Distributed Graph Coloring

Xavier Barthel, Maxime Godfroid and Quentin-Emmanuel Vajda

ULB

Abstract

The graph coloring problem is to assign a color to each node of a graph such as no neighboring nodes have the same color ; and while maintaining as few colors as possible. The need of efficient algorithms capable of solving this problem is great. In this paper, some of the stat of the art heuristics in both centralized and distributed systems used to solve said problem are summarized. The main subject is the algorithm called *FrogSim* which is based on the behaviour of Japanese tree frogs. This algorithm and its implementation is explained in great details. Data resulting from an array of test performed on both random geometric graphs and grid topologies are also discussed. Finally a few possible improvements thought of during our researches of this subject are presented.

Introduction

Graph Coloring

Graph coloring is a problem involving the coloration of a given graph ; each node of said graph has a certain color assigned to it. A valid coloring is found when all neighboring nodes[1] have different colors.

The graph coloring problem is to verify if a given graph can be colored using k different colors and to find said coloration[2] if it exists.

Based on the above, the k^* minimum number of colors necessary for a given graph is determined and the optimal coloration is found.

Usage One of the most well known usage of non-centralized algorithm to solve graph coloring instances is the channel assignment in wireless networks. The radio emitters are represented as the nodes in the graph. A color corresponds to a radio frequency used by a emitter. the coupling of color with node is needed to reduce the interferences and thus enhance the stability of the network. The topology of wireless networks is often similar to a grid or geometric graph, which are both described further.

[1] All the nodes with which a given node has a common link.

[2] Or a coloration using less than k colors.

Centralized Centralized algorithm to solve the graph coloring problem have existed, been studied and improved for a few decades. The following are some of the most well known of these algorithms.

Deterministic The graph coloring problem is what is called a "*NP-Hard problem*", a non-deterministic polynomial algorithm for which no deterministic algorithm performing with polynomial complexity is known. This leads to the computational infeasibility to solve the problem at hand using a deterministic algorithm.

This does not mean that no deterministic algorithm solving this problem exist however. But it does mean that non of these algorithm are applicable in practice. For this reason, they will not be detailed here.

More information concerning deterministic algorithm ave been detailed by De Marco and Pelc (2001) and Barenboim and Elkin (2011).

(Meta-)Heuristic As discussed previously, the graph coloring problem is *NP-Hard*, which means that there exists non-deterministic algorithm to solve it that will run in polynomial time. These algorithms are often referred to as heuristic or meta-heuristic when used in practice.

Heuristics do not guarantee to find the optimal solution to a problem, nor to always give the same solution to a problem. This come from the random nature of their implementation, which itself comes from the non-determinism of the algorithm it represents.

Some example of heuristics capable of solving the graph coloring problem are presented here under. Most of these algorithm are based on general algorithm capable of solving most *NP* problems ; as such they have common notions :

- A result ; for graph coloring, an association pairing each node to a color.

- A transformation ; when applied to one or multiple result, it will yield a new result. Examples of such transformations in graph coloring could be to simply assign a new random color to a node of the given result, or swap the colors of two randomly picked nodes.

Evolutionary The general evolutionary algorithm is heavily inspired by the evolution of a population as studied in biology. The notions used are the idea that a population will evolve by reproduction,mutation and selection. These are the different transformations applied on a population composed of possible results to the problem.

An application of this heuristic to the graph coloring problem has been proposed by Malaguti et al. (2008).

Tabu Search The very general principle of the Tabu search is to evolve from a valid solution and search for optimization by applying transformations. The idea is then to only apply transformations that will yield the best results and are not *tabu* ; which means that they have not been applied recently.

This general algorithm as been specialized and optimized for the purpose of the graph coloring problem, notably by Blöchliger and Zufferey (2008). The particularity of this heuristic is the decomposition of the global solution allocating a color to each node, in multiple sub-solutions giving colors to only a subset of nodes.

Genetic Local Search Detailed by Dorne et al. (1998), this algorithm is a hybrid approach to the graph coloring problem. Combining the global association of results set provided by the genetic heuristic, and the search for local optimization of the Tabu search, gives good results. In particular for the *DIMACS* benchmarking graphs.

Variable Neighborhood The great performances of the variable space search proposed by Hertz et al. (2008), comes from its unique ideas. The main one being to find local optimizations in specific search spaces where global constraints are not guaranteed and to combine these optimizations to obtain a valid result for the whole problem.

Distributed The graph coloring problem can be resolved in a centralized manner as previously explained. However, in practice most possible uses of graph coloring are in distributed systems.

The distributed approach depends on swarm intelligence where multiple agents all possessing only limited ressources[3] work together to achieve a global goal. In the case of graph coloring problem, each node is a separate agent whose limited resources only allow to communicate with its direct neighbors. The common goal in this scenario is finding the optimal coloring.

Due to the constraints brought by distributed systems, namely the lack of ressources, the algorithms used here are much more specialized. Because this lack of ressources can be of many different nature a given problem can have many different heuristics solving it while optimizing each a different aspect. for instance, a distributed system whose main constraint is its real time nature will have algorithm designed around the idea of minimal time complexity whereas if the main constraint was the low computational power of each agent the same problem would be solved with an algorithm specialized to require less calculations.

The focused attributes of the following will be time complexity, calculation complexity as well as message load[4].

Graph coloring also allows even more specialization because each topology of graphs can have an associated algorithm.

One of the most effective, while still being very general work, was produced by Finocchi et al. (2002). It presents three versions of an algorithm focusing on different constraints of distributed systems.

Another interesting heuristic was presented by Kubale and Kuszner (2002). The DLF (*Distributed Largest-First*) algorithm uses a random ordering of the nodes as well as a partial ordering based on the degree of each node[5] to simplicitate and accelerate the process of picking a valid color for each agent.

This short and non-exhaustive list of different techniques can easily be extended. In fact, any techniques of artificial intelligence can reasonably easily and with varying degrees of success be adapted to fit the problem at hand. However we have here only presented a few chosen techniques that can be compared to the *FrogSim* algorithm.

The heuristic based on the calling behaviour of the Japanese tree frog is the main subject of this paper. The main idea behind it is to emulate the anti-phase synchronization of said frogs to allow each node to discover its color as fast as possible while sending minimal messages. It

[3]Computational ressources, memory, battery ...

[4]The size of the messages exchanged by the different agent. The smaller it is the less time spent transfering information which in turn diminishes the energy required to operate.

[5]The degree of a node being the number of neighbors a node has.

is explained in greater details in the following sections.

This idea has been exploited in the past by Lee and Lister (2008) and later refined by Lee (2010). The *FrogSim* algorithm, presented in the rest of this paper, is another improvement on these two techniques.

Methods

Calling Behaviour Model

This algorithm is based on the calling behaviour of Japanese tree frog. As said by Aihara (2009), it has been observed that in the calling behavior of multiple males frogs interacting together, the anti-phase synchronization is stable and, if it occurs, the in-phase synchronization is transient. The male frogs developped this ability in order to get the female frogs to locate them, which is complicated if their calls are performed too closely.

The Japanese tree frog behaviour is modeled in this algorithm to rank the nodes. Firstly, some terms need to be defined:

round: Time unit during which an event occurs for all the nodes;

phase: Denoted $\theta_i \in [0, 1)$, it specify the moment a particular event occurs for a given node i during the round;

color: The colors are identified by natural numbers. The color of a node i is denoted by $c_i \in \mathbb{N}$.

An event–which occurs for all nodes each round–consists of a computation followed by sending a message containing the current values of the node, resulting from the computation. When the phase θ_i occurs, the node i reads the messages contained in his queue M_i, those messages are sent by its neighbors during the last round and by neighbors having smaller phases. The algorithm proposed by Hernández and Blum (2010) is divided in two phases. The first one uses the model of the calling behaviour of the Japanese tree frog to define a valid coloring. The second phase enhances the coloring returned after the execution of the first phase.

Phase 1

```
if r < self.k:
  if len(self.msgStack) > 0:
    self.newTheta()
    self.minColor()
    self.setRelevance()
  else:
    self.relevance = 1
  self.convergence /= self.rho
  msg = dict(theta=self.theta,
      color=self.color,
      relevance=self.relevance)
```

Implementation 1: Phase 1.

The first phase starts by sending a trigger message containing the value K which specify the number of rounds of the first phase. When an event occurs for a node i, it begins by looking the messages, sent by his neighbors, contained in his queue M_i. A message $m \in M_i$ is defined as follows:

$$m = \langle \theta_m, c_m, relevance_m \rangle$$

Where θ_m is the phase of the sender node, c_m his color and $relevance_m$ his the weight given to the sender node. The $relevance_m$ is proportional to the number of messages received by the sender node when his event occurred, $relevance_m = \frac{1}{|M_i|^2}$. This value quantify the fact that the nodes with a small amount of neighbors should see there θ-values converge faster. A new value for θ_i is based on the following function:

$$\theta_i = \theta_i + \alpha_i \times \sum relevance_m * inc(\theta_m - \theta_i) \qquad (1)$$

Where α_i is used to control the convergence of the system, which can be gradually updated at the end of each node event as it follows $\alpha_i = \alpha_i / \rho$. The $inc()$ is the phase shift function:

$$inc(x) = \begin{cases} x - 0.5 & \text{if } x \geq 0 \\ x + 0.5 & \text{if } x < 0 \end{cases} \qquad (2)$$

This function is different from the model proposed by Aihara (2009) but has proved to have better convergence properties as experimented by Hernández (2012). Then the color is chosen by a node i in order to get the smallest value according to the received messages.

$$c_i = \min \{c \in \mathbb{N} | c \neq c_m \forall m \in M_i\}$$

We can see that the color pick is done due to the phase shift, which creates a time "frame" when a node sees his event occur without knowing it may not received some of its neighbor's messages. Thus when the θ-values have converged, the coloring does not change. Finally, to conclude the execution of this event, the message queue M_i is cleared.

Phase 2

```
if r == self.k:
  if self.color == 1:
    self.power = Node.getPower()
elif len(self.msgStack) > 0:
  self.keepHigherPower()
  self.minColor()
msg = dict(power=self.power,
     color=self.color)
```

Implementation 2: Phase 2.

The second phase is used to improve the color scheme resulting from execution of the first phase. To do so, each node gets a power value during the first execution of this

phase. For a node i, if $c_i = 1$ his $power_i$ value is set to a positive random number and if $c_i > 1$, $power_i = 0$. As said by Hernández and Blum (2010), it must be notice that the θ-value does not change anymore. The principle of this algorithm is pretty simple, a particular node check his message queue and keep the messages coming from the nodes with $power_m \geq power_i$. Then the node i choose the minimal color based on the remaining messages.

$$c_i = \min \{c \in \mathbb{N} | \, (c \neq c_m \wedge power_m \geq power_i) \, \forall m \in M_i\}$$

After that, the node take the maximal power value that it had encountered in his message queue M_i.
And finally, the node clears his message queue and sends a message containing his color and his power value,

$$m = \langle c_m, power_m \rangle$$

Results

Random Geometric Graphs

A random geometric graph is a graph of n nodes built randomly using the following method : for each of the n nodes, randomly assign a position within R^2 where both coordinates are positive and smaller than 1. Then for each nodes, calculate their Euclidian distance to all the other nodes. Each pair of nodes whose distance is smaller than a certain r is linked together by an edge.

This kind of graph topology is a good representation for wireless network. The graph given at the Figure 1 is a randomly generated graph on which the algorithm is going to be executed.

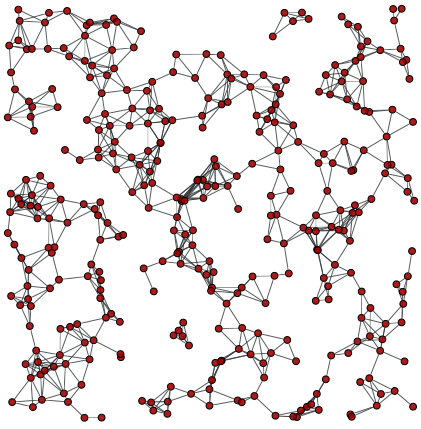

Figure 1: Random geometric graph with 350 nodes and an edge for the pairs with an euclidean distance lower than 0.75.

To test if the coloring returned by the FrogSim algorithm is good, we look for the maximum k-*clique*[6] as such we know there is at least k colors needed. A 8-clique for the graph given at the Figure 1 is represented in red in the 2.

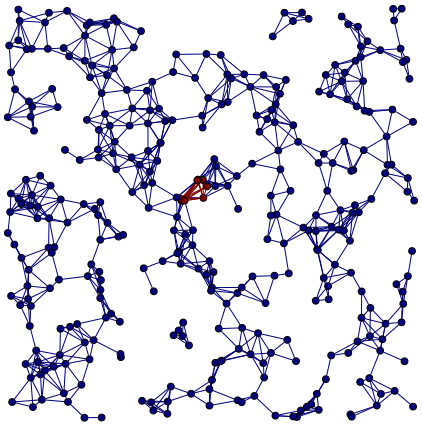

Figure 2: Lower bound for the minimum number of colors needed (8).

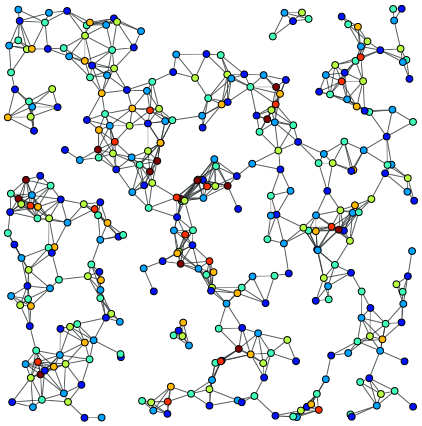

Figure 3: Instance of a valid coloring returned by the FrogSim algorithm.

Convergence Rate The algorithm specified by Hernández and Blum (2010) is designed to converge as fast as possible, depending on the parameter. We can see in the Figure 4 and the Figure 5, that the system get stable near the twentieth communication round. The need of convergence of the system can be explained in terms of getting an acceptable coloring in a minimum amount of communication rounds. Once the θ-values do not change anymore, the colors do not change either.
First the algorithm is tested with the typical parameters specified by Hernández and Blum (2010), ($\alpha = 0.5$). The resulting coloring are acceptable solutions where the number of colors is equal to the minimum possible or very close– usually k^* or $k^* + 1$.

[6] A *k-clique* being a subset of k nodes having an edge between any pair of these nodes.

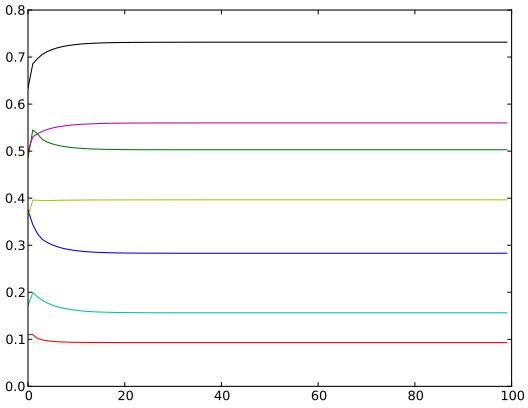

Figure 4: Convergence of the θ-values for the nodes of a clique.($\alpha_i = 0.5$)

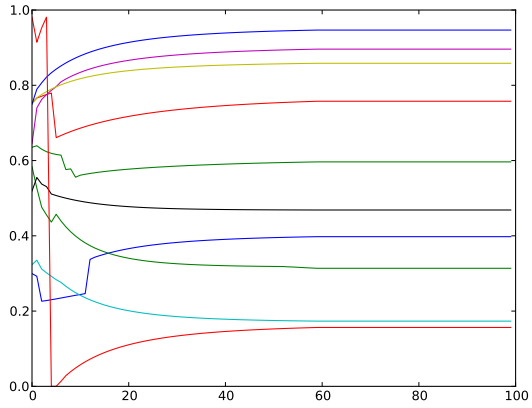

Figure 5: Convergence of the θ-values for the nodes of a clique for a specifier α-value.

A second set of parameters have been tested. The second experiment take into account some remarks of Hernández (2012) and initialize the α-values–the convergence–of the nodes according to the number of neighbors of each node. First, a convergence value is set for all the nodes of the system as an upper bound value. Then the value is changed, during the start message, according with the number of neighbors that a particular node get–which is denoted n_i.

$$\alpha_i = \alpha * 1 - \frac{1}{\ln n_i + 2} \tag{3}$$

This value is set as such in order to force the less influenced nodes to stabilize first. An example of the θ-values comportment for a clique[7] is shown in the Figure 5 and the evolution of the average θ-values in the system is draw in the Figure 6.

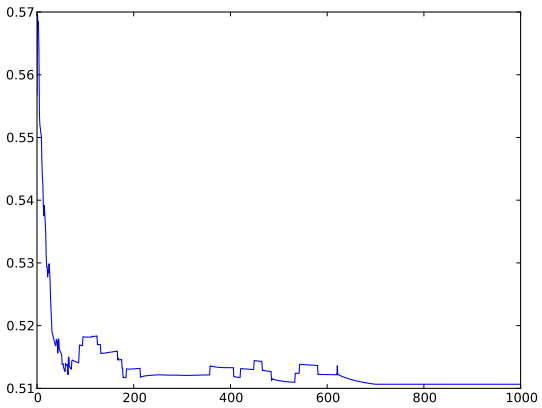

Figure 6: Average of the θ-values for a system with specified a convergence value for each node.

Grid topologies

Graphs representing grid topologies have very similar look to a chessboard where each cell is a node and edges are present between adjacent cells. These graphs are essentially matrices where a link only exists between cells directly next to each other ; as such grid topologies have the same two dimensional parameters as matrices do. A grid topology graph will be classified following it's $n \times m$ dimensions.

An example of such a graph can be found in Figure 7.

Grid topologies are very easily colored theoretically and in centralized systems. This comes from the fact that all such graphs only ever need two colors. Indeed, as seen in Figure 8, the optimal coloring is that of a chessboard : every other node is black and the nodes inbetween are white[8].

[7]This is interesting because a large clique will be composed of nodes that are highly influenced.

[8]Any other pair of colors can of course be used.

However, as Hernández and Blum (2010) discussed, distributed systems are unable to discover an optimal coloring as easily. Because each node only knows its direct neighbors, it is impossible for a node to know its exact position in the grid and as such to know what its optimal color should be. More complex algorithms are thus needed to find optimal coloring, such as *FrogSim*.

Convergence Rate The algorithm takes more times to stabilize for the grid topology because of the influence–even indirect–of each node on all the other nodes of the system. As such the optimal coloring is much harder to achieve due to the unique solution. Similarly to any other topologies the algorithm should expand stability from the less influenced nodes, which in this case are the four corners of the grid. An example of θ-values behaviour is shown at the Figure 11.

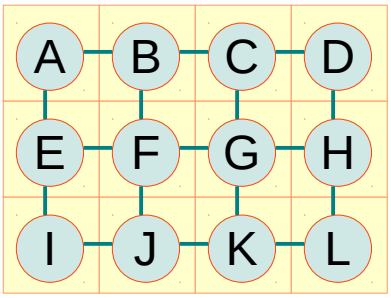

Figure 7: Example of a 4×3 grid topology graph.

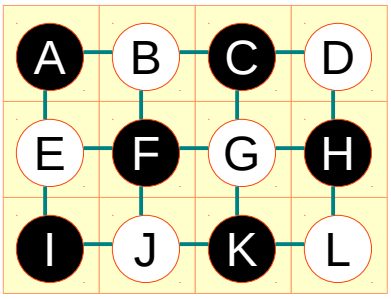

Figure 8: Optimally colorized Figure 7 graph.

As a simple remark, the optimal solution has been returned for the grid topology after we made an error by initializing the convergence rate at 75 instead of 0.75. The comportment of the phase was a permanent switch between approximately 0 and 1.

Discussion

Amelioration for Grid Topologies

The discussed algorithm being a distributed heuristic that also uses the concept of spanning tree and master node of the graph to be colored can be optimized for graph representing a grid topology. In fact, when accepting the use of a master node in our distributed algorithm, we can easily build a deterministic algorithm that will achieve better performances.

We know that the optimal coloring of a grid topology is that of a chessboard[9]. The challenge to find such an optimal coloring with a distributed algorithm is that the nodes do not know their exact position within the grid : they only know their neighbors.

Because we accept the use of a master node, we can use the following algorithm to color the graph. Each node, starting with the master node, will broadcast to its neighbors

[9] See Grid Topology on page 5.

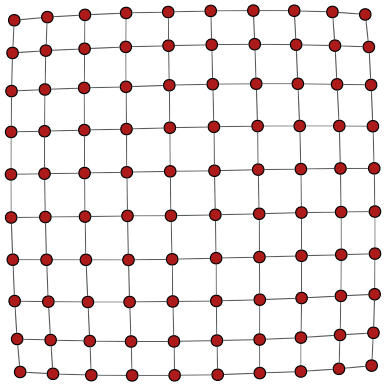

Figure 9: 10×10 grid.

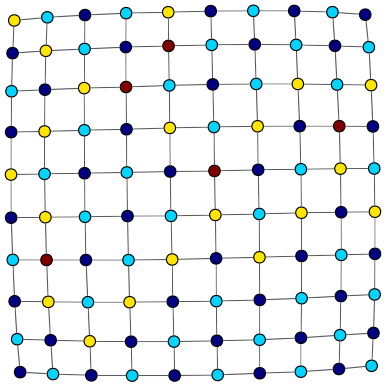

Figure 10: 10×10 grid coloring instances returned by FrogSim.

a message containing its distance[10] from the master node. The nodes with an even distance from the master node will arbitrarily be colored black, and those with odd distance white. The messages will only be sent when the node is sure of its distance ; this happens when itself has received a message from a neighbor giving the neighbor's distance, from which the node can easily calculate its own distance : the neighbor's distance is simply incremented.

The algorithm starts with the master node sending a message containing 0 to all its neighbors. all these neighbors now broadcast a 1, and so on. When a node receives a message after having broadcasted its distance, the message is ignored.
In practice, the messages are broadcasted along the spanning tree of the graph which implies that the time for the algorithm to conclude is dependent on the height of said spanning tree ; it will take the time for h successive message to be sent (and analyzed when received), where h is the

[10] The number of hops necessary to get to the master node from the current node.

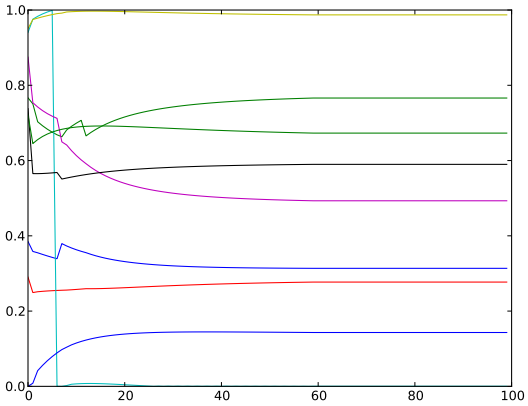

Figure 11: Evolution of the θ-values for a 3×3 sub-graph of a grid.

height of the spanning tree.

In a grid topology the spanning tree of minimal height h^* has it's root in the middle of the grid. Say the grid has dimensions $n \times m$, h^* can be found using the following formula :

$$h^* = \lceil \frac{max(m,n)}{2} \rceil$$

h^* is the total number of time steps required by this proposed algorithm to converge on the optimal coloration of the given graph representing a grid topology.

References

Aihara, I. (2009). Modeling synchronized calling behavior of japanese tree frogs. *Phys. Rev. E*, 80:011918.

Barenboim, L. and Elkin, M. (2011). Deterministic distributed vertex coloring in polylogarithmic time. *J. ACM*, 58(5):23:1–23:25.

Blöchliger, I. and Zufferey, N. (2008). A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comput. Oper. Res.*, 35(3):960–975.

De Marco, G. and Pelc, A. (2001). Fast distributed graph coloring with o(&dgr;) colors. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 630–635, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Dorne, R., kao Hao, J., Scientifique, P., and Besse, G. (1998). A new genetic local search algorithm for graph coloring. In *In Parallel Problem Solving from Nature - PPSN V, 5th International Conference, volume 1498 of LNCS*, pages 745–754. Springer-Verlag.

Finocchi, I., Panconesi, A., and Silvestri, R. (2002). Experimental analysis of simple, distributed vertex coloring algorithms. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 606–615, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Hernández, H. (2012). *Swarm Intelligence Techniques for Optimization and Management Tasks in Sensor Networks*. PhD thesis, Universitat Politècnica de Catalunya.

Hernández, H. and Blum, C. (2010). Distributed graph coloring: An approach based on the calling behavior of japanese tree frogs. *CoRR*, abs/1011.5349.

Hertz, A., Plumettaz, M., and Zufferey, N. (2008). Variable space search for graph coloring. *Discrete Applied Mathematics*, 156:2551–2560.

Kubale, M. and Kuszner, L. (2002). A better practical algorithm for distributed graph coloring. In *Parallel Computing in Electrical Engineering, 2002. PARELEC '02. Proceedings. International Conference on*, pages 72–75.

Lee, S. and Lister, R. (2008). Experiments in the dynamics of phase coupled oscillators when applied to graph colouring. In *Proceedings of the Thirty-first Australasian Conference on Computer Science - Volume 74*, ACSC '08, pages 83–89, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Lee, S. A. (2010). k-phase oscillator synchronization for graph coloring. *Mathematics in Computer Science*, 3(1):61–72.

Malaguti, E., Monaci, M., and Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS J. on Computing*, 20(2):302–316.