**Proceedings of the 17th IASTED International Conference**
**MODELLING AND SIMULATION**
**May 24-26, 2006, Montreal, QC, Canada**

# A METHOD FOR THE MINIMUM COLORING PROBLEM USING GENETIC ALGORITHMS

Haidar Harmanani and Hani Abas
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

**ABSTRACT**

This paper presents a method to solve the graph coloring problem for arbitrary graphs using genetic algorithms. The graph coloring problem, an NP-hard problem, has important applications in many areas including time tabling and scheduling, frequency assignment, and register allocation. The algorithm was implemented and tested on various set of instances including large DIMACS challenge benchmark graphs, all yielding favorable results.

**KEY WORDS:** Genetic algorithms, combinatorial optimization, graph coloring.

## 1 Introduction

The *graph coloring problem* is an important problem and has wide applications in contemporary problems with exponential complexity. The graph coloring problem has many possible applications including the time-tabling problem [11, 12], the register allocation problem [2], and the digital circuit testing problem [6]. Other applications where the graph coloring problem has proven to be useful are digital circuits synthesis and logic optimization [7].

The graph coloring problem is NP-complete and determines the minimum number of colors needed to color a given graph. Let $G = (V, E)$ be an undirected graph where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices in $G$, and $E \subseteq V \times V$ is the set of edges in $G$. Let $k$ be a positive integer such as $k \leq |V|$. A $k-$coloring of $G$ is a function $c : V \rightarrow \{1, 2, ..., k\}$ such that $c(u) \neq c(v)$ whenever $\{u, v\} \in E$. In other words, the numbers $1, 2, ..., k$ represent $k$ colors, and adjacent vertices must have different colors. If there exists such a coloring of $G$ that uses no more $k$ colors, $G$ admits a $k-$coloring ($G$ is $k-$colorable). The minimal $k$ for which $G$ admits a $k-$coloring is called the chromatic number and is denoted by $\chi(G)$. Garey and Johnson [5] provided an extensive study for the graph coloring problem indicating that it was solvable in polynomial time for $k = 2$ but remains NP-complete for all fixed $k \geq 3$ and, for $k = 3$, for planar graphs having no vertex degree exceeding 4. For an arbitrary $k$, the problem is NP-complete for circle graphs and circular arc graphs (even their representation as families of arcs), although for circular arc graphs the problem is solvable in polynomial time for any fixed $k$. The general problem can be solved in polynomial time for comparability graphs, for chordal graphs, for (3, 1) graphs, and for graphs having no vertex degree exceeding $three$.

Several approximations and search algorithms were developed to solve the *graph coloring problem*. The most common approximation algorithm is that of successive augmentation. In this approach a partial coloring is found on a small number of vertices and this is extended vertex by vertex until the entire graph is colored [12, 11, 9]. Some approaches allow some uphill moves during the search process using simulated annealing [3] or tabu-search [8]. Johnson et al. [10] formulated the graph coloring problem using three simulated annealing techniques and an augmentation algorithm. Davis et al. [4] proposed a genetic algorithm solution; however, the algorithm gave poor results. Kirovski et al. [13] proposed a graph coloring algorithm that uses two distinct algorithms and a their hybrid. The first is based on successive augmentation while the second is based on a lottery-scheduling driven iterative improvement. The hybrid uses initially the augmentation-based algorithm in order to reduce the search space before proceeding into the lottery-based algorithm.

This paper presents a genetic algorithm to solve the *graph coloring problem* (chromatic number), an NP-complete problem, for arbitrary graphs. The algorithm explores the search space by iteratively decreasing the number of colors and reducing coloring conflicts.

The remainder of the paper is organized as follows. In section 2 we give some background on genetic algorithms. In section 3 we formulate the graph coloring problem using our genetic algorithm and describe our chromosomal representation, genetic operators, and our genetic algorithm. Finally, experimental results are presented in section 4. We conclude with remarks in section 5.

## 2 Genetic Algorithms

Genetic algorithms [15, 16] (GA) is a stochastic combinatorial optimization technique that mimics some principles of natural evolution in order to solve optimization prob-

*This work was supported in part by the Lebanese American University under grant URC-t2006-04

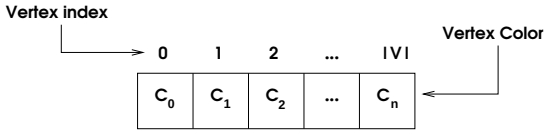


Figure 1. Chromosomal representation

lems of high complexity. A group of randomly initialized points of the search space (individuals) is used to search the problem space. Each individual encodes all necessary problem parameters (genes) as bit strings, vectors, or graphs. Each gene represents one of the parameters to be optimized. The population evolves following a parody of Darwinian principle of the survival of the fittest. Individuals are selected according to their quality, measured by a fitness function, to produce offsprings and to propagate their genetic material into the next generation. Each offspring undergoes a sequence of operators (such as mutation, inversion, crossover) with a certain probability to provide diversity of the population avoiding premature convergence to a single local optimum. The iterative process of selection and combination of "good" individuals should yield even better ones, until a solution is found or a certain stop criterion is met. The main advantage of using a genetic algorithm is that it only needs an objective function with no specific knowledge about the problem space. The challenge, however, remains in finding an appropriate problem representation that results in an efficient and successful implementation of the algorithm.

# 3 Problem Formulation

The graph-coloring problem is to determine the minimum number of colors needed to color a given graph and is formally defined as follows [5]:

> **Instance**: Graph *G = (V, E)* and a positive integer $K \leq |V|$
>
> **Question**: Is G K-colorable, i.e., does there exist a function $f : V \rightarrow \{1, 2, ..., K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

In what follows, we describe our formulation for the graph coloring problem using genetic algorithms including our chromosomal representation, genetic operators, and cost function.

## 3.1 Chromosomal Representation

In order to solve the graph coloring problem, we propose the simple chromosomal representation that is shown in Figure 1. A chromosome encodes a solution using a simple vector whose length is equal to the number of vertices in the graph. Every gene or its position in the graph corresponds to a vertex color. Figure 2 shows a simple graph along with the corresponding chromosome.

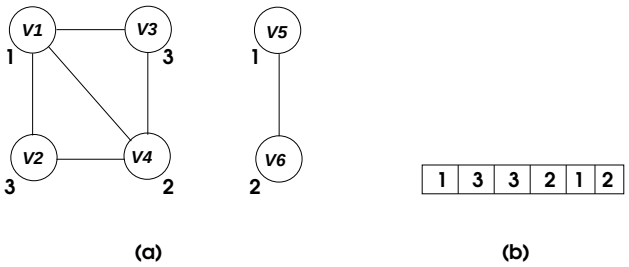


Figure 2. (a) Simple graph and a possible coloring, (c) Corresponding chromosome representation

## 3.2 Initial Population

The initial population is very important in determining the final solution. We generate the initial population through a random perturbation that assigns $|V|$ random colors to the genes of the chromosomes. Thus, this would result with an initial population of size $N_g$ where every vertex is assigned a unique color.

## 3.3 Genetic Operators

In order to explore the solution space, we use two genetic operators, *crossover* and *mutation*. The operators are applied iteratively in every generation with their corresponding probabilities.

### 3.3.1 Crossover

The crossover operator is important to overcome information loss that occurs in GA. Typically, genetic operators, especially crossover, perform better if they would incorporate specific domain knowledge.

In our algorithm, we use a one-point crossover that aims at partitioning the vertices into compatible sets in order to improve the coloring. This is done by mostly performing the crossover cuts at predetermined locations based on infeasibilities. If the chromosome is infeasible, the crossover operator locates the first gene that causes an infeasible coloring thus creating two sets, one with feasible coloring and one with infeasible coloring. The crossover operator takes place at this point in order to construct feasible coloring by minimizing the penalty. However, if the chromosome is feasible, then the crossover cut-point is selected randomly.

### 3.3.2 Mutation

The mutation operator is important as it explores the design space and injects diversity in the population. Thus, the operator helps the chromosome to move from one configuration to another.

The mutation operator is applied to feasible as well as to infeasible chromosomes. In the case of infeasible chro-

mosomes, the operator finds the first two adjacent nodes with the same color and the color of one of the two nodes is changed in order to locally repair the chromosome. The operator explores both options and the color of the node that leads to the least penalty is changed. This is done by locally exploring the chromosome at a specific node; the node's color that leads to a better cost is randomly mutated to a the other color. In the case of no conflicts in the chromosome, then the mutation operator decreases the number of colors by randomly eliminating one color.

### 3.4 Cost Function

The graph coloring problem leads to the following two optimization criteria:

1. One and only one color is to be assigned to each vertex.

2. No two adjacent vertices should have the same color.

In order to minimize the number of colors, we try to minimize the number of conflicts in the vertex coloring. Thus, every time we have a conflict in a chromosome, the chromosome is kept in the solution but a penalty is assigned. A feasible chromosome has a penalty of 0.

In order to efficiently assign the penalty in the graph, the chromosomes that have been changed are scanned in order to determine the least number of colors, $k$, that have been found by the algorithm in feasible chromosomes. The algorithm next assigns a penalty $n$ to every edge violation (two nodes that have the same color). The result of the fitness function is the sum of all penalties. In other words,

$$Fitness = \sum_{i=0}^{N}(K + \sum_{i=0, i\neq j}^{N} Penalty(i,j)) \qquad (1)$$

where $Penalty(i,j)$ is equal to $n$ if $i$ and $j$ are connected and 0 Otherwise.

### 3.5 Selection

As the genetic operators are applied, the population size will increase. In order to maintain the size of the population, a subset of the population is selected. The chromosomes are first sorted in increasing order of cost. The algorithm selects $M$ chromosomes from the population based on elitism and a stochastic approach. Thus, the first 50% of the new population in the current populated is kept. The other 50% is randomly selected from the remaining chromosomes. The reason why this is done so that the algorithm avoids saturation and ensures diversity in the population.

```
Genetic_GraphColoring()
{
   M = Population size.
   N_g = Number of generations
   N_0 = Number of offsprings = N_g / 2
   Get the population size and the nb. of generations (Ng)
   Generate an initial population, current_pop
   for i = 1 to M
     evaluate(current_pop)
   Keep_the_best()
   NumberColors = V
   for i = 0 to N_g do
   {
     if (BestFitness < NumberColors)
           Squeeze the colors set
     Divide Population into two halves, P_1 and P_2
     for (i = 1 to N_0)
           Select two chromosomes randomly from P_1
           apply crossover with probability P_c
           Add offspring to the population
     for (i = 1 to N_0)
           Select a chromosome from P_2
           apply mutation with probability P_m
           Add offspring to the population
     Evaluate the population fitness for P_i
     new_pop = select(current_pop, offspring)
     current_pop ← new_pop
   }
}
```

Figure 3. Genetic graph coloring algorithm

### 3.6 Genetic Coloring Algorithm

Every feasible chromosome in the population corresponds to a possible solution to the graph coloring problem. However, the population has in addition to the feasible chromosomes, unfeasible ones.

During every generation, the population is divided into two halves and a set of offsprings, $N_0$ are produced by applying each of the genetic operators, *mutation* and *crossover*. The crossover operator produces random offspring that have a mixture of parental properties while the mutation operator explores the design space and introduces new solutions as well as reduces the number of colors. Finally, sub-populations are combined and a fixed number of chromosomes is chosen.

Initially, every node in the graph is randomly assigned a distinct color resulting with $|V|$ coloring. As the algorithm evolves and since the algorithm does not know the chromatic number of the graph, $\chi(G)$, we incrementally $squeeze$ or reduce the number of colors every time a feasible coloring with $k$ colors is achieved. The algorithm stops either when the maximum time is exceeded or if the algorithm fails to improve the number of colors after some generations. The genetic graph coloring algorithm is shown in Figure 3.

| Parameter | Value |
|---|---|
| Crossover | 30% |
| Mutation | 80% |
| Population Size | 50 |
| Number of Generation | 200,000 |

Table 1. Parameter settings

## 4 Experimental Results

The proposed algorithm was implemented using the C language and tested on various benchmarks on a *Xeon 3.2 Ghz* workstation. The algorithm was tested on the following test instances:

- The first set of test instances are graphs from Donald Knuth's Stanford *GraphBase*. We attempted in this category *book graphs* where each node represents a character. Two nodes are connected if the corresponding characters encounter each other in the book. We use in this category three graphs generated based on *Anna karenina*, *David Copperfield*, and *Huckleberry Finn*. The other set of graphs in this category are the *queen graphs*. Given an $n \times n$ chessboard, a queen graph is a graph on $n^2$ nodes, each corresponding to a square on the board. Two nodes are connected by an edge if the corresponding squares are in the same row, column, or diagonal. Finally, we attempted graphs that are based on *Mycielski* transformation.

- The second set of test instances are graphs that are available from the center for Discrete Mathematics and Theoretical Computer Science via ftp [14]. These include two *Leighton* graphs which are random graphs with a fixed number of edges and predetermined chromatic number. We test our algorithm using *Leighton* graphs with 450 vertices and chromatic number 5 and 15. We also used graphs that are based on *register allocation* for variables in real codes in addition to *class scheduling* graphs with and without study halls. Finally, we attempted the *random graphs* proposed by Johnson et al. [9]. These are standard $(n, p)$ random graphs and include *geometric* as well as complements of geometric graphs.

Table 4 shows the experimental results for the above graphs. These graphs were largely studied in the literature and constitute a good reference for comparison. Moreover, these graphs are difficult to color and constitute a real challenge for graph coloring algorithms. The first column represents the benchmark file while the next three columns represent the graph's vertex and edge cardinalities, and the optimal coloring if known.

In order to solve a given $k-$coloring instance, we ran the genetic algorithm for $20 - 30$ times with each run

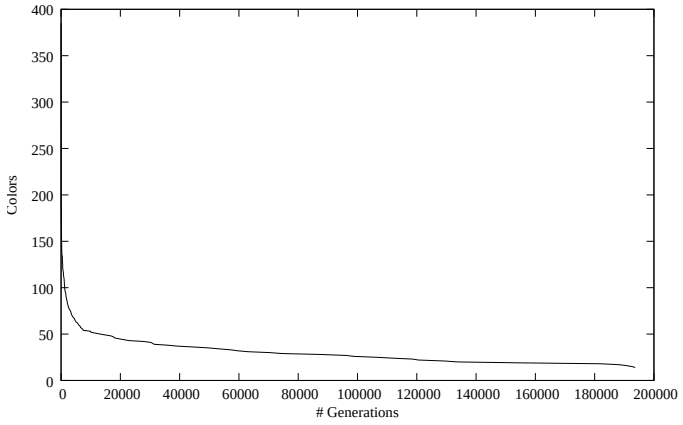


Figure 4. Algorithm convergence in the case of the *school1* course scheduling benchmark

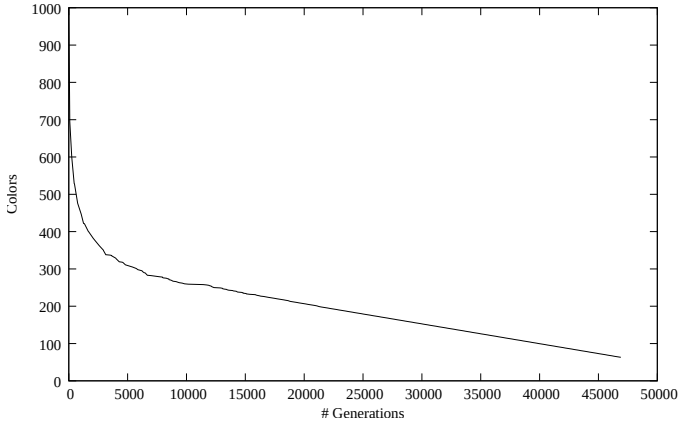


Figure 5. Algorithm convergence in the case of the *flat1000_50_0* random flat graph benchmark

given the same number of generations. The reported results are the averages over all runs. The algorithm was able to find the optimal coloring for the first set of test instances *(GraphBase)* and stopped before reaching the maximum number of generations.

For the second set which corresponds to real life examples, the algorithm was able to find the optimal coloring on all of the 5-colorable Leighton graphs and two of the four 15-colorable graphs. The algorithm also found the optimum coloring on four of the twelve geometric graphs. However, the algorithm was able to find only one optimal coloring for one of the flat graphs. Finally, the algorithm was able to find the optimal coloring for both course scheduling graphs.

It should be noted that the algorithm failed to find the optimal coloring in some difficult-to-color real-life optimizations problems especially when the conceptual problem modeling results in higher vertex degrees. The algorithm in this case failed to converge to the optimal answer; however, the generated coloring was still acceptable.

## 5 Conclusion

We have presented a genetic algorithm to solve the *graph coloring problem* for arbitrary graphs. Our method has shown promising results in solving a hard combinatorial optimization problem in a reasonable time. Currently, we are incorporating a fast deterministic approximation algorithm in our genetic algorithm in order to improve the speed as well as the graphs coloring.

## References


[1] M. O. Berger, "K-Coloring Vertices using a Neural Network with Convergence to Valid Solutions,"*Proceedings of the International Conference on Neural Networks*, 4514-4517, 1994.

[2] G. Chaitin, "Register Allocation and Splling via Graph Coloring," *Proceedings of the ACM SIGPLAN 82 Symposium on Compiler Construction,* 98-105, 1982.

[3] M. Chams, A. hertz and D. de Werra, "Some Experiments with Simulated Annealing for Coloring Graphs," *European Journal of Operational Research*, Volume 32, Number 2, pp. 260-266.

[4] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[5] M. Garey, D. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, 1979.

[6] M. Garey, D. Johnson, H. So, "An Application of graph coloring to printed circuit testing," *IEEE Transactions on Circuits and Systems,* Vol. 23, pp. 591-599, 1976.

[7] G. De Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw Hill, 1994.

[8] A. Hertz and D. de Werra, "Using Tabu Search Techniques for Graph Coloring," *Computing*, Volume 39, Number 4, pp. 345-351, 1987.

[9] D. Johnson, "Approximation Algorithms for Combinatorial Problems," *J. of Computer System Sciences,* Vol. 9, pp. 256-278, 1974.

[10] D. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Colorinig and Number Partitioning," *Operations Research*, Volume 39, Number 3, pp. 378-406, 1991.

[11] F. Leighton, "A Graph Coloring Algorithm for Large Scheduling Problems," *Journal of Research of the National Bureau of Standards,* Vol. 84, pp. 489-506, 1979.

[12] D. Welsh, M. Powell, "An Upper Bound on the Chromatic Number of a Graph and its Appl. to Timetabling Problems," *Computer,* Vol. 10, pp. 85-86, 1967.

[13] M. Potkonjak and D. Kirovski, "Efficient Coloring of a Large Spectrum of Graphs," *in Proceedings of the 35th ACM/IEEE Design Automation Conference*, pp. 427-431, 1998.

[14] Available from the center for *Discrete Mathematics and Theoretical Computer Science* via ftp ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks.

[15] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Boston, 1989.

[16] S.M. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering,* IEEE Computer Society Press, 1999.

[17] D.E. Goldberg, "Sizing Populations for Serial and Parallel Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms,* J.D. Schaffer, Ed., Morgan Kaufman, San Mateo, CA, 1989.

Table 2. Attempted graph coloring results: Leighton graphs, register allocation graphs, geometric graphs, books' graphs, Mycielski graphs, flat graphs, queen $nxn$ graphs, and course scheduling graphs.

| Benchmark Example | Graphs Characteristics | | | # Colors | Diff. from known optimum | Time (minutes) |
|---|---|---|---|---|---|---|
| | # Vertices | # Edges | $\chi(G)$ | | | |
| le450_15a | 450 | 8,168 | 15 | 15 | 0 | 90 |
| le450_5a | 450 | 5,714 | 5 | 5 | 0 | 160 |
| mulsol.i.1 | 197 | 3,925 | 49 | 49 | 0 | 42 s |
| mulsol.i.2 | 188 | 3,885 | 31 | 31 | 0 | 14 |
| mulsol.i.3 | 184 | 3,916 | 31 | 31 | 0 | 28 |
| fpsol2.i.1 | 496 | 11,654 | 65 | 65 | 0 | 80 |
| fpsol2.i.2 | 451 | 8691 | 30 | 30 | 0 | 45 |
| zeroin.i.1 | 211 | 4,100 | 49 | 49 | 0 | 50 s |
| zeroin.i.2 | 211 | 3,541 | 30 | 30 | 0 | 57 |
| zeroin.i.3 | 206 | 3,540 | 30 | 30 | 0 | 31 |
| inithx.i.1 | 864 | 18,707 | 54 | 57 | 3 | 8 hrs |
| inithx.i.2 | 645 | 13,979 | 31 | 33 | 2 | 6 hrs |
| DSJC125.1 | 125 | 1472 | ? | 6 | 0 | 21 s |
| DSJC250.1 | 250 | 6436 | ? | 9 | 0 | 14 |
| DSJC1000.1 | 1000 | 99,258 | ? | 29 | 7 | 4 hrs |
| DSJR500.1 | 500 | 7110 | ? | 13 | 0 | 9 |
| DSJR500.5 | 500 | 117,724 | ? | 128 | 0 | 15 |
| huck | 74 | 602 | 11 | 11 | 0 | 1 s |
| jean | 80 | 508 | 10 | 10 | 0 | 1 s |
| david | 87 | 812 | 11 | 11 | 0 | 2 s |
| myciel3 | 11 | 20 | 4 | 4 | 0 | 1s |
| myciel4 | 23 | 71 | 5 | 5 | 0 | 1s |
| myciel5 | 47 | 236 | 6 | 6 | 0 | 1s |
| myciel6 | 95 | 744 | 7 | 7 | 0 | 8s |
| myciel7 | 191 | 2,360 | 8 | 8 | 0 | 1 |
| flat300_20_0 | 300 | 21,375 | 20 | 20 | 0 | 90 |
| flat300_26_0 | 300 | 21,633 | 26 | 28 | 2 | 120 |
| flat300_28_0 | 300 | 21,695 | 28 | 32 | 2 | 60 |
| flat1000_50_0 | 1000 | 245,000 | 50 | 61 | 11 | 1 hrs |
| queen5_5 | 25 | 160 | 5 | 5 | 0 | 1 s |
| queen6_6 | 36 | 290 | 7 | 7 | 0 | 4 s |
| queen7_7 | 49 | 476 | 7 | 7 | 0 | 2 |
| queen8_8 | 64 | 728 | 9 | 9 | 0 | 2 |
| queen9_9 | 81 | 2112 | 10 | 10 | 0 | 2 |
| queen11_11 | 121 | 3960 | 11 | 11 | 0 | 6.5 |
| queen12_12 | 96 | 1368 | 12 | 12 | 0 | 3 |
| queen13_13 | 169 | 6,656 | 13 | 13 | 0 | 17 |
| queen15_15 | 225 | 10,360 | ? | 17 | 0 | 18 |
| queen16_16 | 256 | 12,640 | ? | 18 | 0 | 23 |
| school1 | 385 | 19,095 | ? | 14 | 0 | 85 |
| school1_nsh | 352 | 14,612 | ? | 14 | 0 | 65 |
| latin_square_10 | 900 | 307,350 | ? | 112 | 12 | 5 hrs |