Genetic Algorithm with Wisdom of Artificial Crowds Applied to the Graph Coloring Problem

Musa M. Hindi and Roman V. Yampolskiy

Computer Engineering and Computer Science
J.B. Speed School of Engineering
Louisville, Kentucky

Abstract

In this paper we present a hybrid technique that applies a genetic algorithm followed by wisdom of artificial crowds approach to solving the graph-coloring problem. The genetic algorithm described here utilizes more than one parent selection and mutation methods depending on the state of fitness of its best solution. This results in shifting the solution to the global optimum more quickly than using a single parent selection or mutation method. The algorithm is tested against the standard DIMACS benchmark tests while limiting the number of usable colors to the known chromatic numbers. The proposed algorithm succeeded at solving the sample data set and even outperformed a recent approach in terms of the minimum number of colors needed to color some of the graphs.

The Graph Coloring Problem (GCP) is a well-known NP-complete problem. Graph coloring includes both vertex coloring and edge coloring. However, the term graph coloring usually refers to vertex coloring rather than edge 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. Moreover, 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 ($\chi(G)$) (Isabel Méndez Díaz and Paula Zabala 1999).

If $k = \{1, 2, 3...\}$ and $P(G, k)$ is the number of possible solutions for coloring the graph G with k colors, then

$$\chi(G) = min(k: P(G, k) > 0) \qquad (1)$$

Graph coloring problems are very interesting from the theoretical standpoint since they are a class of NP-complete problems that also belong to Constraint Satisfaction Problems (CSPs). The practical applications of Graph Coloring Problems include but are not limited to:

- Map coloring (B. H. Gwee, M. H. Lim and J. S. Ho 1993)
- Scheduling (Daniel Marx and D Aniel Marx 2004)
- Radio Frequency Assignment (W. K. Hale 1980; S. Singha, T. Bhattacharya and S. R. B. Chaudhuri 2008)
- Register allocation (Wu Shengning and Li Sikun 2007)
- Pattern Matching
- Sudoku

In this paper we demonstrate the use of genetic algorithms in solving the graph-coloring problem while strictly adhering to the usage of no more than the number of colors equal to the chromatic index to color the various test graphs.

Prior Work

A great deal of research has been done to tackle the theoretical aspects of the Graph Coloring Problem in terms of its generalization as a Constraint Satisfaction Problem (Isabel Méndez Díaz and Paula Zabala 1999). The problem's various applications and solutions have been discussed in detail in Porumbel's paper (Daniel Cosmin Porumbel). Evolutionary computation and parameter control has been detailed in a number of papers including ones by Back, Hammel, and Schwefel (T. Back, U. Hammel and H. P. Schwefel 1997) as well as work by Eiben, Hinterding and Michalewicz (A. E. Eiben, R. Hinterding and Z. Michalewicz 1999). Srinivas and Patnaik examined crossover and mutation probabilities for optimizing genetic algorithm performance (M. Srinivas and L. M. Patnaik 1994). Genetic algorithms and evolutionary approaches have been used extensively in solutions for the Graph Coloring Problem and its applications (F. F. Ali, et al. 1999; K. Tagawa, et al. 1999; Cornelius Croitoru, et al. 2002; C. A. Glass and A. Prugel-Bennett 2003; Justine W.

Shen 2003; Greg Durrett, Muriel Médard and Una-May O'Reilly 2010; Lixia Han and Zhanli Han 2010). Most recent work utilized a parallel genetic algorithm on a similar dataset to the one used in this paper (Reza Abbasian and Malek Mouhoub 2011).

The concept of utilizing a crowd of individuals for solving NP complete problems has also been the topic of various papers. Most notably the Wisdom of Crowds concept has been used in solving the Traveling Salesman Problem (Sheng Kung Michael Yi, et al. 2010b) as well as the Minimum Spanning Tree Problem (Sheng Kung Michael Yi, et al. 2010a). In this paper we attempt to supplement the solution produced by the genetic algorithm utilizing an artificial crowd (Leif H. Ashby and Roman V. Yampolskiy 2011; Leif H. Ashby and Roman V. Yampolskiy 2011).

Proposed Approach

Genetic algorithms share an overall structure and workflow yet they vary in the specific details according to the particular problem. The algorithm consists of a parent selection method, a crossover method and a mutation method.

The general algorithm is:

Algorithm1: General Genetic Algorithm
define: population, parents, child

```
population = randomly generated chromosomes;

while (terminating condition is not reached) {
    gaRun();
}

// a single run of a genetic algorithm
function gaRun() {
    parents = getParents();
    child = crossover(parents);
    child = mutate(child);
population.add(child);
}
```

The goal of the previous 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. This process continues until a terminating condition is reached which could be simply that 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.

The chromosome representation of the GCP is simply an array with the length set 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 1 displays the chromosome representation of a graph of 7 vertices using 4 colors:

The ultimate goal when solving GCPs is to reach a solution where no two adjacent vertices have the same color. Therefore, the GA 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 addition to the GA, if a run fails to reach a solution a Wisdom of Crowds approach will be applied to the top performers in an attempt to produce a better solution.

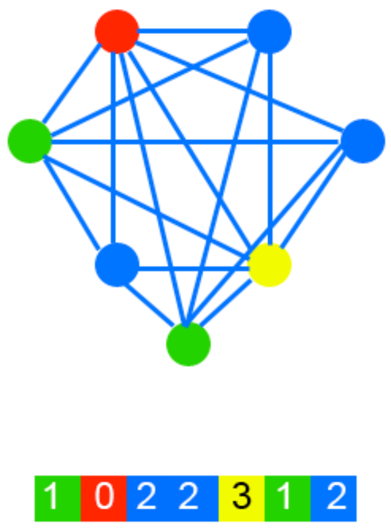

Figure 1: Chromosome representation of a colored connected graph

In our 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 set 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 ends up selected depends on the state of the population and how close it is to finding a solution. The parent selection, crossover and mutation methods are outlined as follows:

Algorithm2: parentSelection1:
define: parent1, parent2, tempParents;

```
tempParents = two randomly selected chromosomes from the population;
parent1 = the fitter of tempParents;

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

return parent1, parent2;
```

Algorithm3: parentSelection2:
define: parent1, parent2

```
parent1 = the top performing chromosome;
parent2 = the top performing chromosome;

return parent1, parent2;
```

Algorithm4: crossover
define: crosspoint, parent1, parent2, child

```
crosspoint = random point along a chromosome;
child = colors up to and including crosspoint from parent 1 + colors after crosspoint to the end of the chromosome from parent2;

return child;
```

Algorithm5: mutation1:
define: chromosome, allColors, adjacentColors, validColors, newColor;

```
for each(vertex in chromosome) {
if (vertex has the same color as an adjacent vertex) {
        adjacentColors = all adjacent colors;
        validColors = allColors – adjacentColors;

        newColor  = random color from validColors;
        chromosome.setColor(vertex, newColor)
  }
}
return chromosome;
```

Algorithm6: mutation2:
define: 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;
```

A bad edge is defined as an edge connecting two vertices that have the same color. The number of bad edges is the fitness score for any chromosome. As mentioned above, the alteration between the two different parent selection and mutation methods depends on the best fitness. If the best fitness is greater than 4 then parentSelection1 and mutation1 are used. If the best fitness is 4 or less then parentSelection2 and mutation2 are used. This alteration is the result of experimenting with the different data sets. It was observed that when the best fitness score is low (i.e. approaching an optimum) 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 violating vertex) results in a solution more often and more quickly than using the other two respective methods.

Finally, the algorithm is run for 20,000 generations or until a solution with 0 bad edges is found. If a solution is not found after 20,000 generations the wisdomOfArtificialCrowds algorithm is run. The algorithm used is a localized wisdom of crowds algorithm that only builds a consensus out of the violating edges in the best solution. Moreover, it uses the best half of the final population to produce an aggregate solution. Only a localized consensus is generated so as not to produce a result that alters the correctly colored vertices. Also, it takes the best half because they share the most similarity and thus will most likely be different at the level of the bad edges rather than the good ones.

Algorithm7: wisdomOfArtificialCrowds
define: aggregateChromosome, newColor, expertChromosomes;

```
expertChromosomes = best half of the final population;
aggregateChromosome = best performing chromosome;

for     each (vertex in graph) {
  if (vertex is part of a bad edge) {
    newColor = most used color for vertex among expertChromosomes;
    aggregateChromosome.setColor(vertex, newColor)
  }
}
```

Data

Data used to test our approach are derived from the DIMACS benchmarking graph collection. DIMACS is the Center for Discrete Mathematics and Theoretical Computer Science. It is part of Rutgers University (The State University of New Jersey Rutgers 2011; Roman V. Yampolskiy and Ahmed El-Barkouky 2011). The data are frequently used in challenges involving constraint satisfaction problems. The files used have a .col extension. Each file contains a header with the number of vertices (p) and the number of edges (e):

```
p edge 496 11654
```

A number of lines follow the header with each line denoting the connected edges and their vertex indices:

```
e 1 100
```

16 files were chosen from the DIMACS collection. The graphs the files represent vary in vertex count, edge count and overall complexity. The vertex count ranges between 11 and 561 and the edge count ranges from 20 to 11654. The rationale behind the selection of these graphs other than the wide range of variation is that there is a known chromatic number for each of them, or at least a good approximation.

The following files were used in this approach: (myciel3.col, myciel4.col, myciel5.col, queen5_5.col, queen6_6.col, queen7_ 7.col, queen8_8.col, huck.col, jean.col, david.col. games120.col, miles250.col, miles1000.col, anna.col, fpsol2.i.1.col, homer.col).

Results

Table 1 displays the following statistics for each file:

- The number of vertices |V|
- The number of edges |E|
- The expected chromatic number $\chi(G)$
- The minimum number of colors used by this algorithm k_{min}
- The minimum number of colors used by a comparative publication using a Hybrid Parallel Genetic Algorithm (HPGAGCP)
- Average time it took to find a solution

The genetic algorithm was developed in Java utilizing JDK 1.6 and JUNG (Java Universal Network/Graph) framework for graph visualization (Joshua O'Madadhain, Danyel Fisher and Tom Nelson). Performance plots were generated using MATLAB R2010a.

The tests were run on a desktop PC with the following specifications:

CPU: Intel Core i7 860 @2.8Ghz
RAM: 8 GB DDR3 @1333MHz
HDD: 500GB 7200 RPM

File	|V|	|E|	Expected $\chi(G)$	k_{min}	HPGAGCP Result	Time (s)
myciel3.col	11	20	4	4	4	0.003
myciel4.col	23	71	5	5	5	0.006
queen5_5.col	23	160	5	5	5	0.031
queen6_6.col	**25**	**290**	**7**	**7**	**8**	**6.100**
myciel5.col	36	236	6	6	6	0.014
queen7_7.col	**49**	**476**	**7**	**7**	**8**	**6.270**
queen8_8.col	**64**	**728**	**9**	**9**	**10**	**47.482**
huck.col	74	301	11	11	11	0.015
jean.col	80	254	10	10	10	0.015
david.col	87	406	11	11	11	0.019
games120.col	120	638	9	9	9	0.027
miles250.col	128	387	8	8	8	0.076
miles1000.col	128	3216	42	42	42	48.559
anna.col	138	493	11	11	11	0.058
fpsol2.i.1.col	496	11654	65	65	65	22.656
homer.col	561	1629	13	13	13	0.760

Table 1: Results of running the proposed algorithm on 16 .col files from the DIMACS collection

The following graphs plot the relationship between the fitness and the generation for a sample set of the files used:

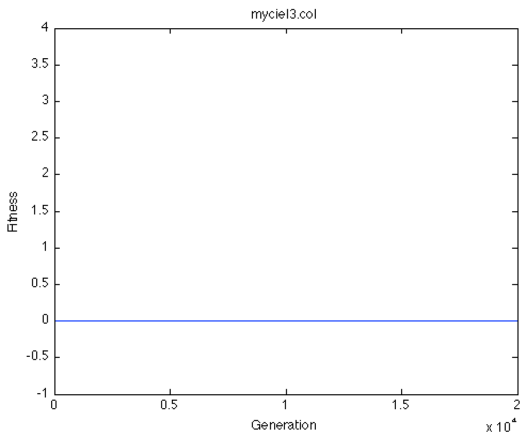

Figure 2: Fitness score over the number of generations for myciel3.col

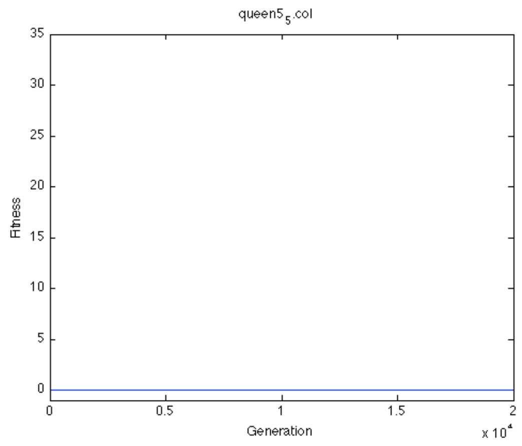

Figure 3: Fitness score over the number of generations for queen5_5.col

The next plot is of particular interest since it clearly represents the erratic behavior of the fitness score between the initial rapid drop until a solution is ultimately found. In standard genetic algorithms the fitness score continues to increase or decrease (depending of the definition of better fitness) until the end of the run. This is not the case here. This factor plays a huge role in obtaining the global optimum with a higher probability than without it as will be discussed later.

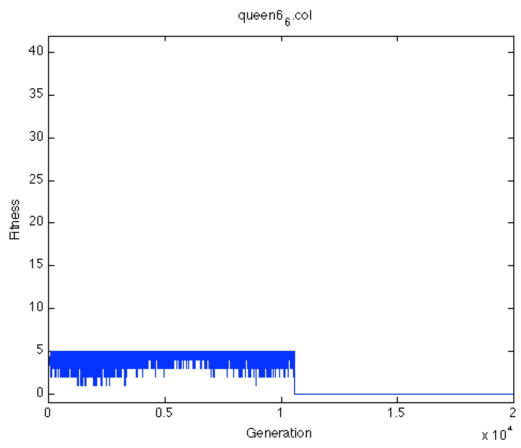

Figure 4: Fitness score over the number of generations for queen6.col

Sample solutions: (graphs were visualized using the JUNG framework (Joshua O'Madadhain, Danyel Fisher and Tom Nelson)):

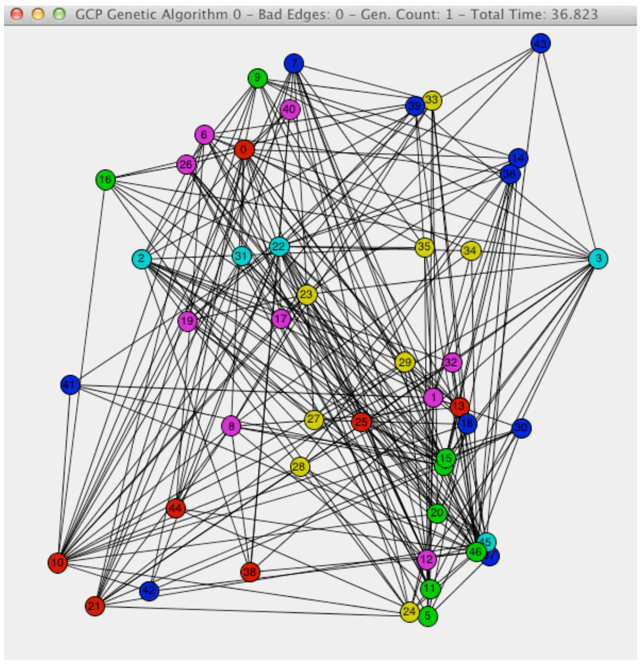
GCP Genetic Algorithm 0 – Bad Edges: 0 – Gen. Count: 1 – Total Time: 36.823

Figure 5: GCP solution for myciel5.col

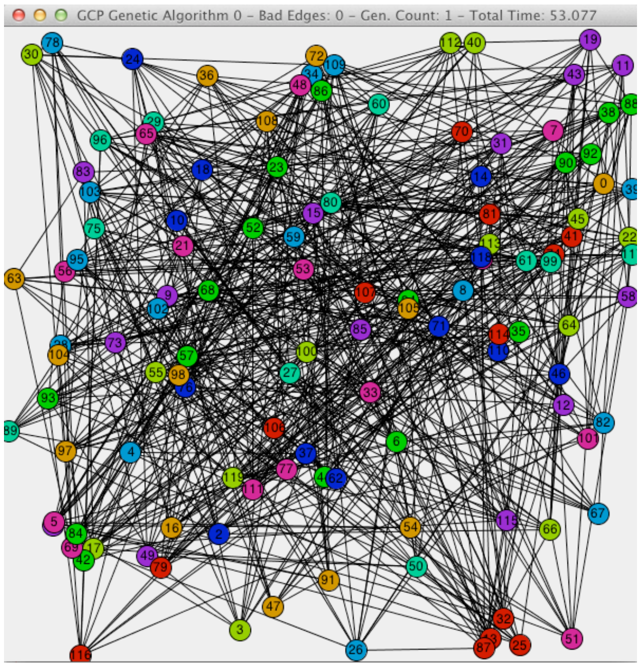
GCP Genetic Algorithm 0 – Bad Edges: 0 – Gen. Count: 1 – Total Time: 53.077

Figure 6: GCP solution for games120.col

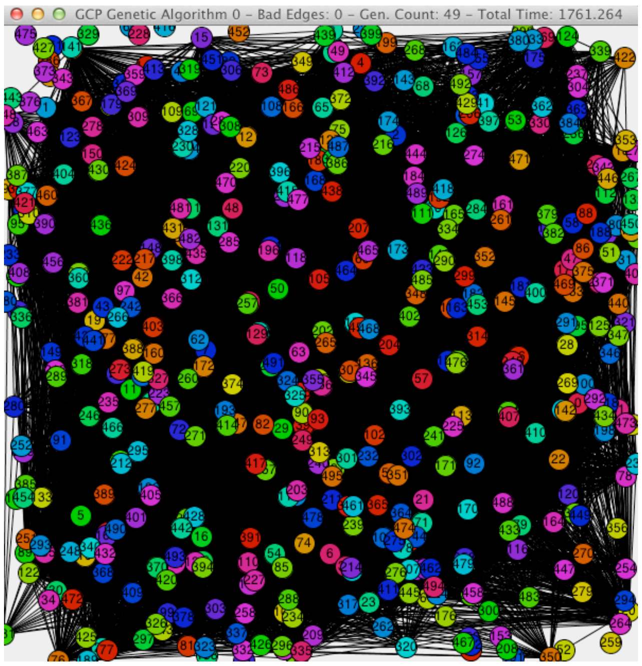
GCP Genetic Algorithm 0 – Bad Edges: 0 – Gen. Count: 49 – Total Time: 1761.264

Figure 7: GCP solution for fpsol2.i.col

Discussion

During the design of this approach, the issue of designing good operators was a constant concern. The goal of any good operator is to bring the chromosomes of a population closer to the desired solution. However, during the process, a chromosome's fitness often improves but eventually ends up in a local optimum. With that in mind, the overall design of this approach aimed to improve fitness without the pitfall of evading the global optimum in favor of a local optimum.

To achieve that, a number of factors need to be considered. Initially, the crossover function is applied to parents that result from the first parent selection method. This method selects parents by conducting a small tournament between random pairs of chromosomes. Two pairs are chosen randomly and the fitter 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. The mutation is carried out at a high rate of 0.7. 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 which can be seen in all the plots through an almost perpendicular drop in fitness. However, in spite of this method's effectiveness at increasing fitness rapidly, it has the side effect of keeping the solution at a local optimum.

To fix that, 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 point crossover is no longer applied. The top performer is selected and is subjected to the second mutation method. This method finds the conflicting vertices and replaces their conflicting colors with random colors; which could be invalid as well. This has the potential to either find a globally optimum solution (i.e. 0 conflicts) or produce a solution that is worse! This can be observed by the erratic pattern in some of the graphs after the sharp descent and before the actual resolution of the problem.

This seemingly worsening fitness is not bad however. In fact, it is partly due to this worsening of the fitness that some solutions are found at all! When the solution becomes worse the fitness score increases. This will force the algorithm back to using the first parent selection and mutation methods. But, the population now contains a solution that hadn't been there before, which increases the possibility of reaching the global optimum. The continuous back and forth between parent selection and mutation methods plays a crucial role in shifting the solution from a local optimum to a global optimum.

Finally, if a solution is not found after 20,000 generations, a Wisdom of Artificial Crowds algorithm is applied to the resultant population to produce a better solution. The genetic algorithm had been producing optimal results and thus, per the algorithmic workflow, the Wisdom of Artificial Crowds postprocessor wouldn't be applied. However, in order to test its effectiveness, a test was conducted by decreasing the generation count to 10,000 to intentionally produce a suboptimal solution. The test carried out on the miles1000.col file. Before the postprocessor was applied the best solution had 5 conflicting edges. After application of the Wisdom of Artificial Crowds postprocessor the graph was colored slightly differently but still had 5 conflicting edges. It is worth noting that the postprocessor added almost no significant time to the overall process. The average additional time needed is 250ms.

The test cases used were very useful in both testing and tuning the algorithm. The algorithm was able to scale across the different graphs and produce optimum solutions in each case. A recent 2011 publication presented a parallel genetic algorithm for the GCP (Reza Abbasian and Malek Mouhoub 2011). This paper used most of the same test files that were used in Abbasian's approach. That approach's proposed algorithm failed to solve three of the graphs and only produced solutions when the color count was increased by 1. The files representing these graphs are queen6_6.col, queen7_7.col and queen8_8.col. The algorithm used in our approach successfully solved these three files using the specified known chromatic number as the number of colors used.

Conclusion

The set of algorithms described in this paper is really a combination of many approaches to solving NP-complete problems. The overarching algorithm used is a genetic algorithm with a subsequent Wisdom of Crowds post-processor. Within the genetic algorithm itself is a set of operators that utilize methods from the genetic algorithm domain as well as applying various heuristics in a stochastic manner. The end result is an overall progressive climb to the peak of the globally optimum solution in a fairly quick manner.

The algorithms described here can also be applied to the various subsets of the general GCP. In particular, Sudoku can benefit from these algorithms where it can be represented as a graph with 81 vertices that must be colored using no more than 9 different colors (i.e. different numbers).

References

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.

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.

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.

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.

Back, T., Hammel, U., and Schwefel, H. P. (1997), "Evolutionary Computation: Comments on the History and Current State," Evolutionary Computation, IEEE Transactions on, Vol. 1, 1, 3-17.

Croitoru, Cornelius, Luchian, Henri, Gheorghies, Ovidiu, and Apetrei, Adriana. (2002), "A New Genetic Graph Coloring Heuristic," in Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA, pp. 63-74.

Díaz, Isabel Méndez, and Zabala, Paula. (1999), "A Generalization of the Graph Coloring Problem," Departamento de Computacion, Universidad de Buenes Aires.

Durrett, Greg, Médard, Muriel, and O'Reilly, Una-May. (2010), "A Genetic Algorithm to Minimize Chromatic Entropy," (Vol. 6022), eds. P. Cowling and P. Merz, Springer Berlin / Heidelberg, pp. 59-70.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999), "Parameter Control in Evolutionary Algorithms," Evolutionary Computation, IEEE Transactions on, Vol. 3, 2, 124-141.

Glass, C. A., and Prugel-Bennett, A. (2003), "Genetic Algorithm for Graph Coloring: Exploration of Galinier and Hao's Algorithm," Journal of Combinatorial Optimization, Vol. 7, 3, 229-236, Sep.

Gwee, B. H., Lim, M. H., and Ho, J. S. (1993), "Solving Four-Colouring Map Problem Using Genetic Algorithm," in First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, New Zealand: 24-26 Nov 1993, pp. 332-333.

Hale, W. K. (1980), "Frequency Assignment: Theory and Applications," Proceedings of the IEEE, Vol. 68, 12, 1497-1514, Dec. 1980.

Han, Lixia, and Han, Zhanli. (2010), "A Novel Bi-Objective Genetic Algorithm for the Graph Coloring Problem," in Proceedings of the 2010 Second International Conference on Computer Modeling and Simulation - Volume 04, IEEE Computer Society, pp. 3-6.

Marx, Daniel, and Marx, D Aniel. (2004), "Graph Coloring Problems and Their Applications in Scheduling," in John von Neumann PhD Students Conference, pp. 1-2.

O'Madadhain, Joshua, Fisher, Danyel, and Nelson, Tom, "Jung: Java Universal Network/Graph Framework", Oct. 1 2011, http://jung.sourceforge.net/.

Porumbel, Daniel Cosmin, "Applications to the Graph Coloring Problem."

Rutgers, The State University of New Jersey (2011), "Center for Discrete Mathematics & Theoretical Computer Science."

Shen, Justine W. (2003), "Solving the Graph Coloring Problem Using Genetic Programming," in Genetic Algorithms and Genetic Programming at Stanford 2003, Stanford Bookstore, pp. 187-196.

Shengning, Wu, and Sikun, Li. (2007), "Extending Traditional Graph-Coloring Register Allocation Exploiting Meta-Heuristics for Embedded Systems," in Third International Conference on Natural Computation. ICNC 2007., Haikou, Hainan, China: 24-27 Aug. 2007, pp. 324-329.

Singha, S., Bhattacharya, T., and Chaudhuri, S. R. B. (2008), "An Approach for Reducing Crosstalk in Restricted Channel Routing Using Graph Coloring Problem and Genetic Algorithm," in Computer and Electrical Engineering. ICCEE 2008. International Conference on, 20-22 Dec. 2008, pp. 807-811.

Srinivas, M., and Patnaik, L. M. (1994), "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," Systems, Man and Cybernetics, IEEE Transactions on, Vol. 24, 4, 656-667.

Tagawa, K., Kanesige, K., Inoue, K., and Haneda, H. (1999), "Distance Based Hybrid Genetic Algorithm: An Application for the Graph Coloring Problem," in Congress on Evolutionary Computation, 1999. CEC 99., 1999, p. 2332 Vol. 2333.

Yampolskiy, Roman V. , and El-Barkouky, Ahmed. (2011), "Wisdom of Artificial Crowds Algorithm for Solving Np-Hard Problems," International Journal of Bio-Inspired Computation (IJBIC), Vol. 3, 6, 358-369.

Yi, Sheng Kung Michael, Steyvers, Mark, Lee, Michael D., and Dry, Matthew. (2010a), "Wisdom of the Crowds in Minimum Spanning Tree Problems," in Proceedings of the 32nd Annual Conference of the Cognitive Science Society, Portland, Oregon 10-13 Aug. 2010.

Yi, Sheng Kung Michael, Steyvers, Mark, Lee, Michael D., and Dry, Matthew J. (2010b), " Wisdom of the Crowds in Traveling Salesman Problems."