FrogCOL and FrogMIS: New Decentralized Algorithms for Finding Large Independent Sets in Graphs*

Christian Blum[†1,2], Borja Calvo[‡1] and Maria J. Blesa[§3]

[1] *University of the Basque Country, San Sebastian, Spain*
[2] *Ikerbasque, Basque Foundation for Science, Bilbao, Spain*
[3] *Universitat Politécnica de Catalunya, Barcelona, Spain*

September 2015

Abstract

Finding large (and generally maximal) independent sets of vertices in a given graph is a fundamental problem in distributed computing. Applications include, for example, facility location and backbone formation in wireless ad hoc networks. In this paper we study a decentralized (or distributed) algorithm inspired by the calling behaviour of male Japanese tree frogs, originally introduced for the graph coloring problem, for its potential usefulness in the context of finding large independent sets. Moreover, we adapt this algorithm to directly produce maximal independent sets without the necessity of first producing a graph coloring solution. Both algorithms are compared to a wide range of decentralized algorithms from the literature on a diverse set of benchmark instances for the maximal independent set problem. The results show that both algorithms compare very favorably to their competitors.

Keywords: Swarm intelligence; self-desynchronization; maximal independent set; decentralized algorithms

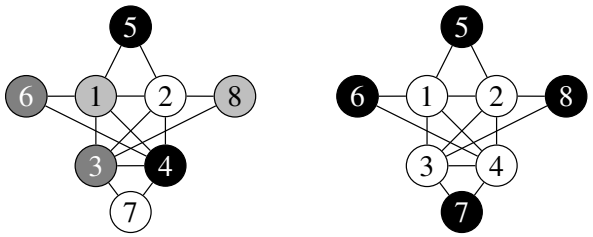
*This work was supported by projects TIN2012-37930, TIN2013-41272P and TIN2007-66523 of the Spanish Government, and project SGR 2014-1034 of the Generalitat de Catalunya. In addition, support is acknowledged from IKERBASQUE (Basque Foundation for Science) and the Basque Saiotek and Research Groups 2013-2018 (IT-609-13) programs. Our experiments have been executed in the High Performance Computing environment managed by RDlab (`http://rdlab.lsi.upc.edu`) and we would like to thank them for their support.

† christian.blum@ehu.es
‡ borja.calvo@ehu.es
§ mjblesa@cs.upc.edu

1 Introduction

The maximal independent set problem can be described as follows. Given an undirected graph $G = (V, E)$, an independent set $V_{\text{IS}} \subseteq V$ is a subset of the vertices of G such that no two vertices of V_{IS} are adjacent, that is, connected by an edge. A *maximal independent set (MIS)* $V_{\text{MIS}} \subset V$ is an independent set to which no other vertex from $V \setminus V_{\text{MIS}}$ can be added without destroying the property of being an independent set. Finally, a *maximum independent set* is a MIS of maximum size. Both problems—that is, the maximal and the maximum independent set problems—are fundamental in computer science and related fields (see, for example, [8]). It is well known that, from the perspective of centralized algorithms, the maximum independent set problem is NP-hard [13], while the maximal independent set problem is in P. In fact, various—rather unsophisticated—greedy algorithms for the generation of maximal independent sets have been presented in the related literature (see, for example, [15]). As a side comment, note that a maximum independent set in a given graph G corresponds to a maximum clique in graph G's complement.

Another well-known problem, with strong relations to the maximal independent set problem, is *graph coloring* [19]. Given an undirected graph $G = (V, E)$, the goal is to find a mapping $c : V \to \mathbb{N}$ assigning to each vertex $v \in V$ a label $c(v)$ such that no two adjacent vertices have the same label—that is, $c(u) \neq c(v), \forall\ (u, v) \in E$—and the number of different labels used is minimal. Vertex labels are also called colors. From the perspective of centralized algorithms the graph coloring problem is NP-hard [20]. By definition, given a valid graph coloring solution, subsets of vertices with the same label (or color) form independent sets.



(a) An optimal graph coloring solution.

(b) The maximum independent set (black vertices).

Figure 1: Example for the relation between graph coloring and independent sets.

Figure 1 presents an illustrative example. Subfigure 1a shows an optimally colored graph. Observe that vertices 1, 2, 3, and 4 form a fully connected subgraph. This is why four different colors are needed for coloring this subgraph. The same four colors can be used to color the rest of the vertices. Each subset of vertices that have the same color forms an independent set. Therefore, the largest independent set of vertices that can be extracted from this coloring solution is of size two. Henceforth, given a coloring solution of a graph, we will regard the largest subset of vertices with the same color as the corresponding independent set solution. This means that, given a coloring solution of a graph,

we also have a well-defined independent set solution. However, subfigure 1b shows that an optimal coloring solution does not necessarily provide a high-quality independent set solution, because the maximum independent set shown in 1b—formed by the black vertices—is of size four.

1.1 Contribution of this work

In this work we consider the problem of, given an undirected graph, finding large independent sets of vertices in a decentralized (or distributed) way. This problem will henceforth be labelled MIS. It has applications, for example, in the context of facility location and backbone formation in wireless networks [10, 21]. In a first step, we study a recently proposed (decentralized) algorithm for graph coloring, which will be labelled FrogCOL in the context of this paper [16, 17, 1]. This algorithm, which is inspired by the self-desynchronization of the calls of male Japanese tree frogs, has the inconvenience that it requires to *translate* graph coloring solutions into independent set solutions. Moreover, it makes use of a mechanism in which a master node is used to identify and store the best solution of a run and to communicate this solution to the other vertices. With the goal of developing a less sophisticated algorithm which is able to maintain a reasonably high solution quality, this paper proposes a cut-down version of FrogCOL labelled FrogMIS. This algorithm directly computes independent sets. Moreover, it does not make use of the mechanism for identifying and storing the best solution of a run. Instead, the last solution generated before convergence is taken as the output of the algorithm. Finally, this paper provides an exhaustive experimental evaluation of the two proposed algorithms, in comparison to a wide range of decentralized independent set algorithms from the literature. To our knowledge, such an experimental comparison of decentralized independent set algorithms has never been done in the related literature.

The results show, first of all, that FrogCOL significantly outperforms decentralized competitor algorithms from the literature. An exception arises in the context of rather large and dense graphs. However, the algorithms that are able to outperform FrogCOL in this setting, require an unpractically large number of communication rounds in order to obtain their results. As a conclusion we can state that FrogCOL has a strong tendency to generate coloring solutions in which the nodes associated to the most-used color correspond to large independent sets. The results also show that—as expected—FrogCOL is generally superior to FrogMIS. However, even though FrogMIS is less sophisticated than FrogCOL, FrogMIS is able to significantly outperform all the other decentralized algorithms from the literature concerning three of the four considered graph types.

Finally, note that the work presented in this paper is an extension of [4]. The extension consists in the introduction of the FrogMIS algorithm, in considering a larger benchmark set containing different types of graphs, and in a comparison to most of the decentralized competitor algorithms from the literature.

1.2 Organization of the paper

The reminder of this article is organized as follows. In Section 2, a short description of FrogCOL is provided. Then, in Section 3, the adaptation to the independent set problem, FrogMIS, is outlined. The experimental evaluation

is documented in Section 4. Finally, conclusions and an outlook to future work can be found in Section 5.

2 The FrogCOL algorithm

In the following we provide a concise description of the FrogCOL algorithm in order for the paper to be self-contained. A more detailed description can be found, for example, in [16]. In the context of the description provided in the following, we assume that the algorithm is implemented in a static wireless ad hoc network with n nodes equipped with homogeneous radio antennas. The type of antennas used implies a communication graph, that is, an undirected graph in which the vertices correspond to the network nodes, and in which two vertices are connected by an edge in case the corresponding network nodes can communicate with each other via their radio antennas. Note that this communication graph, henceforth denoted by G, is the graph to be colored by FrogCOL.

2.1 A preliminary step

Before the start of the algorithm, it is actually necessary—for several reasons, as outlined below—that the wireless ad hoc network is organized in the form of a rooted tree. The decentralized algorithm from [5], which generates a minimum-diameter tree, may be used for this purpose. This algorithm requires $O(|V|)$ communication rounds. The root (or master) node of this tree will have some additional tasks to fulfill. It initiates, for example, a short process in order to calculate the height of the tree. Moreover, the master node initiates the start of the FrogCOL algorithm by means of a broadcast message. Additionally, the height of the tree will be communicated to the remaining network nodes by this broadcast message. As explained further down, the tree is used during the execution of the FrogCOL algorithm for passing the information of the color-to-node assignments to the master node and for calculating the state of convergence which will be used to stop the algorithm.

2.2 Main algorithm

FrogCOL works in communication rounds, each having a length of one time unit. A network node i maintains, at all times, a value $\theta_i \in [0, 1)$ and the current color $c_i \in \mathbb{N}^+$.[1] The θ_i value determines, at each communication round, the precise moment at which a network node executes the short program shown in Algorithm 1. More precisely, assuming that the current communication round starts at time t, node i executes its program at time $t + \theta_i$. The execution of Algorithm 1 includes sending one message. In order to store these messages received from neighboring network nodes, each network node i maintains a message queue Q_i. A message $m \in Q_i$ has the following format:

$$m =< \text{theta}_m, \text{color}_m > \ , \tag{1}$$

[1] Note that, for simplicity and without loss of generality, natural numbers greater than zero are used to uniquely identify colors.

Algorithm 1 Program of each node $i \in V$

1: $\theta_i :=$ calculateNewThetaValue()
2: $c_i :=$ minimumColorNotUsed()
3: sendColoringMessage()
4: clearMessageQueue()

where $\text{theta}_m \in [0.1)$ contains the θ value of the sender and color_m is the color currently used by the sender. In the following we provide a technical description of the functions of Algorithm 1.

The first step of Algorithm 1 consists in the adaptation of θ_i in function calculateNewThetaValue(). More in detail, θ_i is changed on the basis of the messages received from neighboring nodes, that is, the messages from Q_i. Note that only one message from each possible sender node is considered. In the case that Q_i contains two or more messages from the same sender node, the newest one is chosen and the others are discarded. The adaptation of θ_i is done as follows:

$$\theta_i := \theta_i - \alpha \sum_{m \in Q_i} \frac{\mathbf{sin}(2\pi \cdot (\text{theta}_m - \theta_i))}{2\pi} \quad , \tag{2}$$

where $\alpha \in [0, 1]$ is a parameter used to control the convergence of the system. In general, the lower the value of α the smaller the change applied to θ_i. Parameter α will be subject to tuning.[2]

The second step of Algorithm 1 consists in the choice of a (possibly new) color for network node i. This is done in function minimumColorNotUsed(). Formally, the new color is determined as follows:

$$c_i := \mathsf{min}\{c \in \mathbb{N}^+ \mid \forall m \in Q_i \colon \text{color}_m \neq c\} \tag{3}$$

In other words, network node i chooses the color with the lowest identifier that is not already chosen by one of its neighbors, that is, the lowest index color that does not appear in any of the messages $m \in Q_i$. Before finalizing its program, a network node i must communicate its new color to its neighbors (see function sendColoringMessage()). This function is responsible for sending the following message m:

$$m =< \text{theta}_m := \theta_i, \text{color}_m := c_i > \tag{4}$$

To conclude the description of network node i's program, the message queue Q_i is cleared by removing all messages (see function clearMessageQueue()). As a final note, observe that it is ensured that the colorings produced at each communcation round are valid, because the θ-values are used to make nodes choose colors at different times. Moreover, after choosing a color, a node communicates this choice to its neighbors (see step 3 of Algorithm 1). Therefore, a node never chooses a color which is already chosen by one of its neighbors that have already chosen a color in the same communication round.

[2]Note that Eq. (2) is strongly inspired by a model of Japanese tree frogs' de-synchronization behaviour from [3]. We refer the readers interested in the biological background of the FROG-COL algorithm to [16].

2.3 Identifying the best coloring and detecting convergence

The way in which the algorithm identifies a new best coloring and detects convergence is based on the use of the minimum-diameter tree which was generated in a preliminary step (see Section 2.1). In the following we provide a short description of this mechanism. A complete technical description, which is out of the scope of this paper, can be found in [16, 17].

Henceforth, let h refer to the height of the minimum-diameter tree. Note that h corresponds to the maximum number of communication rounds necessary for the master node to pass information to the rest of the nodes, and vice versa. In the following we assume that the master node knows about the number of nodes in the network. At each communication round, each network node i is required to communicate the following information to its parent node in the minimum-diameter tree: (1) a real number corresponding to the sum of the distances between the old theta values and the new ones concerning all nodes included in the subtree of which it acts as root, (2) the index of the largest color used by itself and all nodes included in the subtree of which it acts as root, and (3) an integer indicating the corresponding communication round number. Note that (3) is necessary because the information from the different nodes about their color choice at a specific communication round will not arrive all at the same time to the master node. Note also that no extra messages are required for passing this information. This is because the corresponding information can be added to the coloring messages of Algorithm 1. Even though these messages will be received by all neighboring nodes, only the parent nodes in the minimum-diameter tree will care about this information.

Note that, in the first communication round, the only nodes that report the above-described information to their parents are the leaf-nodes of the minimum-diameter tree. This is because they are the only ones without children, that is, the subtree of which they act as root only contains themselves. In the second communication round, the parents of the leaves will be able to report the aggregated data to their respective parents. Given the height h of the tree, h communication rounds are needed for all the information regarding a specific communication round to reach the master node. This means that the network nodes must store the differences between their old and new theta values, and the information about color use, during h communication rounds. Once the master node has received all the necessary information concerning a specific communication round, it is able to derive the following information. First, it knows the maximum index of any color used at the corresponding communication round. This information can be used to determine if a new best coloring has been found. Second, by dividing the sum of all theta-differences by the size of the network it obtains the average change of the theta values in the corresponding communication round. In case this average change is below a certain threshold value (we used 0.0001 in all experiments), the master node broadcasts a stopping message to all nodes, which terminates the algorithm.

2.4 Initial setting of the θ values

In the original version of the algorithm as described in [16] the initial θ values are chosen at random. In this work, we also tested a second option, which depends

on the number of neighbors with which a network node is able to communicate. This measure is known as the degree of a vertex in graph theoretical terms. In particular, we aim for the θ values to be proportional to the degree of the corresponding network node. The motivation for this choice is as follows. One of the best centralized greedy heuristics lets vertices decide if (or not) to be a member of the maximal independent set depending on their (current) degree. Vertices with lower (current) degrees may choose first. In the setting of a wireless ad hoc network it is easy for each node to determine its degree. This can be done by each node sending a hello message and by counting the number of hello messages received. Let $d_i \geq 0$ denote the degree of network node i. We used the following function for determining the initial θ_i of node i:

$$\theta_i := 1 - 1.1^{-d_i+\epsilon} \ , \tag{5}$$

where ϵ is a uniform random number from $(0, 0.01]$. The small perturbation caused by ϵ is to try to avoid that two (or more) network nodes have the same initial value.

3 The FrogMIS algorithm

The motivation for the development of the FrogMIS algorithm—described below—is as follows. Real-world applications which require the computation of (large) independent sets in graphs may have very different needs. Some algorithms from the literature are developed for applications which ask for a solution of reasonable quality in very few communication rounds. Other algorithms such as FrogCOL are more sopisticated and spend more communication rounds for producing their solutions. However, the solutions produced by these algorithms are usually of higher quality than the ones produced by unsophisticated algorithms. The introduction of FrogMIS is an intent to develop a less sophisticated algorithm than FrogCOL without loosing too much solution quality.

In FrogMIS, all aspects concerning the θ values remain unchanged, that is, FrogCOL and FrogMIS are exactly the same for what concerns the use and the adaptation of the θ values. However, instead of a current color $c_i \in \mathbb{N}^+$, each network node i maintains a variable mis_status_i, which may take values zero or one. If $\text{mis\_status}_i = 0$, network node i has chosen not to be part of the independent set solution. Otherwise, if $\text{mis\_status}_i = 1$, network node i forms part of the independent set solution. Due to this change, the structure of the messages sent by the network nodes in the course of the algorithm changes. A message $m \in Q_i$ has now the following format:

$$m =< \text{theta}_m, \text{mis\_status}_m > \ , \tag{6}$$

where $\text{theta}_m \in [0.1)$ contains the θ value of the sender and mis_status_m contains the value of the mis_status variable of the sender.

The last change concerns function minimumColorNotUsed(). Instead of choosing a new color, in FrogMIS this function examines all messages in Q_i. In case there is at least one message $m \in Q_i$ with $\text{mis\_status}_m = 1$, the value of mis_status_i is set to zero. In other words, if there is just one neighbor that already forms part of the independent set solution, network node i cannot be part of the indepenent set solution itself. Otherwise, the value of mis_status_i is set to one.

Another important difference between FrogCOL and FrogMIS is the following one. While the minimum-diameter tree is used in FrogCOL for identifying and storing the best coloring solution (respectively, independent set solution) found during the course of the algorithm, the solution provided by FrogMIS is simply the solution of the last communication round at the time of convergence.

Concerning the relation between FrogCOL and FrogMIS, note that FrogMIS is—basically—a cut-down version of FrogCOL. The modification of the θ-values is done, in both algorithms, in the same way. However, instead of choosing a new color at each communication round (remember that in FrogCOL a node chooses the smallest color index that has not yet been chosen by any of its neighbors) a node in FrogMIS simply chooses if to be part of the independent set, or not. This corresponds to choosing color 0, if possible, and not choosing any color, in case color 0 cannot be chosen. However, note that color class 0 is not necessarily the largest one, that is, the set of nodes that have color 0 is not necessarily larger than the set of nodes having a different color x. Therefore, it is clear that FrogMIS cannot perform better than FrogCOL. Even more, in many cases the performance of FrogMIS can be expected to be worse than the one of FrogCOL. On the other side, FrogMIS has the advantage that, at the end of a communication round, each node already knows if it belongs to the current independent set, or not. In the case of FrogCOL, the master node determines (with some delay) for each communication round the largest color class, and informs the remaining nodes. Finally, another difference between FrogCOL and FrogMIS concerns the fact that FrogCOL always returns the best independent set encountered, while FrogMIS returns the one from the last communication round, which is not necessarily the best one encountered throughout a run.

4 Experimental evaluation

FrogCOL and FrogMIS were implemented in C++ without the use of any external libraries. Experiments were performed by means of discrete event simulation. In the following we first describe the set of benchmark instances that we generated to test both algorithms. Then, we describe the tuning experiments that were performed in order to determine the setting of parameter α (which determines the convergence speed) and the initial setting of the θ-values in the context of the FrogCOL and FrogMIS algorithms. Finally, after a short description of the chosen competitor algorithms, an exhaustive experimental evaluation is presented.

4.1 Benchmark instances

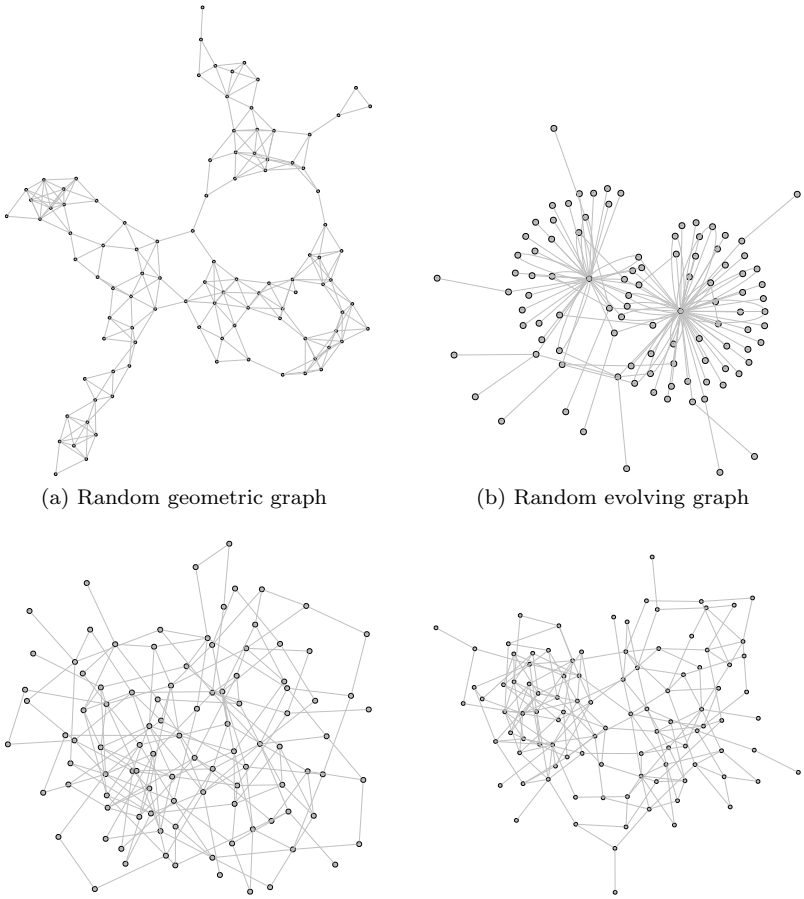
With the aim of testing our algorithms in different scenarios we have created four different sets of graphs: random geometric graphs (RGG), random graphs (RG1), clusterized random graphs (RG2) and random evolving graphs (REG). Figure 2 shows examples of these graphs. In each of the four cases, graphs of different properties—for what concerns, for example, the density—and different sizes have been created. In particular, for each type we generated graphs of

100, 1000 and 5000 nodes, that is, $n \in \{100, 1000, 5000\}$. Finally, note that 30 random graphs were created for each combination between graph characteristics and size.

The random geometric graphs were created as follows. First, the nodes are assigned to random coordinates from the unit square. Then, a 'radius' (r) is fixed and each pair of nodes at a distance smaller or equal than the radius is connected by an edge. The radius controls the density of the graph, that is, the larger the radius the denser the resulting graph is. In order to find a reasonable range for the r-values for each graph size n, the following experiments were performed. For each combination of $r \in [0.01, 0.3]$ (by steps of 0.01) and n we generated 100 random geometric graphs and recorded the number of times (out of 100) these graphs were connected. Based on these results we determined the smallest r-values to be considered for the three graph sizes to be $r = 0.14$ (in case $n = 100$), $r = 0.049$ (in case $n = 1000$) and $r = 0.024$ (in case $n = 5000$). With these values of r, the generated random geometric graphs have a probability of approx. 5% to be connected. Moreover, the resulting graphs are rather sparse. In order to find suitable upper ranges for the r-values, we examined the *(relative) average degrees* of the generated graphs. Hereby, the term (relative) average degree refers to the average degree of a node expressed in terms of the fraction of all nodes to which the respective degree corresponds. For example, assume that $n = 100$ and that a node is, on average, connected to 10 neighbors. In this case the relative average degree is 0.1. In particular, we decided that the densest graphs to be considered in this study should have a (relative) average degree of 0.05. Our experiments have shown that such graphs can be generated with $r = 0.169$ (in case $n = 100$) and $r = 0.134$ (in case $n \in \{1000, 5000\}$). Finally, we decided to generate 30 random geometric graphs for 10 different r-values between the previously determined lower and upper ranges. Figure 2 (a) shows an example of random geometric graph with 100 nodes.[3]

The second and third types of graphs that we generated are random graphs. These graphs are created adding edges between nodes totally at random, with a given probability. This probability controls the density of the graph. The main difference between random geometric graphs and random graphs is that in the former ones only nodes that are placed close together may be connected while in the latter ones any two nodes may be connected. Two different types of random graphs were generated: (1) standard random graphs where any two nodes have the same probability of being connected and (2) clusterized random graphs, where nodes are divided into two groups and the probabilities of being connected in each cluster and between clusters are different. Regarding the standard random graphs, we have selected the probability of connecting two nodes (p) from $\{0.03, 0.04, 0.05\}$ in graphs of size 100 and from $\{0.01, 0.03, 0.05\}$ for the remaining graph sizes. The case of two-cluster graphs is more complex. We have used two different probabilities, henceforth labelled low and high, for the generation of these graphs. In the case of $n = 100$, low and high correspond to 0.06 and 0.1, and for the remaining graph sizes, low and high correspond to 0.02 and 0.1. Then, graphs with the following three configurations were generated: (1) both clusters have a low inner connection probability, (2) both clusters have a high inner connection probability, and (3) one cluster had a low

[3]Note that although in these graphs the nodes have spatial coordinates, these have not been used to plot the graph. Indeed, all the graphs in the figure have been created using the 'neato' layout of the `dot` software



(a) Random geometric graph

(b) Random evolving graph

(c) Random graph

(d) Clusterized random graph

Figure 2: Examples of the types of generated graphs. Displayed graphs consists of 100 nodes.

and the other one a high inner connection probability. In all cases the probability of connecting nodes from different clusters was set to 0.005. Figures 2(c) and (d) show examples of random graphs and clusterized random graphs, respectively.

The last type of graph we have created are random evolving graphs. These graphs are constructed adding nodes one by one. Every time a node is added, it can be connected to any of the existing nodes with a probability that depends on the degree of that node and a parameter, the *preferential attachment exponent* (*pa*). The values used for the *pa* parameter are $\{0.5, 1.5\}$. The maximum number of such connections is limited by a second parameter ($arc_{\max}$) for which the values used are from $\{3, 6\}$. Therefore, a total of four combinations of parameters have been used for each graph size. Depending on the parameters, this procedure creates graphs made of star-like structures, as can be seen in Figure 2(b). In general, the higher the *pa* parameter the bigger the star structures.

In summary, our benchmark is made of four types of graphs: 900 random geometric graphs, 270 random graphs with one cluster, 270 random graphs with two clusters, and 360 random evolving graphs. All the graphs, with the exception of the random geometric graphs, have been created using `R` and the *igraph* package [6]. In particular, the `preference.game` function was used to create random graphs while random evolving graphs were generated using the `aging.prefatt.game` function. Finally, note that during the generation of the graphs, non-connected graphs were discarded. In other words, all the graphs in the benchmark set are connected graphs.

4.2 Algorithm tuning

Both FrogCOL and FrogMIS have two parameters whose values need to be determined. The first one is parameter α from Eq. (2) affecting the update of the θ values. This parameter basically determines the magnitude of the θ value update, and has, therefore, an influence on the convergence speed. The range of possible values for α that we selected is $\{0.01, 0.1, 1.0\}$. The second parameter concerns the initial setting of the θ values. The first option, henceforth labelled *random*, assigns values chosen uniformly at random from $[0, 1)$ to all the θ's. The second option, henceforth labelled *heuristic*, was described in Section 2.4.

We decided to make use of the automatic configuration tool irace [22] for the tuning of the two parameters. More specifically, irace was applied to tune both FrogCOL and FrogMIS separately for each of the four types of graphs and for each instance size from $\{100, 1000, 5000\}$. Concerning the random geometric graphs, 10 graphs per instance size—one graph for each considered value of the radius r—were randomly generated for tuning purposes. Similarly, 12 graphs for tuning were randomly generated per instance size—four graphs for each of the three considered (combinations of) edge probabilities—in the context of the random (evolving) graphs. Finally, each run of FrogCOL and FrogMIS was given a budget of maximally 1000 communication rounds, and irace was given a budget of 120 algorithm runs for each tuning process.

The applications of irace produced the configurations of FrogCOL and FrogMIS as shown in Table 1. The following trends can be observed. First of all, in both cases the best value of α decreases with growing graph size.

Table 1: Parameter settings produced by irace for the 12 combinations of graph type and instance size. See the first paragraph of Section 4.1 for the definition of the graph type abbreviations.

(a) Tuning results for FrogCOL

graph type	**instance size**	α	**setting of θ's**
RGG	100	1.0	*random*
	1000	0.01	*heuristic*
	5000	0.01	*random*
REG	100	1.0	*heuristic*
	1000	0.1	*heuristic*
	5000	0.1	*heuristic*
RG1	100	1.0	*heuristic*
	1000	0.01	*heuristic*
	5000	0.01	*random*
RG2	100	1.0	*heuristic*
	1000	0.1	*heuristic*
	5000	0.01	*heuristic*

(b) Tuning results for FrogMIS

graph type	**instance size**	α	**setting of θ's**
RGG	100	0.1	*heuristic*
	1000	0.1	*random*
	5000	0.01	*random*
REG	100	0.1	*random*
	1000	0.01	*random*
	5000	0.1	*heuristic*
RG1	100	0.1	*random*
	1000	0.1	*random*
	5000	0.01	*random*
RG2	100	0.1	*random*
	1000	0.1	*random*
	5000	0.01	*heuristic*

Second, the heuristic setting of the θ-values seems to be the prefered choice in the context of FrogCOL, while the random setting seems to be the preferable choice for FrogMIS. Finally, in the context of all four types of graphs, the obtained settings concerning the largest graphs (size of 5000) is identical for both FrogCOL and FrogMIS.

4.3 Competitor algorithms

A very recent general survey on decentralized algorithms for finding maximal independent sets is provided in [9]. However, up to date no experimental comparison between these algorithms has been published in the literature. Therefore, we decided to implement most of the algorithms mentioned in [9]. On one side, there are algorithms belonging to the class of *self-stabilizing* algorithms, where

self-stabilization refers to an optimistic fault tolerance approach for distributed systems that was initially introduced by Dijkstra in [7]. Three self-stabilizing algorithms for the maximal independent set problem were proposed in the literature (see also [14]). The first one was proposed by Shukla et al. in [24], the second one by Ikeda et al. in [18], and the third one by Turau in [25]. Henceforth we will refer to these algorithms as SHUKLA, IKEDA and TURAU. Apart from these three algorithms we also implemented the two randomized algorithms mentioned in [9], which are labelled RAND1 and RAND2 (in the same way as described in [9]). Finally, we also implemented an optimized version (from [23]) of a very recent algorithm published in the *Science* journal [2]. As this algorithm is inspired by the development of the nervous system of fruitflies, this algorithm is henceforth labelled FRUITFLY. Note that all these algorithms are free of parameters. Therefore, no parameter tuning is necessary.

Apart from the decentralized algorithms mentioned above, we implemented the best-known (centralized) greedy algorithm for the MIS problem. This algorithm works as follows: at each iteration, first, it identifies the node with minimal degree and adds it to the maximal independent set under construction. Afterwards, this node—together with all its neighboring nodes—is removed from the input graph. This procedure stops once the input graph is empty. Henceforth, this algorithm is referred to by GREEDY. Finally, the standard integer linear program (ILP) for the maximum independent set problem was implemented and solved by CPLEX (with a computation time limit of 3600 CPU seconds per graph). Both CPLEX and GREEDY were applied in order to study the differences in the quality of the solutions obtained by centralized and decentralized approaches.

Concerning the stopping criteria used for the considered algorithms, note that the application of CPLEX to the maximum independent set problem must be given a computation time limit. For this purpose we decided for a limit of 3600 CPU seconds. The output provided by CPLEX is the best, if not optimal, solution found within the allowed computation time. GREEDY simply finishes once a solution is generated. Both FROGCOL and FROGMIS, and all the remaining algorithms included in the comparison, are decentralized algorithms. In the group of decentralized algorithms, FROGCOL, FROGMIS, FRUITFLY, SHUKLA, RAND1 and RAND2 are randomized approaches, while the remaining ones are deterministic approaches. Instead of computation time—as in the case of centralized algorithms—the main resource requirement of decentralized algorithms is measured by the number of required communication rounds. For FROGCOL and FROGMIS it was decided, quite arbitrarily, that more than 1000 communication rounds would be unpractical. Therefore, this limit was used as stopping criterion for both algorithms. However, in the case of the other decentralized approaches it was decided to provide them with as many communication rounds as necessary in order for them to reach their natural stopping criterion. In all cases, apart from FROGCOL and FROGMIS, the natural stopping criterion is that all nodes have decided if to form part of the independent set, and no conflicts remain. However, for practical reasons we decided to stop the algorithms after 100000 communication rounds, in case the natural stopping criterion was still not reached.

4.4 Statistical assessment of the results

As described in more detail below, the results are generally presented in each row of the result tables in terms of averages over 30 graphs that were randomly generated with the same parameter settings. The results obtained by the considered decentralized algorithms concerning each table row were statistically tested in order to determine the significance of the differences among them. This was done by comparing the results of all decentralized algorithms with the results of the decentralized algorithm obtaining the best performance. As an example, for random geometric graphs of size 5000 and a radius of 0.024 (see Table 2) the best performing algorithm is FrogCOL, as marked by bold font. The results obtained by each decentralized algorithm for the 30 graphs generated using these parameters ($n = 5000$, $r = 0.024$) are compared with the results obtained by FrogCOL to determine the significance of the differences. In fact, the results of FrogMIS are statistically equivalent, as indicated by the ★ symbol (significance level of 0.05). The differences have been assessed using Wilcoxon's signed rank test and the p-values have been corrected for multiple comparison using Finner's procedure [11].

Additionally, we aimed for detecting the differences between the algorithms (if any) for each of the four considered types of graphs. First, all the algorithms have been compared simultaneously using Friedman's test. Then, given that in all the cases the test rejects the hypothesis that all the algorithms perform equally, all the pairwise comparisons have been tested using the Nemenyi post-hoc test [12]. Finally, note that all the tests, tables and plots have been generated using `R`'s **scmamp** package, available at `https://github.com/b0rxa/scmamp`.

4.5 Comparison

The numerical results are shown in Table 2 (random geometric graphs), in Table 3 (random evolving graphs), and in Table 4 (standard and clusterized random graphs). The first two columns of each table indicate the parameters of the corresponding graphs, that is, the graph size (n) and parameters that depend on the graph type. In the case of random geometric graphs the second table column provides the radius r, in case of random evolving graphs it provides information on parameters pa and $arc_{\max}$, and in the case of standard (respectively, clusterized) random graphs the second table column provides the probability of an edge to exist (respectively, the corresponding probabilities for the edges to exist in the two clusters). Each table row provides average results for 30 graphs. Note that each algorithm was applied exactly once to each graph. For most of the algorithms the results are presented in two table columns, where the first one (with heading **avg.**) provides the average result obtained for the corresponding 30 graphs, and the second one (with heading **rounds**) indicates the average number of communication rounds needed. Exceptions are Greedy and Cplex, which are centralized algorithms for which we only provide the average results obtained. Other exceptions are Rand1 and Rand2 which, for simplicity reasons, were implemented in a sequential way (without all the message passing necessary in distributed systems). Therefore, in the case of Rand1 and Rand2 only the average results obtained are given. The last exception concerns FrogCOL, where an additional third column (with heading **rbf**) is provided which

indicates the average communication round number at which the best solution was found. Finally, note that the best results of each table row (obtained by any of the decentralized algorithms) are marked with bold font. Moreover, those decentralized algorithms whose performance is not significantly worse than the one of the best performing decentralized algorithm at a significance level of 0.05 are marked in the table with the ★ symbol. Additionally, the result obtained by CPLEX is marked with an asterisk in case the optimality of the obtained result could be proven for all 30 graphs.

Additional statistical information is provided in terms of a comparison between the algorithms for the four different types of graphs in Figure 3, which presents the result of the Nemenyi test in a graphical way. Briefly, each algorithm is positioned in the segment according to its average ranking concerning the respective graph type. Then, the critical difference (CD) is computed for a significance level of 0.05 and the performance of those algorithms that have a difference lower than CD are regarded as equal. This is indicated in the graphic by horizontal lines joining the respective algorithms.
The results allow to make the following observations:

- FROGCOL consistently outperforms all the other decentralized algorithms concerning all considered graph types and densities. The only exception occurs in the context of random geometric graphs when both graph size and graph density become rather large. In these cases, FRUITFLY is able to obtain better results than FROGCOL. However, this is at the cost of a rapidly growing number of communication rounds. In fact, we have stopped all algorithm runs once a maximum number of 100000 (hundred thousand) communication rounds was reached, because we considered such high communication round requirements as rather unpractical. In order to show when (with growing graph density) the communication round requirements of an algorithm become rather exaggerated, we underlined all cases in which the number of needed communication rounds is greater than 1000. As can be seen, this only happens in the case of FRUITFLY and SHUKLA.

- The more communication rounds an algorithm invests the better are, in general, its results. In particular, the four algorithms that spend, on average, more communication rounds than the others (FROGCOL, FROGMIS, FRUITFLY, and SHUKLA) clearly exhibit a higher performance in terms of the obtained solution quality.

- Concerning the comparison between FROGCOL and FROGMIS, we can observe that—as expected—FROGCOL consistently outperforms FROGMIS. However, remember that FROGMIS has the advantage of not having to make use of a master node to identify the best solution found during a run. Moreover, in FROGMIS—at convergence—each node already knows if it belongs to the independent set. In other words, there is no need for a master node to communicate the structure of the final solution to the other nodes. Even though we still used the master node mechanism in FROGMIS for determining the convergence of the algorithm, it would be possible to implement FROGMIS without this mechanism by providing a fixed communication round limit, rather than stopping the algorithm at convergence.

Table 2: Results for random geometric graphs. The table structure is described in the first paragraph of Section 4.5.

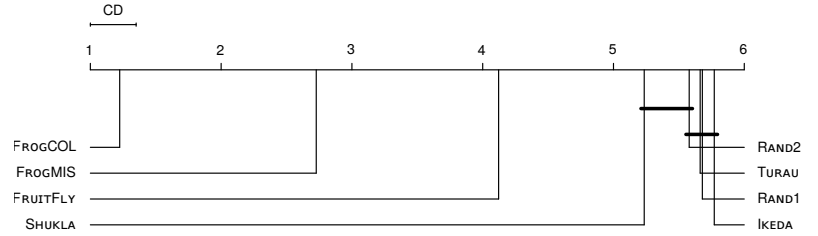
n	r	Cplex	Greedy	FruitFly		Shukla		Ikeda		Turau		Rand1	Rand2	FrogCOL			FrogMIS	
				avg.	**rounds**	**avg.**	**rounds**	**avg.**	**rounds**	**avg.**	**rounds**			**avg.**	**rbf**	**rounds**	**avg.**	**rounds**
100	0.140	*31.30	30.70	27.80	28.47	25.63	42.67	25.70	4.50	25.90	5.40	26.17	26.10	**31.07**	204.13	1000.00	29.10	463.27
100	0.143	*30.87	30.03	26.80	28.07	25.43	43.70	25.60	4.43	25.53	5.43	25.17	25.73	**30.53**	129.93	1000.00	28.57	436.70
100	0.146	*30.33	29.60	26.57	28.33	24.90	42.17	25.37	4.50	25.80	5.60	25.07	25.30	**29.93**	242.77	1000.00	28.03	456.27
100	0.149	*29.50	28.90	25.80	28.67	24.47	43.90	23.97	4.60	24.40	5.60	23.87	24.40	**29.10**	276.83	1000.00	27.33	446.07
100	0.152	*28.83	27.83	25.03	31.73	23.77	43.13	23.57	4.87	23.57	5.30	23.83	23.87	**28.47**	331.07	1000.00	25.97	475.20
100	0.155	*27.60	26.77	24.13	33.27	23.27	43.80	22.87	4.70	23.00	5.27	22.63	22.90	**27.30**	240.47	1000.00	25.47	513.27
100	0.158	*27.63	26.87	24.03	34.53	22.47	45.57	22.63	4.77	22.50	5.43	22.40	22.90	**27.33**	260.27	1000.00	24.57	447.60
100	0.161	*26.20	25.40	22.80	37.13	21.70	45.40	21.03	4.63	21.93	5.77	21.57	21.40	**25.83**	274.00	1000.00	23.40	450.60
100	0.164	*25.97	25.03	22.40	41.13	20.80	43.50	20.67	4.60	20.90	5.43	20.93	21.43	**25.57**	231.63	1000.00	22.93	384.50
100	0.169	*24.97	24.10	21.73	43.00	20.57	45.03	20.10	4.60	20.03	5.40	20.13	20.17	**24.50**	185.53	1000.00	22.20	432.13
1000	0.049	*263.77	255.07	226.27	69.60	212.40	436.80	213.20	6.17	211.90	6.63	212.70	214.53	**247.67**	32.00	720.83	227.60	731.07
1000	0.058	*208.90	198.23	174.90	117.60	162.87	458.83	162.00	6.50	162.70	7.53	161.47	163.47	**189.90**	3.40	733.63	176.93	767.17
1000	0.067	*171.43	161.10	142.23	208.60	130.83	480.30	130.37	6.70	129.87	7.67	129.83	129.60	**151.93**	2.97	723.73	140.57	759.60
1000	0.076	*141.13	131.53	117.80	478.87	105.30	481.00	104.60	6.80	105.30	7.77	105.30	104.90	**122.87**	1.53	749.13	114.13	802.17
1000	0.085	*118.90	110.93	99.43	*1138.33*	87.43	485.13	85.87	6.77	86.70	7.80	86.73	87.07	**102.37**	1.30	751.30	94.33	749.53
1000	0.094	*100.97	93.90	**85.80**	*3186.13*	74.20	493.37	72.87	6.80	72.90	8.03	72.63	72.30	85.47	1.10	761.50	79.33	775.27
1000	0.103	*87.57	80.63	**75.57**	*7939.13*	63.17	492.83	62.77	7.07	62.13	7.90	63.23	62.30	74.20★	1.00	762.60	68.13	694.10
1000	0.112	*76.60	70.63	**66.90**	*19749.60*	54.50	497.67	54.07	7.23	54.83	7.53	54.17	54.37	64.37	1.00	738.93	58.20	684.30
1000	0.121	*67.57	62.57	**59.53**	*60417.13*	47.97	498.93	47.43	6.83	47.17	7.73	47.03	47.60	56.50	21.97	714.60	51.30	678.27
1000	0.134	*57.20	52.53	24.27	*100000.00*	40.77	503.87	40.07	7.10	40.10	7.83	39.93	40.10	**47.53**	90.33	727.03	43.10	667.57
5000	0.024	*1142.77	1089.03	964.53	128.87	898.23	*2277.43*	899.60	7.27	902.50	8.53	897.83	905.30	**975.90**	540.13	755.33	972.40★	763.20
5000	0.036	620.07	577.07	**515.83**	*1270.87*	459.30	*2447.70*	453.87	7.90	452.97	8.87	455.57	456.60	502.00	374.07	764.97	495.30	767.47
5000	0.048	373.10	357.13	**332.13**	*36682.60*	276.40	*2489.13*	273.67	8.13	273.00	9.20	275.23	273.83	309.17	413.13	731.97	294.10	754.73
5000	0.060	252.13	243.20	42.03	*100000.00*	185.60	*2496.93*	182.30	7.77	183.43	9.17	184.73	183.00	**210.90**	367.13	760.20	196.93	744.87
5000	0.072	184.07	177.30	2.80	*100000.00*	134.97	*2504.30*	131.20	7.97	131.47	8.83	132.50	131.90	**153.90**	362.73	779.60	139.87	716.77
5000	0.084	140.93	134.93	0.27	*100000.00*	102.03	*2503.50*	99.87	8.13	99.60	9.17	99.37	99.83	**117.90**	280.43	774.10	105.73	689.53
5000	0.096	111.60	106.17	0.00	*100000.00*	79.97	*2504.63*	78.77	8.10	78.60	8.70	78.73	78.20	**93.30**	230.13	678.50	82.57	706.80
5000	0.108	90.23	86.63	0.00	*100000.00*	64.50	*2504.37*	63.73	7.77	63.40	8.47	63.33	63.10	**75.83**	204.77	598.07	67.43	690.80
5000	0.120	75.13	71.83	0.00	*100000.00*	54.07	*2504.80*	52.10	7.57	51.90	8.77	52.63	52.60	**63.27**	223.43	668.20	54.57	710.67
5000	0.134	61.60	58.57	0.00	*100000.00*	44.27	*2505.07*	43.43	7.70	43.47	8.67	43.10	43.07	**52.20**	162.03	568.17	45.47	611.93

Table 3: Results for random evolving graphs. The table structure is described in the first paragraph of Section 4.5.

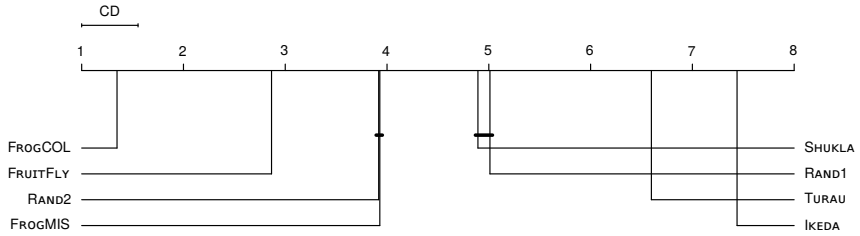
n	$(pref, arc_{max})$	Cplex	Greedy	FruitFly		Shukla		Ikeda		Turau		Rand1	Rand2	FrogCOL			FrogMIS	
				avg.	**rounds**	**avg.**	**rounds**	**avg.**	**rounds**	**avg.**	**rounds**			**avg.**	**rbf**	**rounds**	**avg.**	**rounds**
100	(0.5,3)	*61.17	52.07	58.97	19.53	54.30	49.60	49.50	4.77	50.27	5.40	54.90	56.67	**61.17**	114.30	1000.00	55.03	277.33
100	(0.5,6)	*53.40	46.47	50.10	20.27	46.50	49.73	42.53	5.00	43.03	5.50	46.73	47.73	**53.33**	108.57	1000.00	48.87	263.50
100	(1.5,3)	*86.03	85.03	85.23★	17.27	83.27★	65.93	46.57	3.17	49.47	4.33	84.47★	84.80★	**86.03**	3.23	970.00	84.83★	1000.00
100	(1.5,6)	*89.60	89.20	89.03★	17.13	88.80★	75.90	51.97	2.97	73.47★	4.03	88.63★	88.87★	**89.60**	1.13	991.93	88.77★	1000.00
1000	(0.5,3)	*609.73	495.70	580.57	25.13	546.17	486.83	488.27	7.50	506.00	6.90	540.13	561.03	**603.20**	2.90	446.60	547.70	722.73
1000	(0.5,6)	*541.50	449.27	508.73	26.67	474.17	533.40	422.40	7.83	438.43	8.03	473.20	491.30	**526.53**	1.53	618.30	482.73	726.33
1000	(1.5,3)	*952.63	950.40	950.73	22.60	948.70	664.23	601.97	3.70	793.73	4.30	949.07	949.13	**952.60**	3.57	1000.00	948.83	919.80
1000	(1.5,6)	*962.30	961.77	961.20★	25.00	959.73★	698.87	303.80	3.10	641.57	4.10	960.23★	960.67★	**962.30**	3.00	1000.00	960.77★	933.40
5000	(0.5,3)	*3053.77	2484.17	2908.13	30.40	2724.50	*2404.37*	2456.30	9.47	2532.37	7.90	2713.47	2814.30	**3019.37**	3.00	647.13	2845.03	693.40
5000	(0.5,6)	*2692.77	2193.40	2530.87	31.80	2357.00	*2664.60*	2097.07	9.83	2165.50	8.83	2332.93	2417.57	**2608.10**	1.00	992.60	2437.27	989.97
5000	(1.5,3)	*4881.40	4876.40	4878.17	28.87	4876.40	*2686.60*	4310.33	3.83	4666.53	4.80	4875.23	4877.87	**4881.30**	36.07	717.60	4877.90	694.73
5000	(1.5,6)	*4915.70	4914.87	4913.60	28.93	4911.80	*2462.00*	4899.67	3.50	4908.93	4.63	4911.77	4912.73	**4915.63**	16.83	393.97	4911.77	469.27

Table 4: Results for standard and clusterized random graphs. The table structure is described in the first paragraph of Section 4.5.

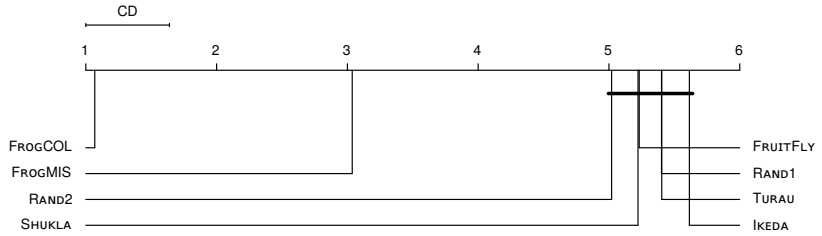
n	p	Cplex	Greedy	FruitFly		Shukla		Ikeda		Turau		Rand1	Rand2	FrogCOL			FrogMIS	
				avg.	**rounds**	**avg.**	**rounds**	**avg.**	**rounds**	**avg.**	**rounds**			**avg.**	**rbf**	**rounds**	**avg.**	**rounds**
100	0.030	*50.23	46.10	45.07	19.73	43.33	46.63	42.83	4.87	42.97	5.23	42.90	43.80	**49.87**	273.67	1000.00	45.20	223.30
100	0.040	*45.67	42.13	40.83	21.20	37.97	49.90	37.67	4.97	38.63	5.67	38.57	39.43	**45.37**	276.87	1000.00	40.53	285.10
100	0.050	*43.20	38.63	38.30	22.00	35.33	48.03	36.00	5.20	36.50	6.00	34.70	36.77	**42.80**	333.40	1000.00	37.57	309.77
1000	0.010	309.97	260.50	267.13	40.33	242.10	548.97	239.07	8.03	241.27	8.47	240.70	243.83	**275.37**	1.00	530.13	263.27	561.17
1000	0.030	151.67	123.83	130.30★	*2594.67*	114.43	546.17	113.00	9.50	112.77	10.53	113.77	114.13	**132.17**	230.97	694.77	125.20	624.80
1000	0.050	96.03	83.97	0.30	*100000.00*	77.97	541.70	77.43	10.17	77.37	11.40	78.33	78.60	**93.80**	443.90	1000.00	86.57	1000.00
5000	0.010	405.43	423.20	1.07	*100000.00*	396.50	*2698.43*	391.53	11.90	393.17	12.80	394.60	393.77	**441.17**	124.27	972.73	418.10	975.07
5000	0.030	n/a	175.77	0.00	*100000.00*	166.47	*2602.13*	166.03	13.10	165.97	14.40	166.93	165.47	**189.67**	423.20	963.23	171.80	970.03
5000	0.050	n/a	115.13	0.00	*100000.00*	109.27	*2571.60*	109.23	13.97	108.07	15.13	109.07	108.63	**126.60**	449.87	977.50	110.77	983.57
100	(0.06,0.06)	*48.40	45.07	43.60	19.13	41.10	43.43	41.70	4.73	42.13	5.37	42.17	42.87	**48.20**	215.27	1000.00	43.13	1000.00
100	(0.06,0.1)	*45.43	41.73	40.67	20.87	38.37	49.17	37.77	4.93	37.83	5.73	37.77	38.67	**45.20**	280.53	1000.00	39.63	1000.00
100	(0.1,0.1)	*41.30	37.37	36.70	22.93	33.97	49.53	33.93	5.33	34.83	5.83	33.87	34.53	**40.80**	339.13	1000.00	35.13	1000.00
1000	(0.02,0.02)	272.80	224.20	235.07	51.07	209.10	550.63	207.37	8.57	208.70	9.27	208.30	210.70	**241.00**	2.57	850.73	228.83	513.00
1000	(0.02,0.1)	194.10	164.53	169.40	*2334.67*	151.10	552.47	148.80	9.83	149.63	10.63	150.20	150.07	**170.77**	6.80	873.57	167.67★	1000.00
1000	(0.1,0.1)	95.00	80.13	0.03	*100000.00*	74.50	542.33	72.40	10.00	73.40	11.13	74.43	74.43	**84.67**	177.03	840.87	82.97★	1000.00
5000	(0.02,0.02)	266.20	363.90	0.00	*100000.00*	333.53	*2678.93*	330.10	12.23	329.20	12.77	334.47	332.80	**368.53**	244.80	967.83	351.90	963.90
5000	(0.02,0.1)	n/a	256.33	0.00	*100000.00*	237.03	*2639.83*	233.20	13.27	231.83	13.80	234.87	234.37	**261.77**	482.67	985.13	245.03	989.30
5000	(0.1,0.1)	n/a	108.03	0.00	*100000.00*	103.27	*2570.67*	102.67	14.37	102.17	14.47	103.33	101.93	**119.87**	552.30	986.57	104.63	962.67



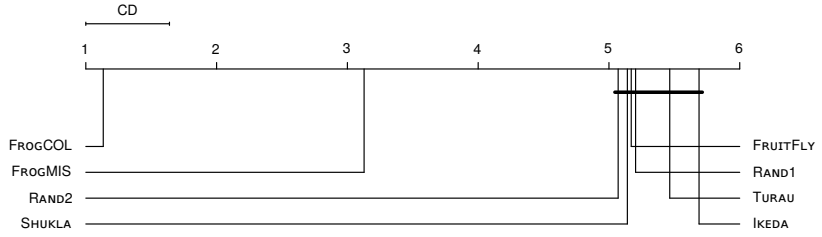
(a) Random geometric graphs.



(b) Random evolving graphs.



(c) Standard random graphs.



(d) Clusterized random graphs.

Figure 3: Critical difference plots.

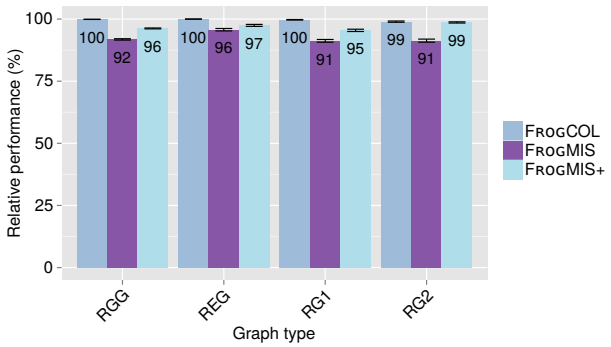
In order to show the impact of the algorithmic features that are removed from FrogCOL in order to obtain FrogMIS, we applied an extended version of FrogMIS, equipped with the master node mechanism for storing the best solution of each run, to all problem instances. This algorithm version is henceforth called FrogMIS+. Figure 4a presents the results of FrogCOL, FrogMIS+ and FrogMIS in terms of their *relative performance*—that is, the best algorithm for a certain setting has always a value of 100%—for each of the four considered graph types. The difference in solution quality between FrogMIS+ and FrogMIS shows the impact on solution quality of not storing the best solution of a run. Moreover, the difference in solution quality between FrogCOL and FrogMIS+ shows the impact on solution quality of only using two color classes instead of using as many colors as required for a coloring solution.

- Except for random geometric graphs of size 1000 and 5000, FrogCOL is able to achieve results comparable to those obtained by Cplex. Moreover, remember that Cplex is, in some cases, not even able to provide a feasible solution within one hour of computation time. This is the case for standard and clusterized random graphs of medium and high density. In order to provide a comparison of FrogCOL, which is the best decentralized algorithm, with Cplex and Greedy, Figure 4b presents the results of these three algorithms in terms of their *relative performance*—that is, the best algorithm for a certain setting has always a value of 100%—for each of the four considered graph types. It can be observed that, even though FrogCOL is outperformed by Cplex and Greedy on random geometric graphs, on average it performs comparable to—or even better than—both centralized algorithms on the remaining graph types.

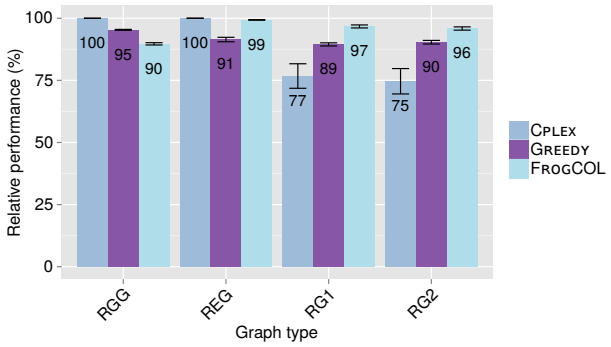
- Interestingly, the performance of Turau, Ikeda, Rand1 and Rand2 is nearly equivalent for what concerns random geometric graphs, standard random graphs, and clusterized random graphs. However, in the case of random evolving graphs, the performance of Turau and Ikeda drops significantly in contrast to the one of Rand1 and Rand2, as can be seen in Figure 3b.

- The critical difference plots (see Figure 3) illustrate the global performance of the algorithms. The following can be observed:

  - There are statistical differences between the first three algorithms—according to the average ranking—when applied to random geometric graphs. In particular, the order of these is FrogCOL, FrogMIS, and FruitFly. In contrast, no statistical differences can be detected among the remaining five algorithms.
  - Regarding random evolving graphs, the best-performing algorithm is again FrogCOL. In contrast to the other types of graphs, the second-ranked algorithm for this type of graphs is FruitFly, followed by FrogMIS and Rand2, which have no significant differences among them. Shukla and Rand1 have a similar peformance and, thus, are clusterized together. Finally, the worst algorithms, with statistical significance, are Turau and Ikeda.



(a) Comparison between FrogCOL, FrogMIS+ and FrogMIS.



(b) Comparison between Cplex, Greedy and FrogCOL.

Figure 4: Comparison of different groups of algorithms, averaged over the four considered types of graphs. Remember that RGG refers to random geometric graphs, REG to random evolving graphs, RG1 to standard random graphs, and RG2 to clusterized random graphs. Algorithms are presented in terms of their *relative performance*—that is, the best algorithm for a certain setting has always a value of 100%—for each of the four considered graph types. Note that the height of the bars (rounded to the closest integers) is displayed within the bars.

– In the context of random graphs—both in the case of standard and clusterized random graphs—FrogCOL is the best algorithm with statistical significance, followed by FrogMIS. No significant differences are detected among the remaining algorithms.

Finally, we aim at confirming the tendency of FrogCOL towards colorings in which the set of nodes that are assigned to the *most-used color* corresponds to a rather large independent set. For this purpose we make use of random geometric graphs, which—among all tested graph types—seem to be most difficult for decentralized algorithms. In particular, we generated 100 random geometric graphs for $n \in \{100, 5000\}$ and for the two radius values that correspond to the lower and upper bounds for the two graph sizes (see also Table 2). In Figure 5, for each of the resulting four cases we display the average color distribution after the first communication round in contrast to the color distribution after convergence. On the x-axis of these graphics we can find the indices of the

used colors. The bars (including the standard deviation) indicate for each color index the number of nodes that are assigned the respective color. For example, the graphic in Figure 5c shows that—in the case $n = 5000$, $r = 0.024$—the color with the lowest index is used by around 895 nodes (on average) after the first communication round. In contrast, the same color is used by around 970 nodes (on average) after the last communication round. This clearly indicates the tendency of the algorithm towards the creation of colorings in which the set of nodes that are assigned to the most-used color corresponds to a rather large independent set. Moreover, the four graphics indicate that this is a general trend, independent of graph size and density, and may serve as an explanation for the success of the algorithm.

5 Conclusions and future work

In this work we, first, studied an existing decentralized algorithm for graph coloring for its ability to generate large independent sets. The algorithm under consideration (FrogCOL) is a bio-inspired technique whose inspiration is the self-desynchronization behaviour that can be observed in the calling of male Japanese tree frogs. In a second step, we have adapted this algorithm in order to directly produce independent set solutions, without the necessity to first produce a graph coloring solution. Both algorithms are compared to a wide range of decentralized algorithms from the literature for the maximal independent set problem. The experimental evaluation has been performed on a large and diverse benchmark set. The results show that, as expected, FrogCOL outperforms FrogMIS. However, both algorithms compare very favorably against the competitors from the literature. As part of future work we intent to study if the FrogCOL algorithm can be applied, respectively adapted, to other problems with relations to graph coloring.

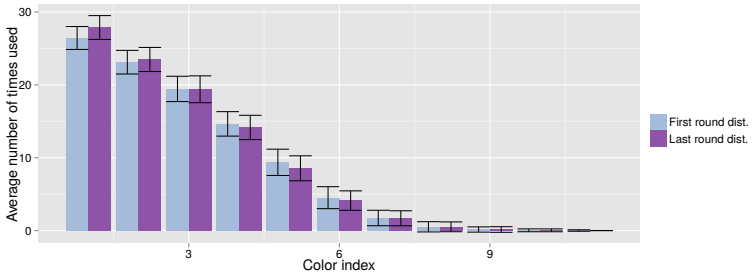
References

[1] Online scientific news site Science*Daily*: Frog calls inspire a new algorithm for wireless networks (2012). URL `http://www.sciencedaily.com/releases/2012/07/120717100123.htm`

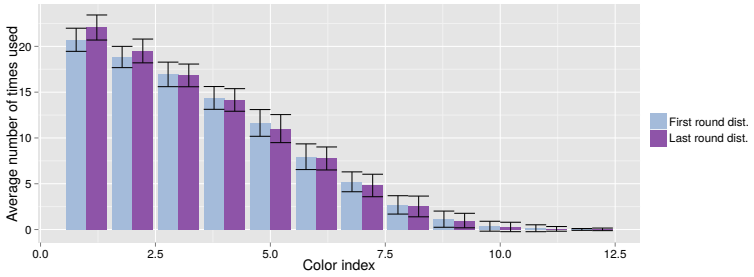
[2] Afek, Y., Alon, N., Barad, O., Hornstein, E., Barkai, N., Bar-Joseph, Z.: A biological solution to a fundamental distributed computing problem. Science **331**, 183–185 (2011)

[3] Aihara, I., Kitahata, H., Yoshikawa, K., Aihara, K.: Mathematical modeling of frogs' calling behavior and its possible application to artificial life and robotics. Artificial Life and Robotics **12**(1), 29–32 (2008)

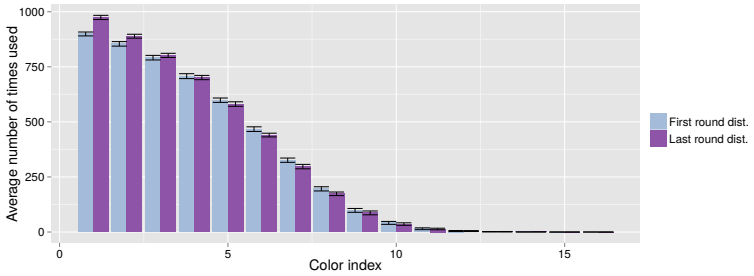
[4] Blum, C., Blesa, M.J., Calvo, B.: Can frogs find large independent sets in a decentralized way? Yes they can! In: M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M.A. Montes de Oca, C. Solnon, T. Stützle (eds.) Proceedings of ANTS 2014 – 9th International Conference on Swarm Intelligence, *Lecture Notes in Computer Science*, vol. 8667, pp. 74–85. Springer Verlag, Berlin, Germany (2014)



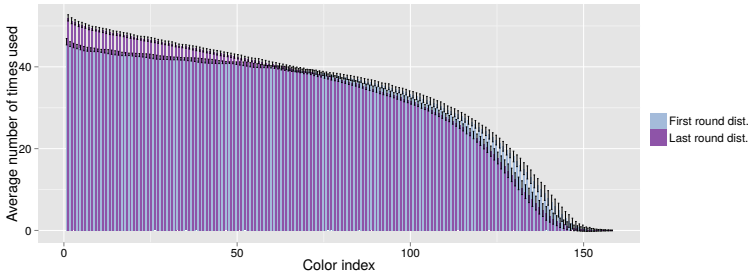
(a) $n = 100$, $r = 0.14$.



(b) $n = 100$, $r = 0.169$.



(c) $n = 5000$, $r = 0.024$.



(d) $n = 5000$, $r = 0.134$.

Figure 5: The graphics show the distribution of the use of different colors both at the start of the FrogCOL algorithm (see *First round dist.*) and after convergence (see *Last round dist.*) averaged over 100 random geometric graphs of sizes 100 and 5000, and different values of r.

[5] Bui, M., Butelle, F., Lavault, C.: A distributed algorithm for constructing a minimum diameter spanning tree. Journal of Parallel and Distributed Computing **64**(5), 571–577 (2004)

[6] Csardi, G., Nepusz, T.: The igraph software package for complex network research. InterJournal Complex Systems **1695**(5) (2006)

[7] Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM **17**(11), 643–644 (1974)

[8] Erciyes, K.: Distributed Graph Algorithms for Computer Networks. Computer Communications and Networks. Springer, London, UK (2013)

[9] Erciyes, K.: Maximal independent sets, chap. 10. In: *Computer Communications and Networks* [8] (2013)

[10] Erciyes, K., Dagdeviren, O., Cokuslu, D., Yilmaz, O., Gumus, H.: Modeling and simulation tools for mobile ad hoc networks, chap. 3. In: Loo et al. [21] (2011)

[11] García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information Sciences **180**(10), 2044 – 2064 (2010)

[12] García, S., Herrera, F.: An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of Machine Learning Research **9**, 2677 – 2694 (2008)

[13] Garey, M.R., Johnson, D.S.: Computers and intractability; A guide to the theory of *NP*-completeness. W. H. Freeman (1979)

[14] Guellati, N., Kheddouci, H.: A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. Journal of Parallel and Distributed Computing **70**(4), 406–415 (2010)

[15] Halldórsson, M.M., Radhakrishnan, J.: Greedy is good: Approximating independent sets in sparse and bounded-degree graphs. Algorithmica **18**, 145–163 (1997)

[16] Hernández, H., Blum, C.: Distributed graph coloring: An approach based on the calling behavior of japanese tree frogs. Swarm Intelligence **6**(2), 117–150 (2012)

[17] Hernández, H., Blum, C.: FrogSim: distributed graph coloring in wireless ad hoc networks — an algorithm inspired by the calling behavior of Japanese tree frogs. Telecommunication Systems **55**(2), 211–223 (2014)

[18] Ikeda, M., Kamei, S., Kakugawa, H.: A space-optimal self-stabilizing algorithm for the maximal independent set problem. In: Proceedings of PDCAT 2002 – Third International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 70–74 (2002)

[19] Jensen, T.R., Toft, B.: Graph coloring problems, *Discrete Mathematics and Optimization*, vol. 39. John Wiley & Sons (2011)

[20] Karp, R.M.: Reducibility among combinatorial problems. Complexity of Computer Computations **40**(4), 85–103 (1972)

[21] Loo, J., Lloret Mauri, J., Hamilton Ortiz, J. (eds.): Mobile ad hoc networks: Current status and future trends. CRC Press, Boca Raton, FL (2011)

[22] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)

[23] Scott, A., Jeavons, P., Xu, L.: Feedback from nature: An optimal distributed algorithm for maximal independent set selection. In: P. Fatourou, G. Taubenfeld (eds.) Proceedings of PODC 2013 – ACM Symposium on Principles of Distributed Computing, pp. 147–156. ACM Press (2013)

[24] Shukla, S.K., Rosenkrantz, D.J., Ravi, S.S.: Observations on self-stabilizing graph algorithms for anonymous networks. In: Proceedings of WSS 1995 – The Second Workshop on Self-Stabilizing Systems (1995)

[25] Turau, V.: Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. Information Processing Letters **103**(3), 88–93 (2007)