coloring problem have been appreciated and worked upon by the scientists almost for the last two decades. Various genetic operators such as crossover and mutation have been used in the GA probabilistically in the previous works, which distributes the promising solutions in the search space at each generation. This paper introduces a new operator, called double point Guided Mutation operator with a special feature. An evolutionary algorithm with double point Guided Mutation for the Graph Coloring problem is proposed here, which could advance the performance level of simple GA dramatically.

The algorithm has been tested upon a large-scale test graphs and has shown better output than the earlier works on the same problem. This paper describes the advancement of performance of simple GA applied upon the problem of graph coloring using a operator called double point Guided Mutation in association of the general genetic operators Crossover and Mutation used probabilistically. Our work is still going on for designing better algorithms.

Keywords: *GA, Guided Mutation, MSPGCA*

1. Introduction

Let G = (V, E) be an undirected graph. A k-coloring of G is a partition of V into k subsets *Ci, i = 1 ..... k,* such that no adjacent nodes belong to the same subset. The graph coloring problem is to find a k-coloring of G with k as small as possible. This smallest k corresponds to the chromatic number *X(G)* of graph G. It is well known that this problem is NP-hard [11], and consequently, heuristic methods must be used for large graphs. If two adjacent vertices x and y have the same color r, vertices x and y are called conflicting vertices, the edge {x, y} is called a conflicting edge, and r is called a conflicting color. If there is no conflicting edge, then the color classes are called stable sets, and the k-coloring is said legal. The graph-coloring problem (GCP) is to determine the smallest integer k such that there exists a legal k-coloring of G. The reasons why the graph-coloring problem is important are twofold. First, there are several areas of practical interest, in which the ability to color an undirected graph with a small number of colors as possible, has direct influence on how efficiently a certain target problem can be solved. Such areas include, timetable scheduling [1], examination scheduling [2], register allocation [3], printed circuit testing [4], electronic bandwidth allocation

*Corresponding Author

[5], microcode optimization [6], channel routing [7], the design and operation of flexible manufacturing systems [8], computation of sparse Jacobian elements by finite differencing in mathematical programming [9], etc. The other reason is that, the graph coloring problem has been shown to be computationally hard at a variety of levels: not only its decision problem variant is NP complete [10], but also its approximate version is NP-hard [11]. These two reasons are important enough to justify the quest for heuristics to solve graph-coloring problem. A variety of heuristics approaches have been proposed to produce optimal or near optimal colorings in a reasonable amount of time.

2. Heuristics and evolutionary algorithms for graph coloring

A well known simple heuristic method is the Genetic Algorithm. It has been being used for coloration of graphs. In the context of genetic algorithms, cycles are referred to as *generations*, the so-called *crossover operators* achieve the cooperation step and the self-adaptation step consists of what is known as *mutation*. Costa, Hertz and Dubuis [12] describe a procedure that combines a simple descent method to achieve self adaptation within the general framework of a genetic algorithm. Hertz and De Werra [13] defined the descent method, which is based on moving from a solution to a neighbor solution. The objective function is modified to assigned weights to edges. The weights are changed from one generation to the next to avoid always manipulating the same conflicting edges. The mutation operator consists of replacing, with a given probability, a solution by a randomly chosen neighbor. The mutation probability is changed during the search, using a systematic scheme. A union crossover, originally designed by Costa for a scheduling application, was adapted for graph coloring to implement the cooperation step. Evolutionary algorithms have also been adapted in the context of graph coloring. Evolutionary methods operate on a population of solutions and seek improved outcomes by a sequence of *cycles* consisting of a cooperation step and a self-adaptation step. In the cooperation step, solutions in the current population exchange information with the goal of producing new solutions that inherit good attributes. In the self-adaptation step, solutions modify their internal structure without interacting with other solutions in the population. Fleurent and Ferland [14] proposed another evolutionary method for graph coloring. This implementation uses a graph-adapted recombination operator for the cooperation step. It also employs an entropy measure to evaluate the diversification of the solutions in the population. If the entropy is zero, then all the solutions in the population are identical. This measure is used both to design stopping rules and to influence parent selection. Another important feature is that members of the population are subject to local search. A variant of the Tabu-search scheme is used as one of the self-adaptation steps. In another evolutionary approach, Eiben, et al. [15] employs the 3-coloring problem to test several variants of an asexual evolutionary algorithm. The algorithm uses an order-based representation and an adaptation mechanism. Some other genetic algorithm heuristic method [16] has been developed later on. Neural networks have also been applied to the graph coloring problem. This effort was started more than 10 years ago by Dahl [17], and more recently continued by Jagota [18]. Neural network applications in this context are based on mapping the k-coloring problem to a Hopfield network. This process is accomplished by first considering a reduction to the maximum independent set (MIS) problem, followed by a mapping of MIS onto a Hopfield network. Jagota follows the

approach of choosing an initial k value and gradually decreasing it in an attempt to find improved feasible colorings. If the algorithm fails to find a feasible coloring in one phase, the current k value is increased and the process continues. Since k is initially set to a sufficiently large value, the procedure is guaranteed to yield a proper coloring. The procedure was tested on a set of 30 graphs associated with the Second DIMACS Challenge. The results were compared with the parallel procedure Hybrid of Lewandowski and Condon [19]. Jagota's implementation is outperformed by Hybrid in all but 6 instances. One such approach is Ant colony optimization [20] [21]. We have provided a partial review of the graph coloring literature. For more information on the graph coloring problem and a more comprehensive bibliography, we refer Michael Trick's "Network Resources for Coloring a Graph" (http://mat.gsia.cmu.edu/COLOR/color.html), Joe Culberson's "Graph Coloring Page" (http://web.cs.ualberta.ca/~joe/Coloring/index.html).

3. Evolutionary Algorithm for graph coloring

The population-based approach of GA allows large jumps in the search space. However GAs have not proved successful for graph coloring because of the large degree of symmetry of the solution space. In fact, because of this symmetry mismatch, it is very unlikely to produce a fit offspring when combining two good solutions. Thus GAs are often considered an inappropriate approach for problems such as graph coloring with a highly degenerate objective function. In order to compensate for this degeneracy advanced search techniques need to be applied. This paper proposes an evolutionary algorithm for graph coloring problem. The two main components of this algorithm are as follows:

- Heuristic search
- Evolutionary algorithm

Here we have applied simple genetic algorithm (GAGCA) as heuristic search technique to generate population better than the initial population. The evolutionary algorithm may be genetic algorithm, ant colony optimization etc. Here we have considered the genetic algorithm with a multi-point special mutation operator. This section describes the main components of the multi-point special mutation algorithm for graph coloring. Integer strings encode the chromosomes. For each generation blind crossover operator (like standard GA), repair operator (to convert from invalid to valid chromosome) and a multi-point special mutation operator (in place of random mutation) are applied.

3.1 Representation and fitness

In our algorithm, we represent the chromosomes as a set of integers (1, 2, 3….n). These integers are nothing but the color of the nodes. The positions of these integers are the node numbers for which those particular colors have been assigned. An example of a chromosome for a graph of 5 vertices is shown below:

1 4 3 2 1

Here the nodes 1 and 5 have the same color 1. 2nd, 3rd and 4th nodes have the colors 4, 3 and 2 respectively. The fitness function is nothing but the total number of colors used, i.e., distinct integers in the chromosome.

3.2 Initial population

The algorithm first builds an initial population, by randomly assigning colors to different nodes. In this pool, some chromosomes will denote valid coloration and some of them are invalid. But we have not discarded these invalid colorations. Instead, we use a repair operator *rep_op*, which coverts the invalid chromosome to a valid one. In the pool, we have considered, one chromosome generated by the GAGCA. This *rep_op* checks whether two adjacent nodes have the same color or not. In that particular case, it replaces one such color by any randomly generated color.

3.3 Multi-point Special mutation

We have applied a multi-point special mutation operator to improve the fitness of the chromosomes of the pool. Here, we have named the guided mutation as special mutation. Finally, the improved chromosomes are considered for the next generation. This process repeats for a prefixed number of iterations.

mp_*sp_mutation*
Step 1) Choose a particular chromosome
Step 2) Reduce the multiple number of colors in
that chromosome (e.g. by replacing two of the used colors by other used colors at two places)
Step 3) If the coloration is invalid then apply the
rep_ op (here it'll try to repair the invalid coloration to valid one using the reduced number of colors available)
Step 4) If it succeeds or the coloration is valid then
place this chromosome in the pool
Else goto step 1)

3.4 Algorithm MSPGCA

Step 1) Create random initial pool of chromosomes
(population)
Step 2) Sort the chromosomes of the pool in
ascending order of their fitness value
Step 3) Supply the pool of chromosome to the
GAGCA and generate new population
Step 4) Apply the *rep_ op* to the invalid
chromosomes
Step 5) Calculate the fitness of the pool
Step 6) Pbest = best individual
Step 7) For generation = 1 to max_iteration
Step 7.1) Perform crossover between any
two random pair of
chromosomes with probability pc
Step 7.2) Apply the *rep_ op* to the invalid
offspring
Step 7.3) Select the best individuals from
the newly generated offspring
Step 7.4) Apply *mp_sp_mutation* to the
pool of chromosome for certain

no_of_iteration
Step 7.5) Calculate the fitness of the pool
Step 7.6) Take improved chromosomes for
next generation
Step 8) Output the best coloration

This evolutionary algorithm MSPGCA differs from a genetic algorithm by a special feature. The feature is that, in place of conventional random mutation here a multi-point mutation is applied which is nothing but a forceful decrement in number of colors. The crossover operator remains same as the standard genetic algorithm's blind crossover operator.

3.5 Complexity

The evolutionary algorithm based approaches have time complexity that depends on two fold analytic approach. In our case the initial population generation needs O(n) time complexity, where n is the number of nodes in a test graph. The second step is somewhat open ended. Genetic algorithms are most suitable for MIMD parallel computers and distributed computing systems (including heterogeneous systems) as those composed by networks of workstations. The analysis partly depends on the number of iterations at various levels and practical implementation of parallel genetic algorithm. We are handicapped to test our algorithms on P-IV based sequential machines. The exact time taken by the CPU is given and shows reasonable in time and space.

4. Computational Experiments

In this section, we present experimental results obtained by MSPGCA and make comparisons with other component algorithm (GAGCA). GAGCA is the conventional genetic algorithm by which we have generated a population after applying it on the initial population. In this section we present the results of our algorithm on some benchmark graphs given at http://mat.gsia.cmu.edu/COLOR04/. MSPGCA has been implemented in ANSI C and run on a Xeon 2.4GHz machine with 1GB of RAM running the Linux operating system. We have considered the population size as 10. We have considered the crossover probability pc = 0.1. The no_of_iteration for *mp_sp_mutation* was set to as large as the number of vertices in the graph after trial and error. The value of max_iteration depends on the density and number of nodes of the graph. On an average, for the graphs with less than 100 nodes and density less than 0.5 the value of max_iteration is order of 10. For other graphs it may vary from 20 – 100. For some instances (the graphs of queenm_n series, some other series), GAGCA fail to produce the optimal result, but MSPGCA gives the result. Table-1 shows the experimental results of the two algorithms and the time taken by MSPGCA.

5. Conclusions

We have presented experimental results on some of the DIMACS benchmark graphs and compared our algorithm's performance with that of the other heuristic (Genetic Algorithm GAGCA) on those graphs as well. Indeed, our MSPGCA able to find the best-known results for most of the tested graphs. To strengthen conclusions made about the power of the algorithm, it is worth to test it on some other classes of large graphs.

The main purpose of this paper is to study evolutionary algorithm on graph coloring problem. In the future, we intend to refine MSPGCA and apply it to color large complex graphs in reasonable time.

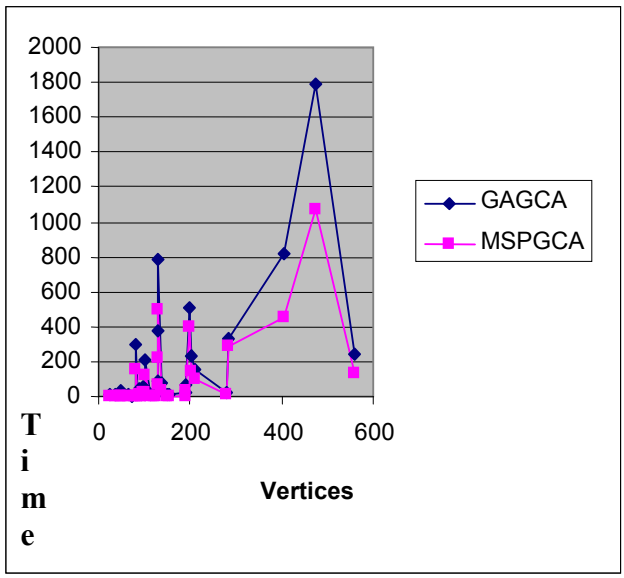

Figure 1. Comparison of performances of GAGCA and MSPGCA

Table1. Comparison results between GAGCA and MSPGCA

Instances	|V|	|E|	X(G)	GAGCA	MSPGCA	Time of MSPGCA(S)
1-FullIns 5.col.b	282	3247	6	7	6	287
1-Insertions5.col.b	202	1227	6	7	5	148
2-FullIns 4.col.b	212	1621	6	8	6	96
2-Insertions4.col.b	149	541	5	5	5	3
3-FullIns 4.col.b	405	3524	7	8	7	450
3-Insertions 4.col.b	281	1046	5	7	5	6
4-FullIns 3.col.b	114	541	7	9	7	2
4-Insertions 4.col.b	475	1795	5	8	5	1071
5-FullIns 3.col.b	154	792	8	9	8	3
anna.col.b	138	493	11	11	11	31
david.col.b	87	406	11	11	11	1
games120.col.b	120	638	9	9	9	1
homer.col.b	561	1628	13	15	13	135
huck.col.b	74	301	11	11	11	3
jean.col.b	80	254	10	11	10	2
miles1000.col.b	128	3216	42	45	42	496
miles1500.col.b	128	5198	73	73	73	217

miles750.col.b	128	2113	31	34	31	69
mug100 25.col.b	100	166	4	5	4	18
mug88 25.col.b	88	146	4	4	4	15
mulsol.i.1.col.b	197	3925	49	52	49	393
myciel5g_col	47	236	6	7	6	1
myciel5gb_col	47	236	6	6	6	1
Instances	**|V|**	**|E|**	***X*(G)**	**GAGCA**	**MSPGCA**	**Time of MSPGCA(S)**
myciel6g_col	95	755	7	10	7	4
myciel6gb_col	95	755	7	13	7	5
myciel7g_col	191	2360	8	32	8	3
myciel7gb_col	191	2360	8	37	8	37
queen5_5.col.b	25	320	5	6	5	1
queen6_6.col.b	36	580	7	9	8	3
queen7_7.col.b	49	952	7	12	7	3
queen8_8.col.b	64	1456	9	15	11	3
queen8_12.col.b	96	2736	12	23	14	22
queen9_9.col.b	81	2112	10	19	10	158
queen10_10.col.b	100	2940	11	20	14	125
DSJC125.1.col.b	125	736	5	8	6	4

6. References

[1] D. C. Wood, "A technique for coloring a graph applicable to large scale time-tabling problems", Computer Journal, vol. 12, pp. 317-319, 1969.

[2] F. T. Leighton, "A graph coloring algorithm for large scheduling problems", Journal of Research of the National Bureau of Standards, vol. 84, no. 6, pp. 489-505, 1979.

[3] F. C. Chow and J. L. Hennessy, "Register allocation by priority based coloring", Proceedings of the ACM Sigplan 84 symposium on compiler construction New York, pp. 222-232, 1984.

[4] M. R. Garey, D.S. Johnson and H.C. So., "An application of graph coloring to printed circuit testing", IEEE Transactions on circuits and systems, vol. 23, pp. 591-599, 1976.

[5] A. Gamst, "Some lower bounds for class of frequency assignment problems", IEEE Transactions on Vehicular Technology, vol. 35, no. 1, pp. 8-14, 1986.

[6] Micheli G. D., Synthesis and Optimization of Digital Circuits. McGraw-Hill, 1994.

[7] S. S. Sarma, R. Mondal and A. Seth, "Some sequential graph coloring algorithms for restricted channel routing", INT. J. Electronics, vol. 77, no. 1, pp. 81-93, 1985.

[8] K. Stecke, "Design, planning, scheduling and control problems of flexible manufacturing". Annals of Operations Research, vol. 3, pp. 187-209, 1985.

[9] T. F. Coleman and J. J. More, "Estimation of sparse Jacobian Matrices and graph coloring problems", SIAM. J. Numer. Anal., vol. 20, pp 187-209, 1983.

[10] Garey, M.R. and D. S. Johnson, Computers And Intractability: A Guide To The Theory of NP Completeness. New York, W. H. Freeman and Co., 1979.

[11] Baase, S. and A. V. Gelder, Computer Algorithms: Introduction To Design and Analysis. Addison-Wesley, 1999.

[12] D. Costa, A. Hertz and O. Dubuis, "Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs", Journal of Heuristics, vol. 1, no. 1, pp. 105-128, 1995.

[13] A. Hertz and D. Werra, "Using tabu search techniques for graph coloring", Computing, vol. 39, no. 4, pp. 345-351, 1988.

[14] C. Fleurent and J.A. Ferland, "Genetic and hybrid algorithms for graph coloring", Annals of Operations Research, vol. 63, pp. 437-464, 1996.

[15] A. E. Eiben, J. K. Hauw and J. I. Hemert "Graph coloring with adaptive evolutionary algorithms", Technical Report, TR-96-1, Leiden University, Netherlands, 1997.

[16] C. Coritoru, H. Luchian, O. Gheorghies and A. Apetrei, "A New Genetic Graph Coloring Heuristic", Computational Symposium on Graph Coloring and its generalizations, COLOR02, Cornell University, September 2002.

[17] E. D. Dahl, "Neural Networks algorithms for an NP complete problem: Map and graph coloring", IEEE International Conference on Neural Networks, vol. 3, pp. 113-120.

[18] A. Jagota, "An adaptive, multiple restarts neural network algorithm for graph coloring", European Journal of Operational Research, vol. 93, pp. 257-270, 1996.

[19] G. Lewandowski and A. Condon, "Experiments with parallel graph coloring heuristics", Technical Report 1213, University of Wisconsin, Madison, 1993.

[20] E. Bonabeau, M. Dorigo and G. Theraulaz, "Inspiration for Optimization from Social Insect Behavior", Nature, Vol. 406, pp. 39–42, July 2000.

[21] T. N. Bui and C. Patel, "An Ant system Algorithm for Coloring Graphs", Computational Symposium on Graph Coloring and its Generalizations, COLOR02, Cornell University, September 2002.