Gravitational Swarm approach for Graph Coloring

Israel Rebollo Ruiz and Manuel Graña Romay

Abstract .

We introduce a new nature inspired algorithm to solve the Graph Coloring Problem (GCP): the Gravitational Swarm. The Swarm is composed of agents that act individually, but that can solve complex computational problems when viewed as a whole. We formulate the agent's behavior to solve the GCP. Agents move as particles in the gravitatory field defined by some target objects corresponding to graph node colors. Knowledge of the graph to be colored is encoded in the agents as friend-or-foe information. This algorithm improves a previous one inspired in Reynolds' Swarm Intelligence (SI) agents. We prove the convergence of the algorithm and test it over well-known benchmarking graphs, achieving good results in a reasonable time.

1 Introduction

The Graph Coloring Problem (GCP) is a classical NP-hard problem which has been widely studied [17, 19, 10, 22, 23]. There are a lot of algorithms like [2, 8, 6] about it, some of them using Ant Colony Optimization (ACO) [11] and Particle Swarm Optimization (PSO) [15]. Up to our knowledge, Swarm Intelligence approaches have not been applied to this problem. The GCP consist in assigning a color to the vertices of a graph with the limitation that a pair of vertices that are linked cannot have the same color.

Israel Rebollo Ruiz
Computational Intelligence Group- University of the Basque Country, e-mail: beca98@gmail.com

Manuel Graña Romay
Computational Intelligence Group- University of the Basque Country e-mail: ccpgrrom@gmail.com

We introduce a new nature inspired strategy to solve this problem following a Swarm Intelligence (SI) [27] approach. The bees [1], ants [13] and flocking birds [9, 26, 25], form swarms that can be interpreted as working in a cooperative way. 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. One of its biggest advantage is that it allows a high level of scalarity, because the problem to be solved is naturally divided into small problems, one for each agent. In real life, when some ants of a colony (also valid for bees, birds or other swarms) fail in its task, it would not alter too much the behavior of the overall system, and in some problems occurs the same, so the algorithm based on SI can be robust against individual failure.

A technique very close to SI is Particle Swarm Optimization (PSO) [7]. The PSO agents, the particles, have complete knowledge of the problem statement and incorporate a specific solution each, having memory of the best position visited in the search space, so they move in the neighboring of that position. Therefore, PSO performs a random population-based search in the solution space using swarm coherence behavior rules.

The work on GCP presented in this paper is an improvement of the approach published in [3, 12]. As in the previous works, SI agents correspondend to graph nodes, and the SI agents try to approach specific space places (goals) where the corresponding graph node acquires a color. We make correspond of the graph edges with antagonist relations between agents. However, in this paper we model the agent's behavior by a simplified dynamic model, removing some unnecesary complexity.

We place the SI agents in a search space with a torus shape, moving towards the color goals. The only behavior rule that we use is the attraction of the goals exerted on the SI agents. We model the attraction to a color goal as a gravitatory field, extended to the entire environment space, which is a fundamental departure from the Reynolds model. When the SI agent reaches a goal, it remains there. The SI agents know the friend/foe relation between them. When a foe SI agent approaches a goal, the antagonistic agents may be expelled from goal if the approaching agent has enough discomfort pressure. This discomfort reaction allows the system to escape local minima that are not solutions of the GCP. We will demonstrate that with this simple rule, our algorithm can solve the GCP with good performance.

The rest of the paper is organized as follow: section 2 presents our Gravitational Swarm Intelligence algorithm, and three ways of application of the algorithm, supervised, unsupervised and a mix of both. In Section 3 we discuss the convergence of the algorithm. In Section 4 we show experimental results comparing our algorithm performance with other methods using well-

known graphs. Finally, section 5 gives some conclusions and lines for future work.

2 Gravitational Swarm Intelligence

The agent's world is a toric surface where the SI agents move attracted by the color goals. If all the SI agents are in a color goal the algorithm stop and the problem is solved, if not, they have to continue moving thorugh the world. When a SI agent reaches a goal it must be sure that there are no enemy SI agents in that goal. We call enemies the SI agents connected by an arc in the underlying graph. Let be $G = (V, E)$ a graph with V vertices and E edges. We define B as a group of SI agents $B = \{b_1, b_2, ..., b_n\}$ where n is the number of vertices. Each SI agent b_i is represented by a position $p_i = (x_i, y_i)$ in the search space and a speed vector $\overrightarrow{v_i}$. Let k be the number of colors or goals that solve the problem, denoted $C = \{1, 2, ..., k\}$, and with an attraction speed $\overrightarrow{v_{gc}}$ for each color.

We can model the problem as a tuple $F = \{B, \overrightarrow{v}, C, \overrightarrow{v_g}\}$ where B is the group of SI agents, $\overrightarrow{v}$ the speed vector in the instant t, C the chromatic number of the graph and $\overrightarrow{v_g}$ the attraction to the goal.

The cost function is:

$$f = |\{\forall i, Cb_i \in C \; and \; \nexists j \mid enemy(b_i, b_j) \wedge Cb_i = Cb_j\}|$$

This energy is the count of the graph nodes which have a color assigned. The predicate *enemy* is true if a pair of SI agents have an edge between them, so they cannot have the same color. Cb_i denotes the color of the SI agent associated with the target goal. A SI agent has a color only if the SI agent is at distance below *nearenough* from a goal. Until the SI agent arrives to a goal, its color cannot be evaluated. The definition of *nearenough* allows to control the speed of the algorithm: if it is small, the algorithm converges slowly but monotonically to the solution, if it is big, the algorithm is faster but it's convergence is jumpy because the algorithm falls in local minima and needs transitory energy increases to escape them. The dynamic of the SI agents in the world is specified by the iteration:

$$\overrightarrow{v_i}(t+1) = (1 - \frac{d}{\|d\|}) \cdot (\overrightarrow{v_i}(t) + d \cdot (\overrightarrow{v_g} \cdot (1 - \lambda))) + (\lambda \overrightarrow{v_r} \cdot p_r), \qquad (1)$$

where d is the difference vector between the agent's position and the position of the nearest color goal, $\overrightarrow{v_g}$ represents the speed to approach to the nearest goal in Euclidean distance from current position p_i, $\overrightarrow{v_r}$ is a noise vector to avoid being stuck in spurious unstable equilibrium, and p_r is a random position. Parameter λ represents the degree of Comfort of the SI agent. When a SI agent b_i reaches to a goal in an instant t, its speed becomes 0. Every

step of time that the SI agent stays in that goal without been disturbed, its Comfort increases, until reaching a maximum value $maxconfort$. When an enemy SI agent outside the goal tries to go inside that goal, the Comfort of the SI agent inside the goal decreases to 0. When Comfort is positive the parameter λ has value $\lambda = 0$. If Comfort of a SI agent is equal to 0 and enemy tries to get inside the goal then $\lambda = 1$ and the SI agent is expelled from the goal to a random point p_r with speed $\overrightarrow{v}_r$. When all the SI agents stop, $\forall i, \overrightarrow{v_i} = 0$; $f = Cb_i$ the problem is solved.

Each color goal has an attraction well spanning the entire space, therefore the gravitational analogy. The magnitude of the attraction drops proportionally with the Euclidean distance d between the goal and the SI agent, but it never disappears. If d is less than $nearenough$ then we make $d = 0$, and the agent's speed becomes 0 stopping it. This is, again, departs from the Reynolds boid system, where boids never stop moving. Figure 1 shows the interface of the software implementation of this system. In the flowchart of figure 2 we can see how the algorithm works for each SI agent.

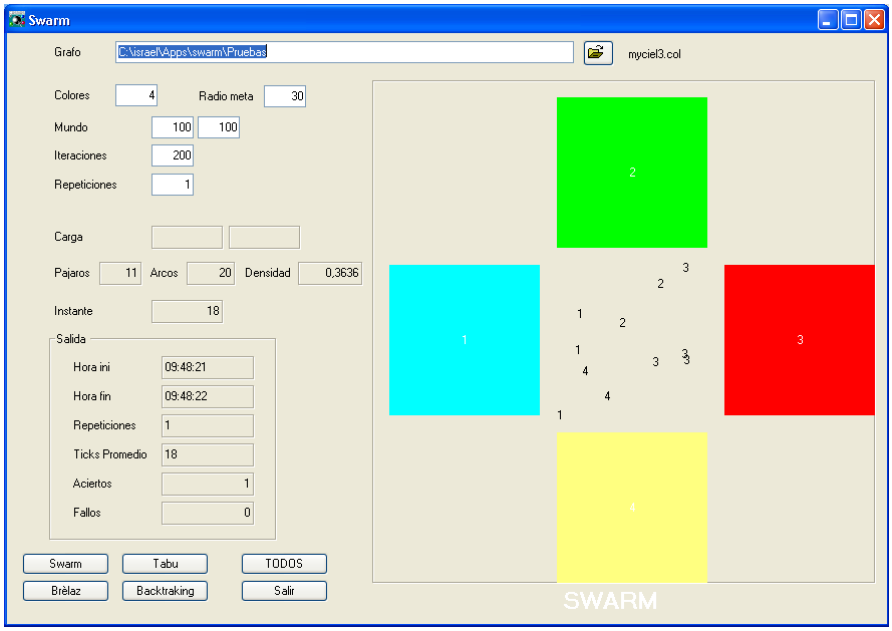

Fig. 1 Snapshot of the software interface of the SI applied to solve the GCP

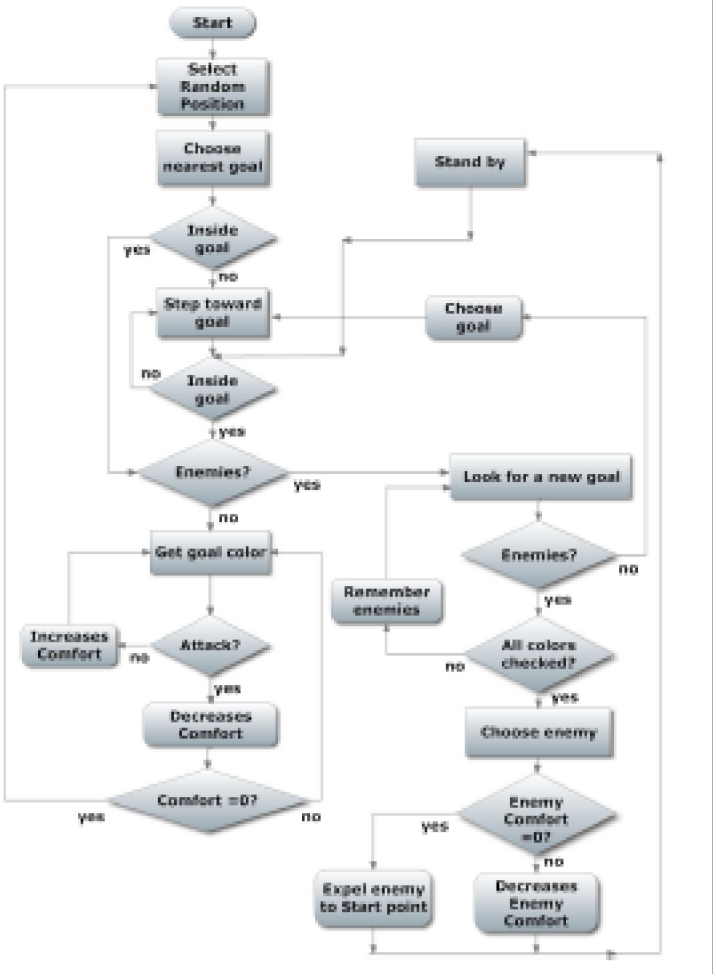

Fig. 2 SI Agent behavior flowchart for GCP

2.1 A Supervised coloring

The algorithm can be thought of as a supervised algorithm in the sense that we provide the number of colors when we specify the number and places of color goals. The chromatic number is the minimum number of colors needed to color the graph. When specifying C color goals we assume that the chromatic number is k or lower. If the algorithm converges then we are right, if not then we assume that the chromatic number is greater than k. Search for the graph's chromatic number can be done starting with a large k and decreasing down to the value until the algorithm fails to converge, or the other way, starting

with low values of k increasing them until the algorithm converges for the first time. If the chromatic number is known, then we can validate our algorithm against this ground truth.

For validation, it's a good idea to use well-known benchmarking graphs, whose chromatic number is known. The Mycielsky graphs [24] are a family of graphs whose chromatic number is equal to the degree of the graph plus one $K = m + 1$ where m is the Mycielsky number. There are also collections of benchmark graphs, such as the DIMACS graphs [18, 19]. For graphs whose chromatic number is unknown the algorithm validation comes from the comparison to other graph coloring algorithms [28, 5].

2.2 Unsupervised coloring

Another way to attack the problem of determining the chromatic number is to remove the color goals and let the SI agents move according to their friend/foe relations. After a long period of time the movement of the SI agents becomes stable, so we can say that the actual configuration as the solution of the problem. The SI agents will never stop, because in the speed function, there is no term to approach a specific place. The solution to the coloring problem would come from the detection of clusters of agents moving together. The attraction to the color goals is substituted by the attraction to friendly agents. Here again we obtain only an approximation to the real solution, because the SI agents can generate a heterogeneous group of clusters. We think that the unsupervised coloring can be used to find a starting upper bound of the chromatic number of a graph, taking into account that there is no stopping condition.

2.3 Coloring waterfall

We can use a mixed strategy to find the solution for the coloring of a graph. First, apply the unsupervised coloring method to estabilishing an upper bound to the problem. Second, in an iterative way, apply the supervised coloring method to look for solutions with lower chromatic number, decreasing the number of goals, until the algorithm cannot stop and give a solution in a given computational time. The pseudo code of the Algorithm 1 specifies this coloring waterfall.

Algorithm 1 Coloring waterfall

```
Input: G the graph to color.
Upper_bound = unsupervised_coloring(G)
while Upper_bound > 1 do
  let Upper_bound = Upper_bound - 1
  solution = supervised_coloring(G,(Upper_bound),max_time)
  if not solution then
     chromatic_number = (Upper_bound+1)
     exit while
  end if
  end while
```

3 Convergence issues

We discuss in this section the convergence of the algorithm from an intuitive point of view. The SI agents start in a position $p_0 = \{x_0, y_0\}$ and with an initial speed $\overrightarrow{v_0}$. The direction and value of the speed vector changes with the dynamics of the system. The value of the SI agent's speed is in the range $[0, 1]$. It depends on the distance between the SI agent and its target color goal, when the distance is below a threshold the SI agent stop. Initially each SI agent tries to get to the nearest goal. The attraction of the goals is strong when the SI agent is far away and weak when the SI agent is near the goal, becoming 0 when the SI agent is inside the goal radius. If antagonistic SI agents are already inside the goal, it tries to expel them out of the goal. An expelled SI agent tries to go to the second nearest goal and so on, until it can enter in a goal without enemies.

If all the SI agents speed is zero, then the algorithm has converged to some fixed state where all of them are inside a goal. Thus, we have solved a part of the convergence problem, now we have to demonstrate that the color assignment is a solution for the problem. That is, the cost function value is equal to the number of nodes when the system converges.

When a SI agent tries to enter inside a goal, if there is one or more enemies in the goal it will try to find another goal empty of enemies. If there is no one, then it will proceed to expel the enemies from one of the goals. The SI agent selects a random foe and evaluates its λ parameter. If it is zero, then that enemy is expelled from the goal to a random point p_r with a speed vector $\overrightarrow{v_r}$ towards a random goal. All the enemies' Comfort inside that goal decreases. It doesn't matter if other enemies $\lambda = 0$, the SI agent can only expel one SI agent at a step. The SI agent doesn't stop because it can be still more enemies in that goal and must wait until the next step to get inside. With this behavior, when a SI agent is inside a goal, it's sure that there are no enemies in the goal. So when our algorithm stops because all the agents have stopped, it has reached a GCP solution, because no two adjacent nodes are in the same color goal and all the agents are "inside" a color goal. If the

agents never stop, the algorithm is unable to find a solution with the stated number of colors

4 Experimental results

We have implemented our algorithm in a graphic development platform trying to obtain an intuitive visualization of the process rather than an efficient implementation in computational terms. Our algorithm is about SI agents moving around the search space, where each step of time all the SI agents move, or stay stood. After each step of time, the cost function must be evaluated to see if the problem is solved or not. For this reason, the time is always referred as iteration steps. 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 $t-1$.

Each time step is similar to a generation of a genetic algorithm, or a simulated annealing phase [16], because we have to move all the SI agents (all the SI agent with speed > 0), and evaluate the cost function. The real computing time of our algorithm depends on the programming language and the computer used, but the steps will be always the same.

For comparison, we have also executed a backtraking implementation, a DSATUR Brèlaz algorithm and a Tabu Search algoritm implementation but we have a lot of tests and benchmarks in the bibliography like[14] where the Mycielsky graph are tested needing between 4 and 416.000.000 backtrack where our GSI algorithm needs only between 21 and 417 steps or [21] where Leightn and DJS aleatory geometric graphs are tested. In [4]the fullins graph family cannot be solved but our algorithm can.

In table 1 we show the results of applying the greedy algorithms. DSATUR is a particular case of backtracking and the results are similar. Backtracking and DSATUR are fixed algorithm so the result is always the same, we let them 1.000.000 cycles until we stop them. Tabu search is not a fixed algorithm and the results can change from an execution to another, so we have tested 20 times with a maximun number of tabus of 2000, showing the tabus or steps and the % of success. These methods are useless when the graph is medium or big, because greedy algorithms are no good for NP-hard problems, and the Tabú search needs a lot of memory to keep a large amount of no valid solutions.

The * denotes that the algorithm stop without finding a solution.

In table 2 we show results on a group of well known graphs [19, 20] whose chromatic number is known. We have launched the algorithm 20 times for each graph instance and we allow them 2000 steps to find a solution. We show the number of vertices, edges, the density of the graph calculated as #edges/#complete_graph_edges, the graph's chromatic number K, the per-

Graph name	k	#Backtracking	#DSATUR	#Tabu Search	%Tabu success
Myciel3	4	1	1	11	100
Myciel4	5	1	1	34	100
Myciel5	6	1	1	107	100
Myciel6	7	1	1	290	100
Myciel7	8	1	1	597	100
queen5_5	5	5	5	442	95
anna	11	*	*	*	0
david	11	*	*	*	0
huck	11	*	*	*	0

Table 1 Backtracking, DSATUR and Tabu Search results

centage of success over 20 experiments and the average number of steps needed to get the solution.

Graph name	#Vertices	#Edges	Density	K	Success %	Average #Steps
anna	138	986	0.10	11	100	300
david	87	812	0.21	11	100	209
huck	74	662	0.22	11	100	84
jean	80	508	0.16	10	100	165
myciel3	11	20	0.36	4	100	21
myciel4	23	71	0.28	5	100	25
myciel5	47	236	0.21	6	100	97
myciel6	95	755	0.17	7	100	92
myciel7	191	2360	0.13	8	100	417
queen 5x5	25	160	0.53	5	100	302
1_fullins_3	30	100	0.23	4	100	37
1_fullins_4	93	593	0.14	5	100	76
1_fullins_5	282	3247	0.08	6	100	222
2_fullins_3	52	201	0.15	5	100	67
2_fullins_4	212	1621	0.07	6	100	176
miles250	128	387	0.04	8	100	317

Table 2 Experimental results

We can see in table 22 that the algorithm success in all graphs and repetitions of the process. Our algorithm compared with the results in table 11is faster and is can solve all the graphs in a reasonable time. The greedy algorithm are exact and always solve the problem but in an exponetial time.

The number of steps is not clearly related to the size or density of the graphs, because some small graphs require higher number of steps than bigger ones, and some sparse graphs require more steps than dense ones, and the contrary is true also. We need to perform an exhaustive exploration over a collection of random graphs in order to make any inference about time complexity. The fact is that the algorithm requires on average much less steps than the allowed maximum.

5 Conclusions

We proposed a new algorithm for the Graph Coloring Problem using Swarm Intelligence. We have modeled the problem as a collection of agents trying to reach some of a set of goals. Goals represent node colorings, agents represent graph's nodes. The color goals exert a kind of gravitational attraction over the entire virtual world space. With these assumptions, we have solved the GCP using a parallel evolution of the agents in the space. We have argued the convergence of the system, and we have demonstrated empirically that it provides effective solutions in terms of precision and computational time. We will continue to test our algorithm on an extensive collection of graphs, comparing its results with state of the art heuristic algorithms.

For future work, we have to improve our implementation of the algorithm to make it faster. Even though our algorithm finds the global optimum of the problem, we will search for new nature inspired behavior rules to improve the algorithm.

References

1. Bahriye Akay and Dervis Karaboga. A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*, In Press, Corrected Proof:–, 2010.
2. Daniel Brèlaz. New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, April 1979.
3. B. Cases, C. Hernandez, M. Graña, and A. D'anjou. On the ability of swarms to compute the 3-coloring of graphs. In S. Bullock, J. Noble, R. Watson, and M. A. Bedau, editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 102–109. MIT Press, Cambridge, MA, 2008.
4. Marco Chiarandini, Thomas Stützle, Fachgebiet Intellektik, Fachbereich Informatik, and Technische Universität Darmstadt Darmstadt. An application of iterated local search to graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, 2002.
5. V. Chvátal. Coloring the queen graphs, 2004. Web repository (last visited July 2005).
6. Derek G. Corneil and Bruce Graham. An algorithm for determining the chromatic number of a graph. *SIAM J. Comput.*, 2(4):311–318, 1973.
7. Guangzhao Cui, Limin Qin, Sha Liu, Yanfeng Wang, Xuncai Zhang, and Xianghong Cao. Modified pso algorithm for solving planar graph coloring problem. *Progress in Natural Science*, 18(3):353 – 357, 2008.
8. R. D. Dutton and R. C. Brigham. A new graph colouring algorithm. *The Computer Journal*, 24(1):85–86, 1981.
9. Gianluigi Folino, Agostino Forestiero, and Giandomenico Spezzano. An adaptive flocking algorithm for performing approximate clustering. *Information Sciences*, 179(18):3059 – 3078, 2009.
10. Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Comput. Oper. Res.*, 33(9):2547–2562, 2006.

11. Fangzhen Ge, Zhen Wei, Yiming Tian, and Zhenjin Huang. 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. M. Graña, B. Cases, C. Hernandez, and A. D'Anjou. Further results on swarms solving graph coloring. In D. Taniar et al., editor, *ICCSA 2010 Part III*, number 6018 in LNCS, pages 541–551. Springer, 2010.
13. Julia Handl and Bernd Meyer. Ant-based and swarm-based clustering. *Swarm Intelligence*, 1:95–113, 2007.
14. Francine Herrmann and Alain Hertz. Finding the chromatic number by means of critical graphs. *J. Exp. Algorithmics*, 7:10, December 2002.
15. Ling-Yuan Hsu, Shi-Jinn Horng, Pingzhi Fan, Muhammad Khurram Khan, Yuh-Rau Wang, Ray-Shine Run, Jui-Lin Lai, and Rong-Jian Chen. Mtpso algorithm for solving planar graph coloring problem. *Expert Syst. Appl.*, 38:5525–5531, May 2011.
16. Manuel Graña y Carmen Hernández Israel Rebollo. Aplicaciones de algoritmos estocasticos de otimización al problema de la disposición de objetos no-convexos. *Revista investigación operacional*, 22(2):184–191, 2001.
17. D. S. Johnson, A. Mehrotra, and M. Trick, editors. *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, Ithaca, New York, USA, 2002.
18. David S. Johnson and Michael A. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26. American Mathematical Society, 1993.
19. D.S. Johnson and M.A. Trick, editors. *Proceedings of the 2nd DIMACS Implementation Challenge*, volume 26. American Mathematical Society, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science.
20. G. Lewandowski and A. Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. pages 309–334.
21. Zhipeng Lu and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241 – 250, 2010.
22. A. Mehrotra and M. Trick. A column generation approach for graph coloring. *INFORMS Journal On Computing*, 8(4):344–354, 1996.
23. K. Mizuno and S. Nishihara. Toward ordered generation of exceptionally hard instances for graph 3-colorability. pages 1–8.
24. Jan Mycielski. Sur le coloureage des graphes. *Colloquium mathematicum*, 3:161–162, 1955.
25. Craig Reynolds. Steering behaviors for autonomous characters, 1999.
26. Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, pages 25–34, 1987.
27. Shyam Sundar and Alok Singh. A swarm intelligence approach to the quadratic minimum spanning tree problem. *Information Sciences*, 180(17):3182 – 3191, 2010. Including Special Section on Virtual Agent and Organization Modeling: Theory and Applications.
28. Jonathan S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9(1):63 – 82, 1988.