An empirical comparison of some approximate methods for Graph Coloring

Israel Rebollo-Ruiz and Manuel Graña

Computational Intelligence Group - University of the Basque Country, email: beca98@gmail.com, ccpgrrom@si.ehu.es

Abstract. The Graph Coloring Problem (GCP) is a classical NP-complete problem for which several approximate solution algorithms have been proposed: Brelaz algorithm, simulated annealing (SA), ant colony optimization (ACO). This paper reports empirical results on the GCP over a collection of graphs of some approximate solution algorithms. Among them, we test a recently proposed Gravitational Swarm Intelligence (GSI). Results in this benchmarking experiment show that GSI performance compares well to other methods.

Key words: Graph Coloring, Gravitational Swarm

1 Introduction

The Graph Coloring Problem (GCP) is a classical combinatorial optimization problem which is of NP-complete complexity [10,14,15,20,21]. The GCP consists in assigning a color to the nodes of a graph with the restriction that any pair of nodes that are linked can't have the same color. The chromatic number K is the minimum number of colors needed to color the graph. Classical algorithms to solve GCP are deterministic search algorithms [8,7,2]. Heuristics and random search allow to obtain approximations to the optimal solutions in bounded time. Recent approaches have applied Ant Colony Optimization (ACO) [11], Particle Swarm Optimization (PSO) [13], and Swarm Intelligence (SI) [27,31].

The bee hives [1], ant colonies [12] and flocking birds [9,29,30] are examples of swarms, whose global spatial configuration and dynamics can be interpreted as working in a cooperative way towards solving a problem. In SI models, the emergent collective behavior is the outcome of a process of self-organization, where the agents evolve autonomously following a set of internal rules for its motion and interaction with the environment and the other agents. Intelligent complex behavior appears from simple individual behaviors. An important feature of SI is that there is no leader agent or central control.

Diverse methods have diverse representations of the problem. ACO approaches make a correspondence between colors traveling over the graph and ants. The space for the motion of the agents is the topology defined by the graph, without any physical correspondence. In PSO, agents contain a full solution of the

problem and exploration is made by generating perturbations around known solutions. In SI graph nodes correspond to agents traveling in a space towards a color coded goal.

The remaining of the paper is organized as follows: section 2 reviews the methods used in the comparison. Section 3 describes the generation of test graph instances. Section 4 give experimental results showing the accuracy finding the solution, computational cost measured in algorithmic iteration steps and time in seconds . Finally, section 5 gives some conclusions and our lines for future work.

2 Graph Coloring Problem methods

We have implemented 5 GCP solving methods as described in the literature: Backtracking, DSATUR, Tabu Search, Simulated Annealing and Ant Colony Optimization. These methods have been proved individually to solve the GCP, but we have not find a direct comparison between all of them. We have developed a new algorithm called Gravitational Swarm Intelligence [26] that is included in this comparison, after proving that our algorithm works with the GCP. This algorithm used new methods of optimization in the artificial intelligence field [6,5]. A brief description of each algorithm follows:

1. Backtracking is a greedy but exhaustive algorithm that explores all the search space and always return the optimal solution if it exists. As the GCP is a NP-complete problem we can use backtracking only in small size problems or especial graphs like the mycielsky graphs [22]. This algorithm is deterministic, so always return the same solution for the same graph instance. Backtracking is no useful with medium size or big graphs, because it needs a huge computational time.
2. DSATUR (Degree of Saturation): this algorithm developed by Brlaz [2] is a greedy backtracking algorithm but does not explore exhaustively all the search space. It looks for the biggest clique in the graph and fix the initial number of colors needed to color it. Then starts the search to determine the color of the remaining nodes of the graph. The clique of a graph [3] is a subset of its vertexes such that every two vertexes in the subset are connected by an edge. It will be necessary at least the same number of colors k as the clique degree to color the graph, that is the reason of the algorithm's name "degree of saturation".
3. Tabu Search (TS): it is a random local search with some memory of the previous steps, so the best solution is always retained while exploring the environment [24]. TS needs a great amount of memory to keep the solutions visited, and if the tabu list is big, it will need so much time to search in the tabu list indeed. A full solution of a big problem can imply a lot of data to keep so could be a limitation in the GCP.
4. Simulated Annealing [28]: inspired in the annealing performed in metallurgy, this probabilistic algorithm finds solutions randomly. If a solution is worse than the previous solution it can nevertheless be accepted as the new solution with a certain probability that decreases with a global parameter called

temperature. At the beginning the temperature is big and almost all the solutions are accepted, but when the temperature cools down, only the best solutions are selected. This process allows the algorithm avoid local maximum. This algorithm has a big handicap when applied to solve the GCP, because there are a lot of neighboring states that have the same energy value. Despite this handicap, Simulated Annealing algorithm provides state-of-the-art results for this problem[23].

5. Ant Colony Optimization (ACO): we have build an implementation following [11] where we have $n * n$ ants making clusters around the colors. We have n ants in each of the n vertexes. Each ant is labeled with a randomly selected color, and the color of a vertex is equal to the color of the maximum size group of ants of the same color in this vertex. In each step, the ants that have a different color of the vertexes color moves through the edges to the neighbors. With the exiting ants and the new coming ants, the color of each vertex is again evaluated until the problem is solved.
6. Gravitational Swarm Intelligence (GSI): this algorithm is inspired in the Gravitation physic law of Newton, and the Boids swarm of Reynolds [29]. The gravitation law has been previously used in Swarm Intelligence [25], different from the GSI formulated for GCP in [27] which does not try to mimic exactly a physical system obeying Newton's law. An intuitive description of the algorithm follows. The GSI for GCP consists in a group of agents representing the vertexes moving in a world where the colors are represented as goal locations that exert an attraction to the agents. When an agent arrives at a goal, it can get that goal color and stop moving if there are no other agents than can't have the same color for the GCP definition, called enemies. The flowchart of figure 1 shows the internal logic works of each GSI agent . Initially a random position is selected for each agent. Depending on the position of the agent and the color goals, it moves toward the nearest goal until reaches a position inside the circle around the color goal defined a given radius.This circle is the region of the space where the agents stay still after getting a color. If two agents can't have the same color, we call them enemies. If there are enemies in that goal, the agent try to expel the enemies outside the goal to a random position. An enemy can be expelled if its internal parameter $Confort = 0$. The $Confort$ on an agent inside a goal grows with time. If the $Confort$ of the enemies is greater than zero then the enemy $Confort$ decreases one unit and the agent is expelled to a random position and start again. Otherwise the agent holds the goal color position and stops moving. If all the agents are stopped then the algorithm has solve the problem.

3 Instances of the problem

We have implemented a graph generator to have our own graph families with specific features, that will help to tune the algorithms. Using Kuratowski's theorem [17][18] we have create five families of planar graphs, increasing the number

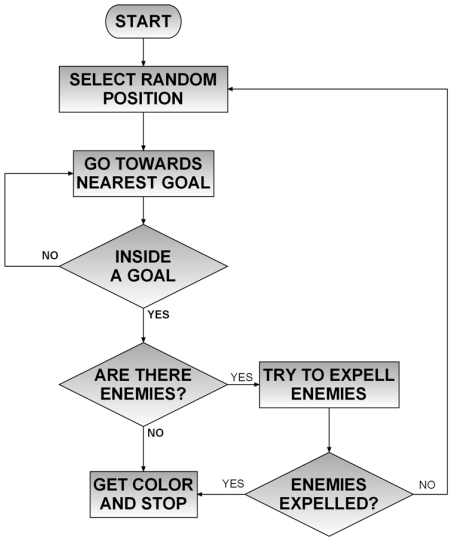

Fig. 1. SI Agent behavior flowchart for GCP

of nodes and vertexes regularly, stating with 50 vertexes and 100 edges and finishing with 250 vertexes and 500 edges. The planar graphs upper bound for the chromatic number is 4 [19]. Kuratowski's theorem is useful to built regular planar graphs, but we have the limitation that we only know the upped bound of the chromatic number, but no the chromatic number.

For validation, it's a good idea to use well-known benchmarking graphs, whose chromatic number is known. For graphs whose chromatic number is unknown the algorithm validation comes from the comparison to other graph coloring algorithms [32,4]. We have test our algorithm in previous works [27] with instances of Mycielsky graphs [22], and the DIMACS graphs [15,16], but in this paper we go further with more explicit problems, and bigger families. In Table 1 we show the features of the graphs used in our test

Table 1. Experimental graph tested features.

Graph name	#nodes	#Edges	Density	K
kuratowski 50x100 (10)	50	100	0.5	4
kuratowski 100x200 (10)	100	200	0.5	4
kuratowski 150x300 (10)	150	300	0.5	4
kuratowski 1200x400 (10)	200	400	0.5	4
kuratowski 250x500 (10)	250	500	0.5	4

We have a total of 50 graph grouped in 5 families. The deterministic algorithms Backtracking and DSATUR has been tested once for each graph (because

are deterministic) and letting them 10^6 steps. The non deterministic algorithms have been tested 30 times for each graph, and letting them 5.000 steps before stopping them, except for the Simulated Annealing algorithm that is faster and lest complex so we let it 50.000 steps.

4 Experimental results

We have made experiments with randomly generated graphs generated. We have implemented all the algorithms using Visual Basic .Net, thus building a GCP suite that will be made public for independent validation of our claims. The graph generator is also been included in this suite to allow other researchers to generate theirs own graphs and solve then with one of the sixth methods. The implementation can be found in `http://www.ehu.es/ccwintco/uploads/b/be/Swarm.rar`.

We have implemented all these algorithm because we desire to perform comparison of the GCP solving methods on new graph instances, instead of using result published in the literature, and also because is difficult to find a working implementation of this algorithms. The programing language, the computer used or even the structures used in the implementation can made a big difference between different works. All the experiments have been run in the same computer.

4.1 GSI implementation

Even though, our algorithm is about SI agents moving around the search space, we haven't use any parallel implementation, even though we claim that our algorithm is scalable. At each time step all the SI agents motion is evaluated. After each time step, the cost function must be evaluated to see if the problem is solved or not. We have two time reference units, the standard hours, minutes and seconds to compare with other algorithms and the iteration steps to compare experiments over the same graph. The real computing time can change from one computer to another, but the steps will be always the same. When we are evaluating the next position of a SI agent in the step t, we take into account the position of the other SI agents in the step $t-1$. All the implementations have been made using the same developing language and trying to use the most homogeneous data structures.

We have arbitrarily defined a 100 x 100 toric world. The goal radius and Comfort parameters have been adjust in order to have better results.. The goal radius has been set to 30 points and the goals have been deployed equidistant in a imaginary circle. The comfort has been set to 4. With this value the algorithm is dynamic enough to get good results. These parameter have been selected empirically. The speed is normalized between [0,1] so is no need to change the agents speed. We have seen that if the goal radius is small and the number of agents is big the convergence is slow, as we expected. The same happens with the comfort. We have demonstrate empirically that our algorithm is scalable because

have almost the same result for the five families. We haven't have exactly the same results for round errors.

4.2 Results

In table 2, 3, 4 and 5 we show a cloud of points corresponding the percentage of success for each method in each graph. Each table there are results of ten graph instances with the same number of vertexes and edges. The deterministic BT and DSATUR algorithms with only 1 execution have a success of 100% or 0 %. The Backtracking and DSATUR algorithm achieved the same result for almost all the instances.

Table 2. Graph coloring results for the kuratowski graph instances of size 50×100.

50x100	BT	DSATUR	SA	TABU	ACO	GSI
K1	100	100	100	80	73	97
K2	100	100	100	90	77	97
K3	100	0	100	47	47	100
K4	100	100	100	80	77	93
K5	100	100	100	60	70	100
K6	100	100	100	87	70	97
K7	0	0	100	67	73	93
K8	0	100	100	50	60	100
K9	100	100	100	43	57	100
K10	100	100	100	77	73	100

Table 3. Graph coloring results for the kuratowski graph instances of size 100×200.

100x200	BT	DSATUR	SA	TABU	ACO	GSI
K1	100	0	100	47	73	97
K2	0	0	100	43	80	97
K3	100	100	100	40	33	97
K4	100	100	100	47	43	90
K5	100	100	100	40	70	100
K6	0	100	100	3	63	97
K7	0	0	100	30	37	90
K8	0	0	100	43	57	100
K9	100	100	100	17	57	97
K10	0	0	100	20	80	93

We can see that GSI algorithm is not the best one in some cases, but is always between the best ones. And when the problem becomes more difficult is the only

Table 4. Graph coloring results for the kuratowski graph instances of size 150×300.

150x300	BT	DSATUR	SA	TABU	ACO	GSI
K1	0	0	90	7	23	87
K2	0	0	57	7	30	80
K3	100	100	100	37	57	93
K4	0	0	83	3	30	90
K5	0	0	100	13	30	93
K6	0	0	100	17	23	93
K7	0	0	93	7	23	90
K8	0	0	100	0	57	87
K9	0	0	100	0	30	97
K10	0	0	100	10	57	93

Table 5. Graph coloring results for the kuratowski graph instances of size 200×400.

200x400	BT	DSATUR	SA	TABU	ACO	GSI
K1	0	0	20	3	30	97
K2	0	0	37	3	17	90
K3	0	0	23	0	0	93
K4	0	0	100	3	7	83
K5	0	0	87	0	23	93
K6	0	0	27	0	30	83
K7	0	0	100	0	17	90
K8	100	100	77	3	0	87
K9	0	0	57	3	7	87
K10	0	0	100	0	7	93

method that still obtains good results. The GSI algorithm has a high level of scalability and for this reason the size of the problem doesn't affect too much to the results. The SA algorithm is the best for small instances but it's accuracy decreases with the size of the problem. The ACO like our GSI algorithm should had a stable behavior, but get poor results, and the size of the problem affects it behavior. Other problem with ACO is the computation time requirements, when the graph size grows the time consumed increases very fast, making this ACO implementation useless for big graphs. Finally, the TS gets very bad results in almost all the situations and it also very expensive in computation time.

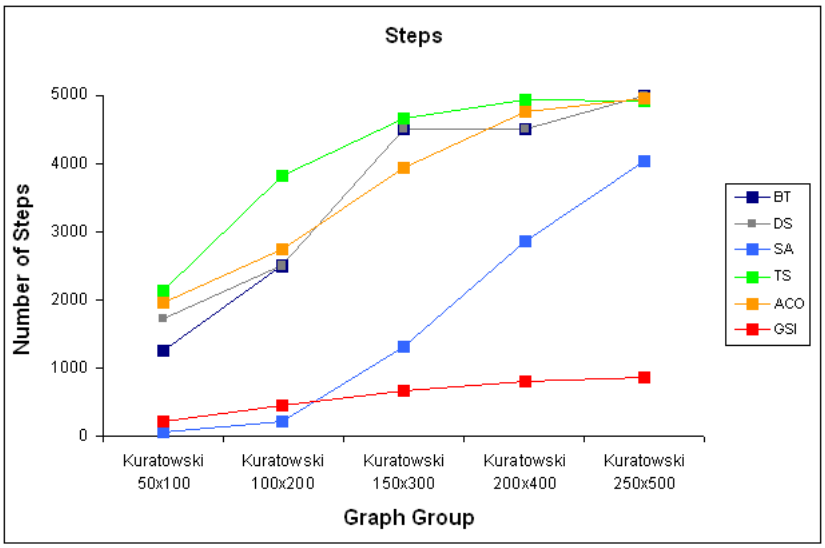

Fig. 2. Average time evolution between families in steps.

Figure 2 shows the average evolution time in steps of the experiment for each graph family and method. We have normalized the number of steps to 5.000 and show them graphically. We can see that SA is the fasted in small instances, but GSI is the fastest with a big difference when the problem start to grow. The Backtracking, DSatur, Tabu Search and ACO algorithms needs about the same number of steps for each graph family. The steps need to solve the problem increases very fast when the problem went difficult. It appears from this experiment that GSI provides good approximate solutions in linear time.

Even though we always are speaking about steps instead of time in seconds,the ACO method is too slow. Among the other five, SA and GSI are the fastest. In figure 3 we can see the evolution of average time in seconds of the experiments grouped by graph instance size. For a clear vision of the results, we have set a maximum value of the ordinate axis of 40 seconds, saturating the plot when the algorithm time goes over this number. The ACO needed more than

600 seconds for the two biggest families. The GSI needs little time, but more than the SA. This is because, even though our algorithm is scalable, the implementation haven't take into account this important feature that would make the GSI the fastest algorithm.

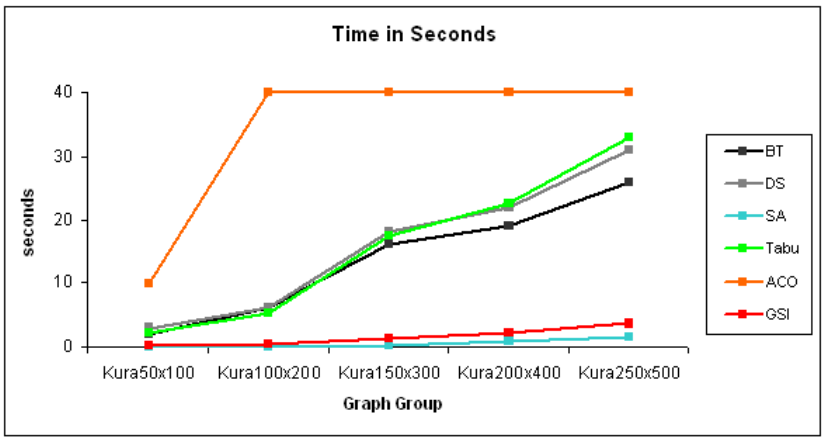

Fig. 3. Average time evolution between families in seconds.

5 Conclusions

We have build a GCP suite for testing six different GCP solution methods. We have added to this suite two graph generators, one for regular planar graphs and other for hard 3-color-able graphs. We have tested all the methods with groups of graphs of increasing size.

We have seen that the stochastic Simulated Annealing is the fastest and the most successful method for small graphs. When the size of the graphs grows, SA starts to have problems to find a solution, but is remains the fastest method. If we haven't been comparing graph with a strict restriction of time and stooped, we will have achieved better results. The GSI approach is among the best methods for small graph and is the best for big graphs, because of its scalability features. The ACO algorithm is very slow and has obtained quite poor results. The Tabu Search it the worst algorithm for this problem and the most time consuming.

Future work will be directed to test the approaches on the Mizuno's graphs [21] and also a bigger group of graph families to continue comparing. We also want to seek for other methods of coloring to add to our suite.

References

1. Akay, B., Karaboga, D.: A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*, In Press, Corrected Proof:–, 2010.
2. Brelaz, D.: New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, April 1979.
3. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16:575–577, September 1973.
4. Chvatal,V.: Coloring the queen graphs, 2004. Web repository (last visited July 2005).
5. Carvalho, A., Corchado, E., Abraham, A.: Hybrid intelligent algorithms and applications. *Information Sciences*, pages 2633–2634, 2010.
6. Wozniak. M, Corchado, E., Graa, M.: New trends and applications on hybrid artificial intelligence systems. *Neurocomputing*, 75:61–63, 2012.
7. Corneil, D. G., Graham, B.: An algorithm for determining the chromatic number of a graph. *SIAM J. Comput.*, 2(4):311–318, 1973.
8. Dutton, R. D., Brigham, R. C.: A new graph colouring algorithm. *The Computer Journal*, 24(1):85–86, 1981.
9. Folino, G., Forestiero, A., Spezzano, G.: An adaptive flocking algorithm for performing approximate clustering. *Information Sciences*, 179(18):3059 – 3078, 2009.
10. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. *Comput. Oper. Res.*, 33(9):2547–2562, 2006.
11. Ge, F., Wei, Z., Tian, Y., Huang, Z.: Chaotic ant swarm for graph coloring. In *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, volume 1, pages 512 –516, 2010.
12. Handl, J., Meyer, B.: Ant-based and swarm-based clustering. *Swarm Intelligence*, 1:95–113, 2007.
13. Hsu, L., Horng, S., Fan, P.: Mtpso algorithm for solving planar graph coloring problem. *Expert Syst. Appl.*, 38:5525–5531, May 2011.
14. Johnson, D. S., Aragon, C. R., McGeoch, L. A., Schevon, C.: Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.
15. Johnson, D. S., Trick, M. A.: *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26. American Mathematical Society, 1993.
16. Johnson D. S., Trick, M. A.: *Proceedings of the 2nd DIMACS Implementation Challenge*, volume 26. American Mathematical Society, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science.
17. Kuratowski, K.: Sur le problme des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.
18. Kuratowski, K.: A half century of polish mathematics: Remembrances and reflections. Oxford, Pergamon Press, 1980.
19. Luzar, B., Skrekovski, R., Tancer, M.: Injective colorings of planar graphs with few colors. *Discrete Mathematics*, 309(18):5636 – 5649, 2009.
20. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. *INFORMS Journal On Computing*, 8(4):344–354, 1996.
21. Mizuno, K., Nishihara, S.: Constructive generation of very hard 3-colorability instances. *Discrete Appl. Math.*, 156(2):218–229, 2008.
22. Mycielski, J.: Sur le coloureage des graphes. *Colloquium Mathematicum*, 3:161–162, 1955.

23. Nolte, A., Schrader, R.: Simulated annealing and graph colouring. *Comb. Probab. Comput.*, 10:29–40, January 2001.
24. Porumbel, D. C., Hao, J.,Kuntz, P.: A search space cartography for guiding graph coloring heuristics. *Computers & Operations Research*, 37(4):769 – 778, 2010.
25. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: Gsa: A gravitational search algorithm. *Information Sciences*, 179(13):2232 – 2248, 2009.
26. Rebollo, I., Graa, M.: Further results of gravitational swarm intelligence for graph coloring. In *Nature and Biologically Inspired Computing*, 2011.
27. Rebollo, I., Graa, M.: *Gravitational Swarm Approach for Graph Coloring*, volume 387 of *Studies in Computational Intelligence*:159 – 168. Springer-Verlag, 2011.
28. Rebollo, I., Grana, M., Hernandez, C.: Aplicacion de algoritmos estocosticos de optimizacion al problema de la disposicion de objetos no-convexos. *Revista Investigacion Operacional*, 22(2):184–191, 2001.
29. Reynolds, C. W.: Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, pages 25–34, 1987.
30. Reynolds, C. W.: Steering behaviors for autonomous characters, 1999.
31. Sundar, S., Singh, A.: A swarm intelligence approach to the quadratic minimum spanning tree problem. *Information Sciences*, 180(17):3182 – 3191, 2010.
32. Turner, J. S.: Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9(1):63 – 82, 1988.