AN APPROACH TO SOLVE THE GRAPH COLORING PROBLEM BY GENETIC ALGORITHMS

Harshada Talnikar[1] , Shivaji Pansare [2]

[1]*Asst. Professor, Dept. Of Computer Science, SN Arts, DJM Commerce, BNS Sci College, Sangamner.Maharashtra, India*

[2]*Asst. Professor, Dept. Of Computer Science, SN Arts, DJM Commerce, BNS Sci College, Sangamner.Maharashtra, India*

ABSTRACT

Let G = (V,E) an undirected graph, V corresponds to the set of vertices and E corresponds to the set of edges, we focus on the graph coloring problem (GCP), which consist to associate a color to each vertex so that no two adjacent vertices possess the same color. In this paper we propose a new genetic algorithm based on heuristic to approximate values of v(G) for GCP which achieves highly competitive results.

Keywords: *Genetic Algorithm, graph coloring problem, chromosome, population, crossover*

INTRODUCTION

The Graph Coloring Problem (GCP) is a well-known NP Complete problem. The term graph coloring usually refers to vertex coloring. Given a number of vertices, which form a connected graph, the objective is to color each vertex such that if two vertices are connected in the graph (i.e. adjacent) they will be colored with different colors. The number of different colors that can be used to color the vertices is limited and a secondary objective is to find the minimum number of different colors needed to color a certain graph without violating the adjacency constraint. That number for a given graph (G) is known as the Chromatic Number [1].

Basic Greedy Coloring Algorithm:

```
1. Color vertex v1 with first color.
2. Do following for remaining V-1 vertices.
     a) Pick vertex v and color it with the lowest numbered color which has not been used
        on any previously colored vertices adjacent to it.
     b) If all previously used colors appear on vertices adjacent to v, assign next lowest
        color to it.
```

The above given greedy based algorithm may not always use minimum number of colors. Also, the number of colors used sometime depend on the order of vertices processing. For example, consider the following two graphs. In graph on right side, vertices 3 and 4 are swapped. If we consider the vertices 0, 1, 2, 3, 4 in left graph, we can color the graph with 3 colors. But if we consider the vertices 0, 1, 2, 3, 4 in right graph, we need 4 colors.

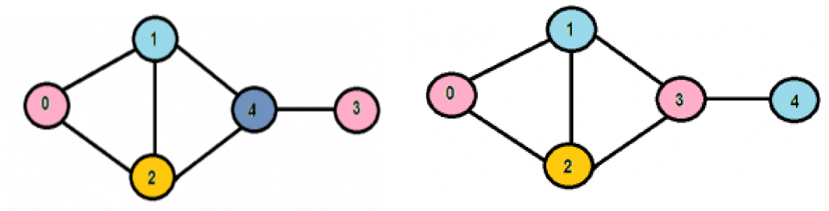

Figure 1: Graphs processed with different orders

So the order of vertices selection is important. Many people have suggested different ways to find an ordering that work better than the basic algorithm on average.

GENETIC ALGORITHMS

Genetic algorithms uses search and optimization technique based on 3 basic principles

1 Selection
2. Crossover
3. Mutation.

GAs simulate those processes in natural populations which are essential to evolution. Exactly which biological processes are essential for evolution? In nature, individuals in a population compete with each other for resources such as food, water and shelter. Also, members of the same species often compete to attract a mate. Those individuals which are most successful in surviving and attracting mates will have relatively larger numbers of offspring. Poorly performing individuals will produce few of even no offspring at all. This means that the genes from the highly adapted, or "fit" individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from different ancestors can sometimes produce "super fit" offspring, whose fitness is greater than that of either parent. In this way, species evolve to become better suited to their environment. Each individual is assigned a "fitness score" according to how good a solution to the problem it is. Potential solutions are obtained as individuals that are evaluated using a fitness function representing a problem being optimized.

Basic structure of a genetic algorithm is shown in the following list:

1. A random population of individuals (potential solutions) is created. All individuals are evaluated using a fitness function.
2. Certain number of individuals that will survive into next generation is selected using selection operator. Selection is somewhat biased, favoring "better" individuals.
3. Selected individuals act as parents that are combined using crossover operator to create children.
4. A mutation operator is applied on new individuals. It randomly changes few individuals (mutation probability is usually low)
5. Children are also evaluated. Together with parents they form the next generation.

Steps 2.-5 are repeated until a given number of iterations have been run, solution improvement rate falls below some threshold, or some other stop condition has been satisfied.

The goal of the any genetic algorithm is to improve the fitness of the population by mating its fittest individuals to produce superior offspring that offer a better solution to the problem. The process continues until a terminating condition is reached and it is the total number of generations has been run or any other parameter like non-improvement of fitness over a certain number of generations or that a solution for the problem has been found [2].

III. PROPOSED ALGORITHM

The chromosome representation of the GCP is an array with the length equal to the number of vertices in the graph. Each cell in the array is assigned a value from 0 to the number of colors – 1. The adjacencies between the vertices are represented by an adjacency matrix of dimensions $n \times n$ where n: the number of vertices. Figure 2 displays the chromosome representation of a graph of 7 vertices using n=4 colors:

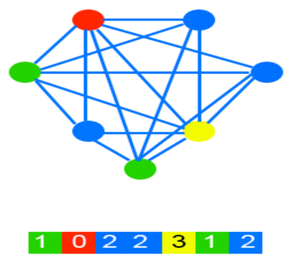

Figure 2: Chromosome representation of a colored 7 vertices connected graph

The Genetic Algorithm process continues until it either finds a solution (i.e. 0 conflicts) or the algorithm has been run for the predefined number of generations. In this approach, the population size is kept constant at all times and with each generation the bottom performing half of the population is removed and new randomly generated chromosomes are added. The population size is kept to 50 chromosomes. The value was chosen after testing a number of different population sizes. The value 50 was the least value that produced the desired results. The GA uses two different parent selection methods a single crossover method and two different mutation methods. Which of the parent selection and mutation methods depends how close it is to finding a solution [3]. The parent selection, crossover and mutation methods are given as follows:

```
Algorithm : parentSelection1
begin
var: parent1, parent2, tempParents
tempParents randomly selected any 2 chromosomes from the population;
parent1 = the fitter of tempParents;

tempParents =  randomly selected any 2 chromosomes from the population;
parent2 = the fitter of tempParents;

return parent1, parent2;
end
```

```
Algorithm: parentSelection2
begin
var: parent1, parent2
parent1 = the top performing chromosome;
parent2 = the top performing chromosome;
return parent1, parent2;
end
```

```
Algorithm: crossover
begin
var: crosspoint, parent1, parent2, child
crosspoint = a random point along a chromosome;
child = colors including crosspoint from parent 1 + colors after crosspoint to the end of the chromosome from parent2;
return child;
end
```

```
Algorithm: mutation1
begin
var: chromosome, allColors, adjacentColors, validColors, newColor
```

```
for each(vertex in chromosome)
     {
         if (vertex has the same color as of its adjacent vertex)
           {
             adjacentColors = all adjacent colors;
             validColors = allColors – adjacentColors;
             newColor = random color from validColors;
            chromosome.setColor(vertex, newColor)
           }
     }
return chromosome;
end
```

Algorithm: mutation2
begin
var: chromosome, allColors

```
for each(vertex in chromosome)
     {
           if (vertex has the same color as an adjacent vertex)
              {
                     newColor = random color from allColors;
                     chromosome.setColor(vertex, newColor)
              }
      }
return chromosome;
end
```

A measure of fitness of any chromosome is number of bad edges i.e. edge connecting two vertices that have the same color. The algorithm is run for some fixed number of generations(e.g 10,000) or until a solution with 0 bad edges is found.

The alteration between the two different parent selection and mutation methods depends on the best fitness. This alteration is determined as the result of experimenting with the different data sets. It was observed that when the best fitness score is low, the usage of parent selection 2 (which copies the best chromosome as the new child) along with mutation2 (which randomly selects a color for the bad vertex) results in a solution more often and more quickly .If the best fitness is greater than 4 then parentSelection1 and mutation1 are used. If the best fitness is 4 or less then arentSelection2 and mutation2 are used.

CONCLUSION

During the design of this approach, the issue of designing good operators was of great concern. Any good operator can bring the chromosomes of a population closer to the desired solution. During the process, a chromosome’s fitness often improves but eventually ends up in a local optimum. This approach aimed to improve fitness without the pitfall of overcoming the global optimum in favor of a local optimum.

Different factors need to be considered, firstly, the crossover function is applied to parents that result from the first parent selection method. This method selects parents by conducting competition between random pairs of chromosomes. Two pairs are chosen randomly and the one with better finess of each pair becomes a parent.
These fit parents are then used as input to this crossover method. The crossover conducts a simple one-point crossover with the cross point being chosen at random.
The result of this crossover is then subjected to the first mutation method. This mutation examines each vertex and if a vertex violates the coloring constraint a valid color is chosen at random.

This process is very effective in reducing the number of conflicts rapidly. However, it has the side effect of keeping the solution at a local optimum. To overcome this problem another parent selection and mutation method is introduced. The two methods are applied when the overall fitness of the best solution drops below 5 conflicts. After that crossover is not applied. The top performer is selected and is subjected to the second mutation method. This method fixes the conflicting vertices and replaces their conflicting colors with random colors. This has the potential to either find a globally optimum solution (i.e. 0 conflicts).

REFERENCES

1. Abbasian, Reza, and Mouhoub, Malek. (2011), "An Efficient Hierarchical Parallel Genetic Algorithm for Graph Coloring Problem," in Proceedings of the 13th annual conference on Genetic and evolutionary computation, Dublin, Ireland: July 12- 16, 2011 ACM, pp. 521-528.

2. Ali, F. F., Nakao, Z., Tan, R. B., and Yen-Wei, Chen. (1999), "An Evolutionary Approach for Graph Coloring," in International Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE 1999, pp. 527-532 vol.525.

3. Ashby, Leif H., and Yampolskiy, Roman V. . (2011), "Genetic Algorithm and Wisdom of Artificial Crowds Algorithm Applied to Light Up," in 16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games, Louisville, KY, USA: July 27 – 30, 2011, pp. 27-32.