The Effects of Varying the Fitness Function on the Efficiency of the Cat Swarm Optimization algorithm in Solving the Graph Coloring Problem

Noel D. Bacarisas and John Paul T. Yusiong
Division of Natural Sciences and Mathematics,
University of the Philippines Visayas Tacloban College
Tacloban City, Leyte, Philippines

ABSTRACT: The Cat Swarm Optimization (CSO) algorithm is a relatively recent addition to the family of algorithms known as Swarm Intelligence and is based on the common behavior of cats. With the claim of improved performance in finding the global best solutions, it is tested in solving the well-studied Graph Coloring Problem (GCP). The goal in GCP is to color a graph using the least number of colors possible such that no connected vertices share the same color. Aside from solving GCP, the effect of a fitness function in the efficiency of CSO is also assessed. Experiment results showed that CSO was able to solve all GCP instances. Furthermore, CSO was able to find the optimal solution for some graph types. Also it is shown that the nature of the solution found is affected by the fitness function used. These results indicate that CSO is a feasible algorithm in solving GCP.
KEYWORDS: Graph Coloring Problem, chromatic number, Cat Swarm Optimization, fitness function, Swarm Intelligence

Introduction

In the past century, the Graph Coloring Problem (GCP) has gained popularity among researchers and has become one of the most well studied optimization problem. The popularity of GCP naturally arises due to its numerous practical applications which include scheduling, time tabling and

frequency assignment [She03]. To add to these, other significant uses also include aircraft flight scheduling and biprocessor tasks allocation [Mar04]. These problems can be modeled into graphs and thus be solved in the same manner as GCP.

Most algorithms which have been used to solve GCP in past decades are optimization algorithms. These include Genetic Algorithm [She03], Simulated Annealing [J+91], Ant Colony Optimization [SE08] and Artificial Bee Colony [FJ11]. Optimization algorithms randomly search for solutions guided by some hints to find the next solution [CPT06]. Due to the randomness of the solution found, optimization algorithms employ fitness functions to help them decide whether a given solution is good or not and how good the solution is compared to other solutions. Fitness functions serve as a guide to the algorithm in finding the optimal solution. With the success of other optimization algorithms like the Artificial Bee Colony [FJ11] algorithm and Ant Colony Optimization [SE08] in solving GCP, this study finds motivation in exploring the feasibility of another optimization algorithm in solving the same problem and at the same time determining how fitness functions affect the efficiency of an algorithm.

Based on the common behavior of cats, a relatively new optimization algorithm is proposed by Chu et. al. This algorithm is known as Cat Swarm Optimization and claims better performance than another popular optimization algorithm which is the Particle Swarm Optimization (PSO) and its improved variant, PSO with weighting factor [CT07]. Although experiments by its proponents indicate that CSO takes more time per cycle, its strength as an optimization algorithm lies in its improved performance in finding the solution to the problem in a relatively few iterations. Previous researches using CSO involved solving Sudoku puzzles [HR10] and clustering [SN09]. The said researches supported the proponents claim that CSO finds the global best solution in a relatively few cycles.

1. Graph Coloring Problem

The Graph Coloring Problem or GCP can be simply described as an assignment of colors to the vertices of the graphs following a constraint. The constraint imposed in GCP is that vertices which are connected by an edge should not have the same color. If two connected edges share the same color then it results to a conflict. When a graph is colored in such a way that there are no conflicts, the graph is regarded to be properly colored. Aside from properly coloring the graph,

GCP also seeks to use the least number of colors necessary thus the actual goal of GCP is to find a proper coloring for a graph using the minimum number of colors needed. This minimum number of colors necessary to properly color a graph is called the chromatic number.

Finding a solution to GCP could lead to two types of solutions. These are the complete solution and the optimal solution. A complete solution is any proper coloring of a graph while an optimal solution is a proper coloring using a number of colors equal to the chromatic number of the graph. Figure 1(a) shows a sample graph, Figure 1(b) illustrates a coloring with conflicts, Figure 1(c) shows a proper coloring and Figure 1(d) shows an optimal coloring of the graph.

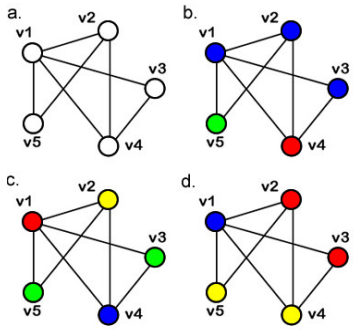

Figure 1. A sample graph and its different colorings

By modeling some real-world problems into graphs, solutions can be found by solving these problems in the same manner as GCP. Figure 2 shows how GCP can solve a class scheduling problem.

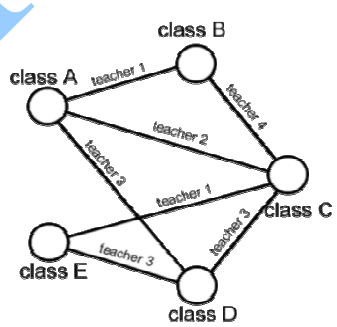

Figure 2. A Graph model of a scheduling problem

In Figure 2, the classes being scheduled are represented as vertices. Two classes are connected if they have the same teacher. If we represent the

available time schedules as colors, the problem would now be to assign a color to each vertex but in order to avoid conflicts in schedule, vertices which are connected should have different colors. Notice that this is just the same as GCP.

2. The Cat Swarm Optimization (CSO) Algorithm

In the search for efficient algorithms to solve computationally complex problems, researchers have noticed the potential of the collective behavior of animals in attaining a common goal to have a significant resemblance with finding an optimal solution. This interesting observation led to the proposal of a lot of different algorithms which are based on the behavior of animals. These algorithms belong to a family of algorithms which is commonly known as Swarm Intelligence. The phrase Swarm Intelligence or SI was first introduced in the late 1980s by Beny and Wang in the context of cellular robotics where a swarm is defined as a population of agents working cooperatively to achieve some purposeful behavior and goal. [DAK08].

By simulating the behavior of animals, efficient algorithms have been created and have gained popularity among other researchers. Popular examples of SI based algorithms include Artificial Bee Colony (ABC) and Ant Colony Optimization (ACO). ABC was realized by observing the behavior of bees when foraging for food meanwhile ACO was designed to model the ants' behavior of leaving pheromone trails while walking in search of food.

In 2006, another SI based algorithm is introduced by Chu et. al [CT07]. This algorithm is known as Cat Swarm Optimization or CSO. As its name clearly suggests, this algorithm is based on the behavior cats. Based on observation, a cat's behavior can be generalized into two. These behaviors are the observantly resting behavior and actively moving or chasing behavior. Although they appear to be lazy and spend most of the time resting, through a more careful observation they are found to be keenly observing their surroundings waiting for their next move to take.

In CSO, these two behaviors are modeled into two sub-modes which are seeking mode for resting and tracing mode for moving. Furthermore in CSO, a cat has a position and velocity. The position of the cat is represents a solution. Using the two sub-modes each cat will update its position thereby exploring and exploiting different positions or solutions to the problem.

2.1. Seeking Mode

When a cat is resting, it observes its environment for the best place to move to given its current position. After it has ascertained the best position to move to, it transfers to that spot. The same thing happens in seeking mode where the observant behavior of cats is imitated by creating copies of the current solution. Each copy would then try to improve the given solution through a process known as exploitation. After all copies have finished exploiting the current solution, the last step would be to select the new solution which would replace the current solution. This new solution would represent the new spot on which the cat has moved to.

Seeking mode uses four (4) important factors namely Seeking Memory Pool (SMP), Seeking Range of Dimension (SRD), Counts of dimension to Change (CDC) and Self Position Consideration (SPC). For a real cat, SMP would dictate the number of observations to consider before deciding the best spot to transfer to. SPC would give the cat the freedom whether to move or not. If the cat finds its current position as the best position then it will not transfer and stay on the same spot. SRD and CDC are both necessary factors in updating the solution. The detailed process in seeking mode is described by the following steps:

1. Create copies of cat_k based on j. If SPC is true set j = SMP – 1 and include cat_k as a candidate cat else set j = SMP.
2. For each copy, select a number of dimensions based on CDC and update their values using SRD percents of their current value.
3. Evaluate the fitness of each copy.
4. Compute for the probability of each copy to be selected as the new position of cat_k using equation (1)

$$P_i = \frac{|FSi - FSb|}{FSmax - FSmin}, \text{ where } 0 < i < j \qquad (1)$$

If a minimization fitness function is used,

$$FS_b = FS_{min}$$

else

$$FS_b = FS_{max}.$$

5. Randomly select a new position for cat_k from the candidate cats.

2.2. Tracing Mode

When a cat is chasing a prey or any moving object, it moves according to its velocity. The positions on which it moves to can be any spot from its current position to anywhere its velocity would allow it to go. This is also the same

in tracing mode where the new solution is determined by updating the current solution using the velocity values. This way of updating the solution introduces a wide search throughout the solution space and is referred to as the process of exploration. This ensures that the algorithm is robust and does not get trapped in the local optima. The tracing mode process can be described by the following steps:

1. Update the velocities of all dimensions using equation (2).

$$V_{k,d} = V_{k,d} + r1 \cdot c1 \cdot (X_{best,d} - X_{k,d}),$$
$$d = 1, 2, \ldots, M \qquad (2)$$

where $X_{best,d}$ is the position of the best cat or the cat with the best fitness value, $X_{k,d}$ is the current position of cat_k, r1 is a random value in range of [0,1] and cl is a constant.

2. Check if the new velocity values are within range of maximum velocity. In case the new velocity goes over the range, set it to maximum.
3. Use equation (3) to update the position of cat_k.

$$X_{k,d} = X_{k,d} + V_{k,d} \qquad (3)$$

2.3. Overall CSO Process

The overall CSO process is shown by the flowchart in Figure 3 and detailed by the pseudo-code that follows.

Cat Swarm Optimization

Begin
  Create N Cats
  Initialize the position, velocity and flag of each cat
  Set cycle = 1
  repeat
    Evaluate the position of each cat and keep the position the cat with the best fitness value.
    If flag of cat_k is Seeking mode
     Apply cat_k into Seeking mode Process
    Else
     Apply cat_k into Tracing Mode Process
    Reinitialize the flag of each cat
    cycle = cycle + 1
  until (termination condition is satisfied)
End

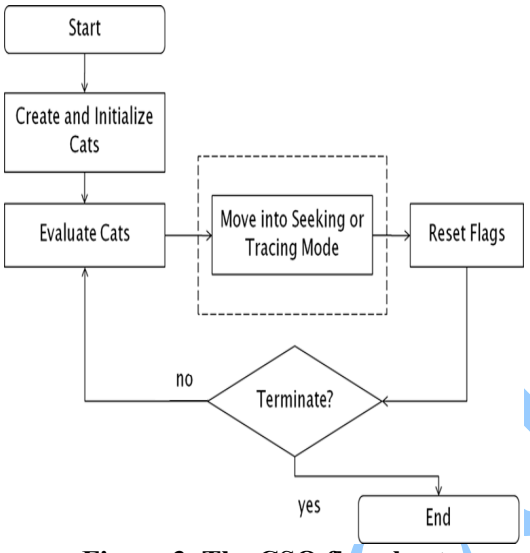

Figure 3. The CSO flowchart

By using a Mixture Ratio or MR, the two sub-modes are combined to form the general flow of CSO. MR dictates the number of cats which would be randomly selected to move into tracing mode while the remaining cats are set to move into seeking mode. To mimic the behavior of cats in which they are resting most of the time, MR is usually given a small value.

3. Chromaticats: Solving GCP using CSO

In solving GCP, we can consider any number of colors to use. However, it is preferable to narrow down the number of colors to consider so that the solution space will also be narrowed down. But on the other hand if the choice of color is narrowed down too much it could happen that the number of colors being considered is not sufficient to color the graph properly thus it would be impossible to solve GCP.

In order to have a reasonable upper bound for the number of colors to consider that ensures a complete solution for a graph; Brook's theorem is utilized in this research. The theorem guarantees that the maximum number of colors necessary to color a certain graph is equal to the graph's Brook's value or the maximum-vertex degree of the graph + 1. This theorem is proven in [Tve83].

In CSO the position of the cat represents a solution to the problem thus when solving GCP, the position of the cats would correspond to different colorings of the graph. In simple terms what happens in the CSO process is that the cats move from place to place until it finds the best position.

Since GCP has two types of solutions, the approach used is to find a complete solution quickly and gradually optimize the complete solution by reducing the number of colors used until the optimal solution is found. However to avoid being trapped in the local optima, regressions would be allowed at some point through exploration of other solutions. This is shown in Figure 4.

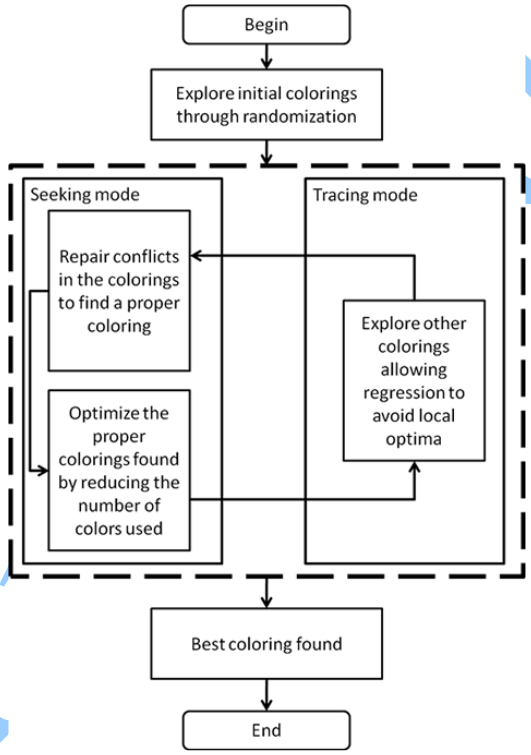

Figure 4. The Chromaticats flowchart

3.1. The Chromaticats Seeking Mode Process

In this approach, seeking mode is designed to find a complete solution and further optimize it to find the optimal solution. In order to find a complete solution quickly, seeking mode uses a color selection which is biased towards lessening the number of conflicts in the graph.

To lessen the number of conflicts, the selection of colors will be biased towards colors which are not yet assigned to any vertex. Since the color selected is not yet used then this ensures that using it would introduce no conflicts. However in the case that there is no more unused color available, the selection would just be done randomly.

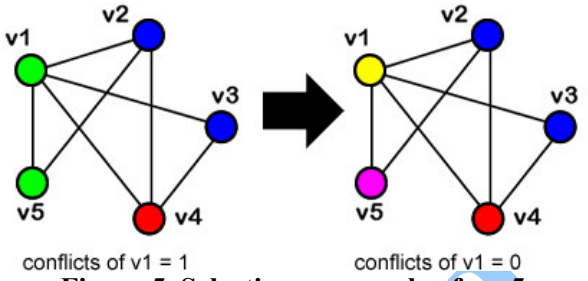

Figure 5. Selecting a new color for v5

When a complete solution is already found, seeking mode would now optimize the solution by lessening the number of colors used. This is done by choosing a vertex from a graph and just copying the color assigned to it. This means that colors which are more commonly used in the graph has a higher probability of being selected and eventually they will replace the less commonly used colors thus reducing the number of colors used as illustrated in Figure 6. Since blue is more commonly used it had a higher probability of being selected as the new color of v4.

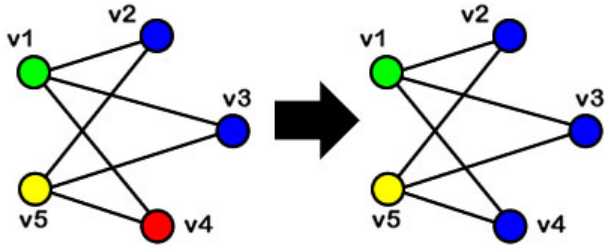

Figure 6. Selecting a new color for v4

To better help in finding a complete solution quickly and also to maintain the completeness of a solution while reducing the number of colors a color change restriction is imposed. In color change restriction, the color update is restricted in such a way that only changes which would result in better colorings or lesser conflicts are allowed. This is done by counting the

number of conflicts a certain vertex has before and after the color change. If the change resulted to more conflicts then it will be reverted otherwise it will be retained. Figure 7 shows a reverted color change while Figure 8 shows a retained color change.

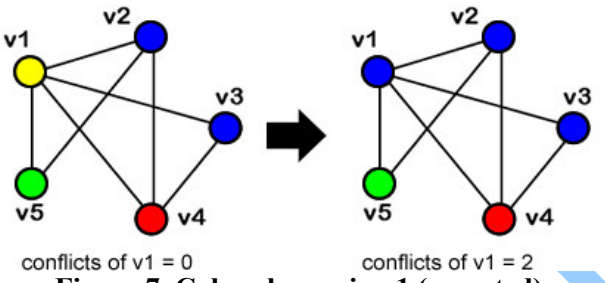

Figure 7. Color change in v1 (reverted)

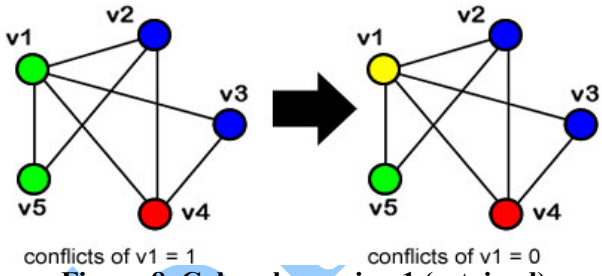

Figure 8. Color change in v1 (retained)

3.2. The Chromaticats Tracing Mode Process

The process in seeking mode ensures finding an equal or better solution from the current solution because of the color change restriction it imposes however this also presents a problem of being trapped in local optima solutions. To avoid such situations tracing mode is implemented without any restriction thus it is free to explore other solutions even if the solutions found introduce regression. Aside from exploring other solutions, tracing mode also helps in reducing the number of colors used by using a color selection similar to seeking mode when a complete solution is already found.

3.3. Fitness Functions

In order to assess the effects of varying the fitness function in the efficiency of CSO, four (4) fitness functions are selected from previous researches [AB08][CO08][HW87][J+91]. The four (4) fitness functions used are as follows.

1. Fitness Function 1

$$fit(g) = \frac{1}{1 + \Sigma\left(\left|\Sigma_{i=1}^{k} V_i\right|\right)} \quad (4)$$

where $E(V_i)$ is the set of conflicts in the *i*th color class. A color class is simply a group of vertices assigned with a specific color.

2. Fitness Function 2

$$fit(g) = \frac{1}{1 + \left(\frac{1}{q(g)} + \left(\left(\frac{V_{constrained}}{c_g}\right)\right)\right)} \quad (5)$$

$$q(g) = \frac{V_g - V_{constrained}}{c_g} \quad (6)$$

where $V_{constrained}$ is the number of constrained edges, C_g is the number of colors used and V_g is the number of vertices in the graph.

3. Fitness Function 3

$$f(g) = \frac{V_g - V_{constrained}}{c_g + V_{constrained}} \quad (7)$$

where $V_{constrained}$ is the number of constrained edges, C_g is the number of colors used and V_g is the number of vertices in the graph.

4. Fitness Function 4

$$f(s) = \sum_{i=1}^{k} |C_i|^2 - \sum_{i=1}^{k} 2|C_i||E(C_i)| \quad (8)$$

where C_i is a color class and $E(C_i)$ is the set of conflicts in the specific color class.

3.4. Output

The output of the program is the best coloring found by the algorithm. Figure 9 shows a sample output.

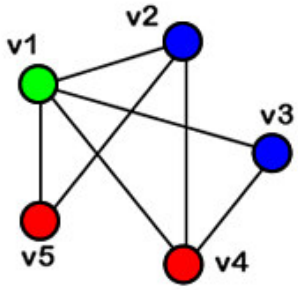

Figure 9. A sample output

4. Experiments and Results

The experiments were conducted with the goal of testing the efficiency of CSO in solving GCP, to assess the effect of varied fitness functions in the efficiency of CSO in finding a solution to GCP and also to compare the results with the performance of Evolutionary Algorithms.

4.1 Experiment Setup

The system used for experimentation has the specifications described below.

Hardware: Intel Pentium(R) Dual-Core CPU T4200 2.0 GHz
1024 MB of RAM

Software: Microsoft Windows 7 Professional 32-bit (6.1, Build 7600)
Java JDK 1.6.0_10 through NetBeans IDE 6.9.1

The setups used were subjected to 30 runs in order to have statistically adequate results and each run was allowed to execute for a maximum of 100 cycles. Table 1 shows the standard parameter values used for the experiments.

Table 1. Standard CSO Parameter Values

Parameter	Value
Number of Cats	30
SMP	5
SRD	0.20
CDC	0.80
SPC	True
C1	2.0
MR	0.02

In the first setup, 10 random graphs of varied sizes and 50% densities were used. The evaluation was done by computing the percentage of colors removed from the graphs' Brook's values (BV) relative to the number of actual number of colors used (C) in their respective solutions. Equation (9), which was used in [CO08], was used for these computations.

$$\%\ of\ BV\ Removed = \frac{BV - C}{BV} \quad (9)$$

The results in Figure 10 show a relative increase in the efficiency of CSO as the graph size increases. This does not mean however that CSO finds better solutions for larger graphs. Smaller graphs would expectedly yield low values in terms of percentage of Brook's value removed since the difference between their Brook's values and chromatic numbers are relatively small. What the results show is that CSO is able to find solutions using few colors even as the graph size is increased.

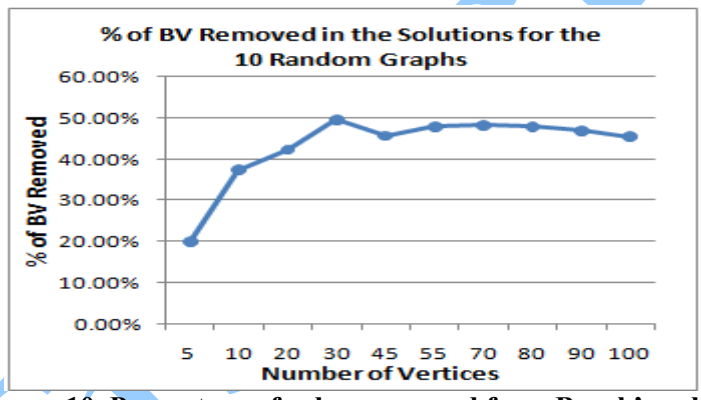

Figure 10. Percentage of colors removed from Brook's value

The behavior of CSO in terms of average fitness value per cycle on the 30 runs is also shown in Figure 11 and it indicates that the algorithm converges to the global best solution in a relatively few cycles. This behavior supports the claim of CSO that it has improved performance in finding the global best solution quickly.

In another setup, the efficiency of CSO in terms of finding optimal solutions to GCP is tested. For this experiment special graphs are used since the chromatic numbers of these graph types are known thus a quick verification of the optimality of a solution can easily be made. In the following results, we use another equation used in [CO08] to assess the optimality of the solutions found by CSO. The equation computes for the percentage of colors used in excess of the graphs chromatic number. The

equation used is shown in equation (10) where *chrom* is the chromatic number of the graph and *C* is the number of colors used in the solution.

$$\% \, Error = \frac{|chrom - C|}{chrom} \qquad (10)$$

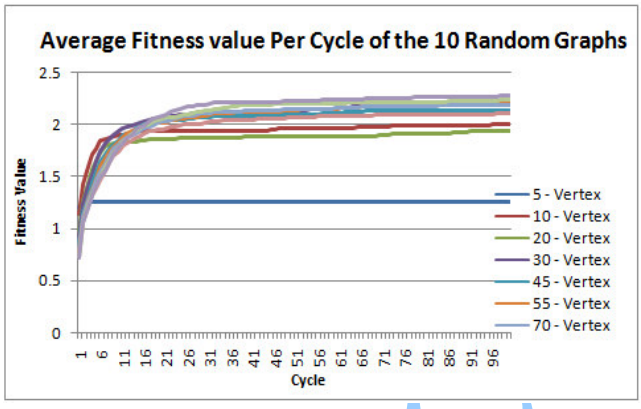

Figure 11. Average fitness value of the 10 test graphs

Table 2. Results for Cycle graphs

Graph Size	No. of Colors Used	Chromatic Number	% Error
25	3	3	0%
50	2.96	2	48%
75	3	3	0%
100	3	2	50%

Table 3. Results for Path graphs

Graph Size	No. of Colors Used	Chromatic Number	% Error
25	2.33	2	16.5%
50	2.96	2	48%
75	2.96	2	48%
100	3	2	50%

Table 4. Results for Bipartite graphs

Graph Size	No. of Colors Used	Chromatic Number	% Error
25	2	2	0%
50	2	2	0%
75	2	2	0%
100	2	2	0%

Table 5. Results for Complete graphs

Graph Size	No. of Colors Used	Chromatic Number	% Error
25	25	25	0%
50	50	50	0%
75	75	75	0%
100	100	100	0%

Based on the results on Table 2 and 3, even cycle and path graphs were the hardest graph types to find for an optimal solution. These results can be explained by looking at the structure of the graph. These graph types have very sparse connections between vertices. These connections serve as the constraints in which the color selection for a certain vertex is based upon. Because of the connections between vertices is insufficient, the color of one vertex has little or no effect at all to the color of another vertex.

For odd cycles and complete graphs, Table 2 and 5 showed 100% success in finding the optimal solutions. This is because their chromatic numbers are equal to their Brook's values thus finding an optimal solution is actually just equivalent to finding the complete solution. Lastly for bipartite graphs, 100% success rate was also obtained in optimizing the solutions. Although the chromatic number is also 2 like in the case of even cycle and path graphs, the relatively denser connections between vertices was sufficient enough to constrain the color selection of a certain vertex in the graph in order to minimize the overall number of colors used.

In [CO08], similar experiment setups were performed to evaluate the performance of the Cultural algorithm or CA in solving GCP and the results are compared in the following tables and figures.

As can be observed and Figure 12, CSO shows a relative increase in efficiency as the graph size increases on the other hand CA produced better results on the smaller graph instances but showed a decline in performance with increased number of vertices.

In terms of running time, the comparison results are graphed in Figure 13. The results show a big increase in the time CSO takes to solve for the solution of the test graphs compared to CA. However this can be explained because CSO has a longer process per cycle especially in seeking mode. Moreover, although CSO takes more computational time to finish one cycle, the extra time helps in finding the global best solution faster as shown in Figure 13.

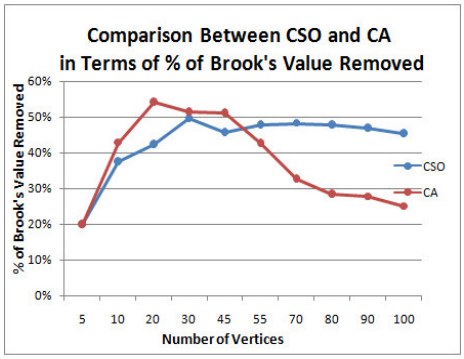

Figure 12. Performance comparison between CSO and CA in increasing graph sizes

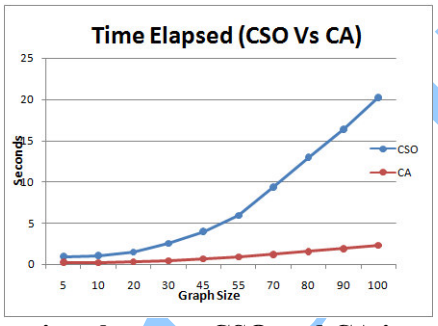

Figure 13. Comparison between CSO and CA in terms of run time

Comparisons in terms of special graphs were also made and the results are tabulated as follows.

Table 6. Comparison with CA using Cycle Graphs

Graph Size	Chromatic Number	# Colors Used (CSO)	# Colors Used (CA)
25	3	3	3
50	2	2.96	3
75	3	3	3
100	2	3	3

Table 7. Comparison with CA using Path Graphs

Graph Size	Chromatic Number	# Colors Used (CSO)	# Colors Used (CA)
25	2	2.33	3
50	2	2.96	3
75	2	2.96	3
100	2	3	3

Table 8. Comparison with CA using Bipartite Graphs

Graph Size	Chromatic Number	# Colors Used (CSO)	# Colors Used (CA)
25	2	2	3.6
50	2	2	11.8
75	2	2	23.2
100	2	2	39.8

Table 9. Comparison with CA using Complete Graphs

Graph Size	Chromatic Number	# Colors Used (CSO)	# Colors Used (CA)
25	25	25	25
50	50	50	50
75	75	75	-
100	100	100	-

In testing for optimality using cycle and path graphs, Tables 6 and 7 showed no significant change between the two algorithms except for the case of the smaller graph sizes. Nonetheless both algorithms experienced the same problem with the structure of even cycle and path graphs.

Using complete graphs, although CA had no problem finding for the optimal solution in the smaller graph sizes, it was not able to find any complete solution for the 75-vertex and 100-vertex graph while CSO found optimal solutions for all test cases. These results can be seen in Table 9.

Bipartite graphs yielded the most significant difference between CSO and CA. As seen in Table 8, CSO had a consistent 100% success in finding optimal solutions whereas CA had difficulty finding one even for the small graph sizes.

A comparison was also made with Evolutionary Programming or EP which was used to solve GCP in [AB08]. The setup used special graphs of small sizes and the algorithms were made to run for an indefinite number of cycles until an optimal solution is found. Table 10 shows the results of the experiment.

The results show that CSO performed considerably better on the odd cycle and complete graphs. Although EP performed better on the even cycle and path graphs the difference was not that significant and CSO also performed faster on all test cases.

Table 10. Comparison between CSO and EP

Graph Type	Graph Size	No. of Cycles Used / Time Elapsed (ms) (CSO)	No. of Cycles Used / Time Elapsed (ms) (EP)
Cycle	20	3.03 / 103	2.4 / 312
Cycle	21	39.16 / 482	193.8 / 2978
Path	20	34.2 / 443	28.6 / 684
Complete	20	3.76 / 130	19.4 / 974

The last experiments were performed to assess the effect of a fitness function in the efficiency of an algorithm in finding optimal solutions. Four fitness functions were used in 4 random graphs of varied sizes. The results exhibited negligible differences among the last 3 fitness functions while unsatisfactory results were produced by fitness function 1. The results are shown in Table 11.

Table 11. Comparison between the 4 fitness functions used

	Function 1	Function 2	Function 3	Function 4
Mean	53.50	44.80	44.60	44.76
Median	53	45	45	45
Mode	52	45	45	45
Max	65	47	47	48
Min	47	42	42	42
SD	4.25	1.34	1.32	1.47
Var	18.12	1.82	1.76	2.18

The first fitness function was later observed to lack the necessary consideration for the number of colors used in a solution thereby explaining the results. Fitness function 1 and 2 shares the same variable, C_g which counts the number of colors used in a solution while fitness function 4 considers the sizes of the color classes. Larger color classes equates to less colors used since having a large color class means having more vertices being assigned to that color. This shows that the fitness function used has a great effect in the efficiency of CSO in solving GCP. The number colors used in the 30 runs also shows the same inconsistent behavior for fitness function 1 as seen in Figure 13.

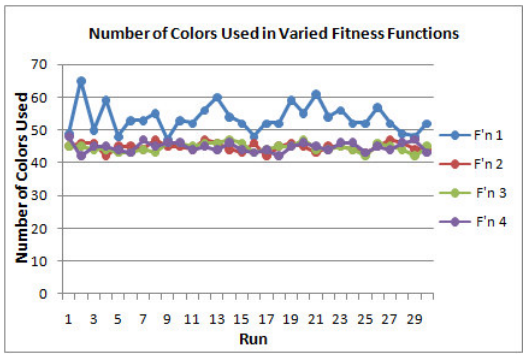

Figure 13. Number of colors used in different fitness functions

Graphs showing the average number of colors used per cycle in the 30 runs conducted are also given in Figures 14 to 17 for the 25, 50, 75 and 100-vertex graphs respectively. The graphs further support the inconsistency on fitness function 1 in generating solutions while the other 3 still showed almost overlapping graphs due to almost negligible differences. Also the graphs for the 3 fitness functions show that CSO consistently converges on the global best solution in the early cycles. This further supports the efficiency of CSO in finding the global best solutions in relatively few cycles.

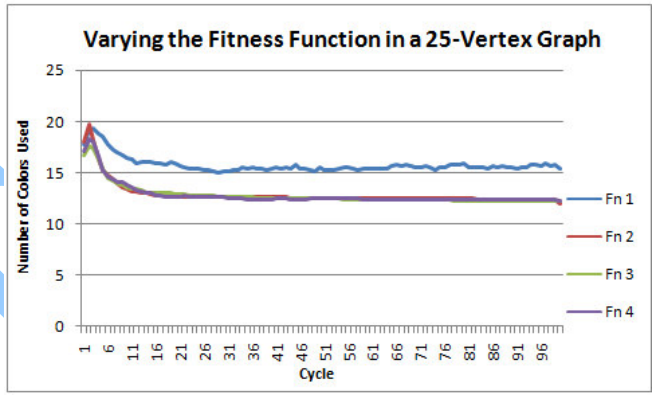

Figure 14. Number of colors used per cycle in a 25-vertex graph

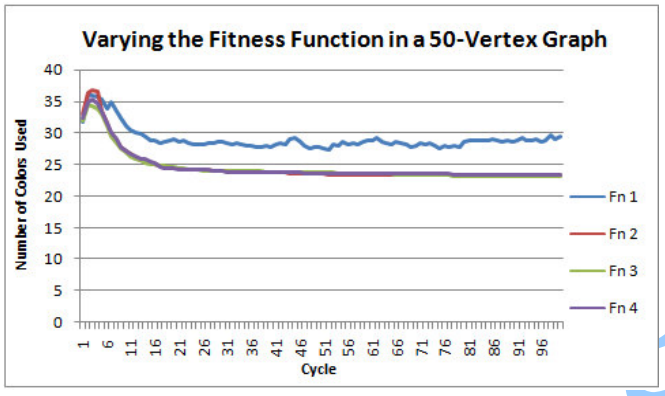

Figure 15. Number of colors used per cycle in a 50-vertex graph

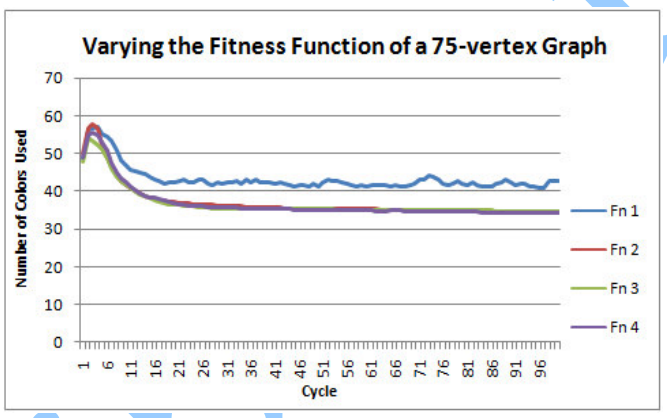

Figure 16. Number of colors used per cycle in a 75-vertex graph

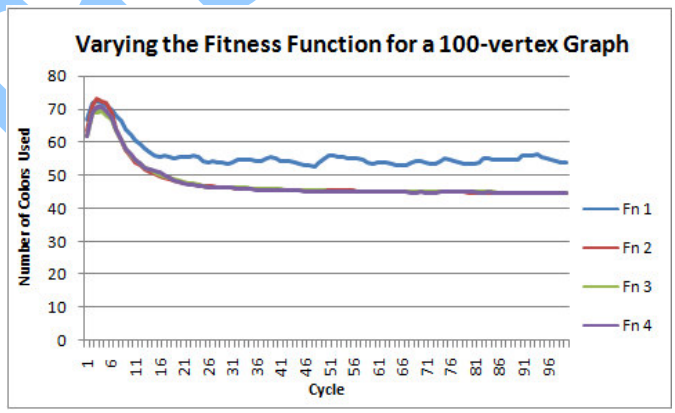

Figure 17. Number of colors used per cycle in a 100-vertex graph

Conclusions

In this research, CSO was used to find for optimal solutions to the popular graph coloring problem of GCP. CSO takes more computational time to finish one cycle. However, its solutions converge to the global best solution in a fewer number of cycles.

This paper also showed that fitness functions greatly affect the algorithm in solving optimization problems. To have satisfactory results the fitness function chosen needs to consider all aspects of the optimization problem that it has to evaluate.

References

[AB08] **J.A.P. Arellano. A.L.L. Barreta** - *EPGraph: Evolutionary Programming for Graph Coloring Problem*, Undergraduate thesis, University of the Philippines Visayas Tacloban College, 2008.

[CPT06] **S.C. Chu, J.S. Pan, P.W. Tsai** - *Cat Swarm Optimization*, in Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence, LNAI 4099, 854-858, 2006.

[CT07] **S.C. Chu, P.W. Tsai** - *Computational Intelligence Based on the Behavior of Cat*, International Journal of Innovative Computing, Information and Control, vol. 3 no 1, 163-173, 2007.

[CO08] **A.P. Cruda, Jr., C.J. Orias** - *CAGraph: Cultural Algorithm for Graph Coloring*, Undergraduate thesis, University of the Philippines Visayas Tacloban College, 2008.

[DAK08] **S. Das, A. Abraham, A. Konar** - *Swarm Intelligence Algorithms in Bioinformatics*. Studies in Computational Intelligence (SCI) 94, 113-147, 2008.

[FJ11] **M. Faraji, H.H.S. Javadi** - *Proposing a New Algorithm Based on Bees Behavior for Solving Graph Coloring*, Int. J. Contemp. Math. Sciences, vol. 6, no 1, 41-49, 2011.

[HR10] **A.N. Homerez, R.A.B. Raagas** - *SudoCatS: Cat Swarm Optimization for Finding Optimal Solution of Sudoku Puzzles* Undergraduate thesis, University of the Philippines Visayas Tacloban College, 2010.

[HW87] **A. Hertz, D. de Werra** - *Using Tabu Search Techniques for Graph Coloring*. Computing, vol. 39, 345-351, 1987.

[J+91] **D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon** - *Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning*. Operations Research vol. 39, no 3, 378-406, 1991.

[Mar04] **D. Marx** - *Graph Coloring Problems and their Applications and their Applications in Scheduling*. Periodica Polytechnica Ser. El. Eng. vol. 48, no 1, 11-16, 2004.

[SE08] **E. Salari, K. Eshghi** - *An ACO Algorithm for the Graph Coloring Problem*. Int. J. Contemp. Math. Sciences, vol. 3, no. 6, 293-304, 2008.

[SN09] **B. Santosa, M.K. Ningrum** - *Cat Swarm Optimization for Clustering*, in Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition, 54 -59, 2009.

[She03] **J. Shen** - *Solving the Graph Coloring Problem using Genetic Programming*. Genetic Algorithms and Genetic Programming at Stanford 2003, 187-196, 2003.

[Tve83] **H. Tverberg** - *On Brook's Theorem and Some Related Results*, Mathematica Scandinavica vol. 52, 37-40, 1983.