International Journal of Computer Science, Communication & Information Technology (CSCIT)
Vol.6 pp.19-26

Self-stabilization on Scale-free Networks

Badreddine Benreguia
Computer Sciences Department
Batna-2 University
Batna, Algeria
badreddine.benreguia@gmail.com

Hamouma Moumen
Computer Sciences Department
Batna-2 University
Batna, Algeria
moumenh@gmail.com

***Abstract*—Many of self-stabilizing algorithms have been proposed in literature to deal with fault-tolerance in distributed systems. Most existing works have utilized random graphs (Erdos-Renyi networks) to simulate self-stabilizing algorithms. In the present paper, we propose the use of self-stabilizing algorithms on scale-free graphs (Barabasi-Albert networks) which are more representative for real networks. After that, we test these algorithms under evolutionary dynamic graphs. Performance is evaluated using extensive simulations where three well known self-stabilizing algorithms are tested: nodes coloring, minimal dominating set and maximal independent set.**

***Index Terms*—self-stabilization, scale-free network, dominating set, independent set, nodes coloring, graph evolution**

I. Introduction

Self-stabilization is a fault-tolerance approach for distributed systems that has been introduced by Djikistra in 1974. A self-stabilizing distributed system is guaranteed to achieve global correct configuration, in a finite time, even with presence of faults inside the system. Suppose a distributed system is in illegitimate configuration after a transient fault, self-stabilization allows to reach a legitimate configuration without any external intervention. Various self-stabilizing distributed algorithms have been proposed in the literature for graph problems such as leader election, nodes coloring, domination problem, independent set identification, construction of spanning tree. A detailed taxonomy of different self-stabilizing algorithms can be found in [1].

Generally, graphs are used to represent networks and complex systems in real-life applications like internet, social networks, network of nerve cells, power grids, systems of transportation, protein particulars, and a lot of other systems that could be found in [2].

Most of the papers have studied self-stabilization on theoretical level using simple undirected graphs and only few of works have used experimental tests on random graphs known as Erdos-Renyi graphs [3]. In fact, it has been proved that most of real networks cannot be represented by the model Erdos-Renyi. Barabasi and Albert have proposed a model known as scale-free network or Barabasi-Albert model [4]. After collecting a set of statistics about real existing networks (like www network, Movie actors, Paper citations) authors study some parameters such as mean distance between two nodes, clustering coefficient, degree distribution [2], [5]. For example in random graphs, degree of the nodes is approximately the same for all the nodes where the degree distribution follows Normal law. This distribution cannot explain the notion of the hubs in the real complex networks where a few of nodes have a very high level of degree. In the proposed model of Barabasi and Albert, the degree is distributed the same manner in real networks following a Power law distribution. Currently, Barabasi-Albert graphs are the most known graphs which are widely used by the community of graph literature.

A. Contribution

The previous works, proposed in literature, have implemented self-stabilization only for random graphs and static structure of graphs. Although Dolev *et. al.* have indicated that uniform self-stabilizing algorithms can work under dynamic graphs [6], there is no work that simulates self-stabilization in the dynamic context where the graph structure is changing. Our main contributions can be summarized as follows:

(i) We use and test three particular self-stabilizing algorithms under scale-free graphs. The algorithms are: minimal dominating set, maximal independent set and nodes coloring.

(ii) We test self-stabilization under graph growing where at any time, a new node comes to connect the existing graph. We attempt to observe the behavior of the self-stabilizing algorithms under the dynamic structure of the graphs.

B. Organization of the paper

The remainder of this paper is structured as follows. Section II discusses the concept of self-stabilization. We give particular explanation for three self-stabilizing algorithms $i.e.$ minimal dominating set, maximal independent set and nodes coloring. In section III, scale-free graphs and complex networks are presented. We introduce our version of the algorithm generating scale-free networks in section IV. Section V presents the idea of integrating self-stabilization for evolutionary graphs. Simulation tests and experiments are shown in section VI followed by a conclusion in section VII.

II. Self-stabilization problem formulation

In a self-stabilizing system, all the nodes have the same collection of rules under the form: **if** $guard$ **then** $statement$ (written generally as: $guard \longrightarrow statement$) and the same local variables that describe the node's $state$, where the guard part is a set of boolean expressions and the statement is an action on the node's state to be committed if the guard is true. The $state$ of every node can be updated by the node itself

ISSN 1737-930X

using its rules. Each node has a partial view of the distributed system ($i.e.$ $guard$ which consists of boolean expressions) on its state and the states of the neighbors. A rule is said $enabled$ (or $priviliged$) if the guard is evaluated to be true. A node will be enabled if at least one of its rules is enabled. Executing the statement of the enabled rule by the node is called a $move$. This execution allows updating the state (local variables) of the node in order to be more legitimate with its neighborhood.

We are interesting in this work on *uniform algorithm*, where all nodes in the distributed system execute the same program or check the same set of rules. If there is no enabled rule for all the nodes, the system is in the legitimate global configuration. However, if there is at least one enabled rule (a move) in the overall system, the system is considered not yet stable and it is expected that the network will be in the correct global configuration after executing a finite number of moves. The execution of self-stabilizing algorithms is managed by a daemon (scheduler) that selects the privileged nodes to move from a configuration to another configuration. Two types of daemons are widely used in self-stabilization literature: central and distributed daemons. In the central daemons, one privileged node is selected among all the privileged nodes to be moved. However, in the distributed daemons, a subset of nodes are selected among the set of privileged nodes to make a move simultaneously. Detailed taxonomy introducing various daemons can be found in [7]. In this paper, we discuss particularly three well-known algorithms of self-stabilization: finding maximal independent set, detection of minimal dominating set and nodes coloring.

A. Maximal Independent Set (MIS)

Let $G = (V, E)$ be a graph, where V is the set of nodes and E is the set of edges. An independent set is a subset of nodes $S \subseteq V$ such that there is no two nodes connected in S. The set S is said *maximal* if there is no superset S' of S such that S' is an independent set. Maximal independent set (MIS) is used in many practical applications like head clusters in sensor networks. Shukla *et al.* [8] have proposed a self-stabilizing algorithm to find the MIS. The idea is very simple: a node v joins the set S if v has no neighbor in S, and v leaves the set S if at least one of its neighbors is in S. Each node v has a local variable called *ind* that takes one value from $\{0, 1\}$. When $v.ind = 1$ that indicates v is in S otherwise $v.ind = 0$. In the legitimate global configuration, the set of nodes $\{v, v.ind = 1\}$ is MIS. Every node v has to check the following two rules:

Rule 1: $v.ind = 0 \wedge \forall u \in N(v) : u.ind = 0 \longrightarrow v.ind = 1$
Rule 2: $v.ind = 1 \wedge \exists u \in N(v) : u.ind = 1 \longrightarrow v.ind = 0$

Where $N(v)$ is the set of v neighbors. It has been shown that MIS self-stabilizing algorithm converges in $O(n)$ moves under central daemon [8].

B. Minimal Dominating Set (MDS)

In a graph $G(V, E)$, a set of nodes $S \subseteq V$ is called a dominating set if every node $v \in V$ is either a member of S or is neighbor to a node of S. A dominating set S is minimal if no proper subset of S is a dominating set.

Hedetniemi *et al.* [9] has proposed a self-stabilizing algorithm for the minimal dominating set (MDS) problem. Each node has a boolean variable that indicates whether it is in S or not. To find a dominating set, the algorithm is based on the following idea: a node joins the set S, if it has no neighbor in S. A node that is a member of S, and has a neighbor that is also a member of S, will leave the set if all its neighbors are not pointing to it. Thus, after stabilization the set S will be MDS. Although, the algorithm stabilizes in $O(n^2)$ using central daemon, another proposed MDS algorithm [10] converges to the global correct configuration in $O(n)$.

Domination has been widely studied in literature and has been adopted in many real-life applications. Address routing, power management, clustering in ad-hoc networks and influence opinion in social networks are some examples of domination application [11]–[14].

C. Nodes coloring

Coloring problem consists to assign one color to each node such that no two adjacent nodes could have the same color. Mathematically, for a graph $G(V, E)$, coloring nodes is a function $c : V \rightarrow N$ where $c(i) \neq c(j)$ if i and j are adjacent.

The elements of N are called the available colors. If a graph G may be colored using k colors, we say that is $k-colorable$. The smallest value of k is called the *chromatic number*. Many self-stabilizing algorithms have been proposed in literature. Hedetniemi *et al.* [15] have proposed grundy coloring using at most $(\Delta + 1)$ colors where Δ is the maximal degree in the graph $i.e.$ $k < \Delta$. The drawback of this algorithm is that each node must to know Δ which is considered as global information. In self-stabilizing systems, it is preferable that nodes know only a partial information about their neighborhood. The algorithm converges to the legitimate global configuration in $O(n)$. Coloring is used generally for channel and frequency assignment in wireless networks.

III. Complex networks

Erdos and Renyi have proposed in 1959 a first model of graphs based on the concept of randomness where every edge has a probability to exist or to not-exist. The growing interest in complex systems has led to collecting tremendous statistics on real networks. In the past few years, many empirical results showed [2], [5] that the model of Erdos-Renyi is very limited and cannot represent large-scale real networks. For example, the distribution of degree in random graphs, that follows Normal law, cannot explain the notion of *hubs* in the real complex networks where a few of nodes have a very high level of degree.

Thus, Barabasi and Albert [4] have proposed the scale-free model which has two main properties: *graph evolution* and *preferential attachment*. For the first property, authors assume that networks are constructed under time by growing. At any moment, there is a new node that comes to connect the existing graph (born of a new node) with an average of connection called *attachment degree*. The attachment degree

Algorithm 1 Generating Scale-free Graph

```
Input: $n$: graph size
       $d$: attachment degree
Output: $g$: a scale-free graph
1: Let $g$ be an initial full-connected subgraph of $d$ nodes
   where each node has $Deg = d - 1$
2: for $i = d + 1$ to $n$ do
3:   Let $i$ be a new node of $g$ with $Deg_i = 0$
4:   Let $AttachDeg_i$ be a random integer in $[1, d]$
5:   for $k = 1$ to $AttachDeg_i$ do
6:     $neighbor \leftarrow$ Select an existing node of $g$ proportion-
       ally to its degree
7:     Connect $i$ to $neighbor$
8:     $deg_{neighbor} \leftarrow deg_{neighbor} + 1$
9:   end for
10:  $deg_i \leftarrow AttachDeg_i$
11: end for
12: return $g$
```

is a parameter that indicates the average number of neighbors like the average number of friends in a given social network. The second property, every new node must to connect to the existing nodes using *preferential attachment* law *i.e.* nodes having high degree have more chances to be selected as neighbors by the new node. Thus, each new node connects to existing nodes with probability proportional to their degrees.

Scale-free networks have an important feature that appears under the evolution process. Some nodes will be highly connected and will be more attractive to be neighbors for the new nodes, called $hubs$. This process is known as $rich-get-richer$ [2].

Instead using Erdos-Renyi graphs, we believe that the presence of the $hubs$ in Barabasi-Albert model will give a different behavior for self-stabilizing algorithms. In this sense, we attempt to test self-stabilization under Barabasi-Albert graphs. Firstly, using a set of well-known self-stabilizing algorithms, we try to establish a comparison process of self-stabilization under random graphs and scale-free networks. Secondly, we check the feasibility of self-stabilization under graph evolution. Notice that previous works has supposed that self-stabilization is suitable only for fault-tolerance like dealing with problems of memory corruption.

IV. Generating scale-free networks

In this section, we present our algorithm used for generating synthetic Barabasi-Albert graphs. Although, there exists tools to generate scale-free graphs like Networkx of Python, the need to integrate the self-stabilizing algorithms under the evolution of the graph leads us to propose our own generator shown in algorithm 1.

The algorithm starts using an initial subgraph of d nodes. In our case, we have used a full-connected graph as an initial subgraph, but it is possible to utilize other structure, like full-disconnected subgraph which is used in Networkx. Note that in reality, d is always less and very small than n. In the next step, the graph evolution is represented by the coming of new nodes one-by-one. Each new node connects to the existing graph by preferential attachment where the degree of attachment of the new node is a random value in $[1, d]$. In the case of Networkx, the generator uses a constant attachment degree as d. However, in our case we have used a random integer in $[1, d]$ because, in reality, new nodes connect with value close to (not fixed) attachment degree.

Table I illustrates an example of generating three models of graphs. The generated graphs have size of 1000 nodes using an attachment degree of 10 for scale-free graphs and a probability $p = 0.01$ for random graphs $i.e.$ every node has likely 10 neighbors. The degree distribution is shown in table I. It indicates clearly the differences in degree distribution. For random graphs, most of nodes have degree between 1 and 20. Generally the degree distribution fits a Normal law for random graphs (where the mean equals 10 in this example). For the scale-free graphs, the two generators give frequencies that follow the well known power law distribution in Barabasi-Albert model [5]. As for the Networkx generator, it gives values of degree greater or equal to 10. This is due to the manner the generator connects new coming nodes where every node has to attach exactly to 10 neighbors. Thus, no node could have a degree less than 10. This later problem does not appear in our case because we use random values in $[1, d]$ as attachment degree to connect new nodes. It is possible to see that the two algorithms gives hubs having degree greater than 100. Our generator gives 2 hubs while in the second there is 11 hubs, one of them has 177 neighbors. It is worth indicating that we have observed during evolution, the hubs are generally the nodes which are created at the beginning of the evolution. For example in the case of Networkx generator, the hub (of 177 degree) is originally the first node coming directly after constructing the initial subgraph. For our generator, the two hubs are the nodes created during evolution at the moments 2 and 3 respectively. This does not reflect the reality because hubs may be not created at the beginning of evolution in complex networks. For example, in the network of paper citation [2], hubs (papers highly referenced) could be created at the middle of evolution.

TABLE I
Degree distribution on three examples of synthetic graphs

	Number of nodes		
Degree	Random generator Erdos-Renyi	Our generator Barabasi-Albert	Networkx generator Barabasi-Albert
1-9	468	637	0
10-19	529	262	736
20-29	3	51	134
30-39	0	15	51
40-49	0	11	25
50-59	0	10	15
60-69	0	6	10
70-100	0	6	18
101-177	0	2	11

V. Slef-stabilization under graph evolution

All the previous works in literature have indicated that self-stabilization is more suitable for fault-tolerance in distributed systems. In this paper, we show that self-stabilization is suitable too for the natural process of evolution that exists in complex networks. The idea of integrating self-stabilization is as follows: suppose at time t_n a graph G is stable (in legitimate configuration) for a given algorithm. At time t_{n+1} a new node $(n+1)$ comes to join the graph G. We suppose that at the connection moment, the new node selects a random state for itself. For example, in the MDS problem, the new node selects randomly a value from $\{0,1\}$ to indicate if it is in MDS or not. The connection of the new node with its random state will probably make the graph G in illegitimate configuration. Thus, the process of self-stabilizing must to react in order to move to the global legitimate configuration again. Note that the algorithm re-stabilizes after the addition of just one node.

Algorithm 2 Self-stabilization Under Evolution

```
Input: $g$: graph in legitimate configuration of $n$ nodes
       $d$: attachment degree
       $k$: number of nodes will connect $g$
Output: $g$: graph in legitimate configuration of $n+k$ nodes
1: for $i = 1$ to $k$ do
2:     Let $i$ be a new node with a random state
3:     $g \leftarrow$ Connect $i$ to $g$ with preferential attachment
4:     Reaction of self-stabilizing algorithm
5: end for
6: return $g$
```

For a long period of evolution $i.e.$ lot of new connecting nodes, the algorithm must to stabilize after adding new one node before the coming of a second new node. It is possible to consider the coming of new node like a fault that occurs on the distributed system which the self-stabilizing algorithm has to deal with it. Generally, it is assumed that the interval between two faults (two new nodes) is so long to allow the system executing its original task during the stable period.

VI. Simulations results

We conduct simulations on three sides: (1) executing self-stabilization on scale-free graphs (2) comparing self-stabilization on random graphs and scale-free graphs (3) and test self-stabilization under graph evolution. Three self-stabilizing algorithms are used in the experiments: maximal independent set [8], [9], minimal dominating set [9] and grundy coloring [15]. To check the efficiency : (i) we calculate the speed of convergence (number of moves needed to achieve legitimate configuration) and calculate a second parameter that depends on the kind of the algorithm. For minimal dominating set, we seek the cardinality of MDS. For maximal independent set, the cardinality of MIS is taken into account. In the coloring problem, the goal is to find the minimal number of colors. Synthetic generated graphs, integration of self-stabilization with evolution process and benchmark tests have been written in Java. However, we have used the Kuszner code [16] to execute the above self-stabilizing algorithms under central daemon. We take the average of calculation after conducting 5 tests in each experiment.

A. *Attachment degree impact on self-stabilization*

In this section, we study the behavior of self-stabilizing algorithms under the impact of changing the attachment degree. Curves are plotted using values of attachment degree ranging from 2 to 100. For each algorithm, two sizes of graphs are used: graphs with 1000 nodes and graphs with 5000 nodes.

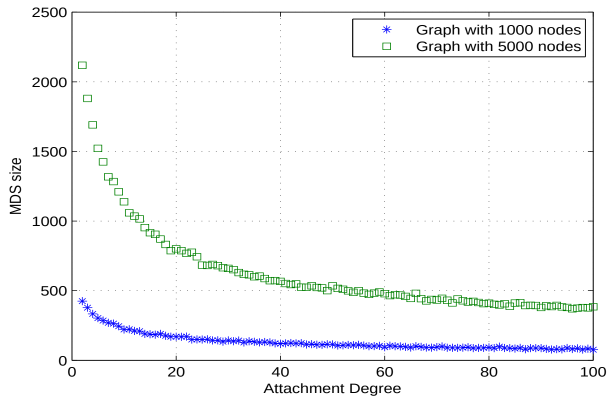

Fig. 1. Size of MDS according attachment degree

1) MDS: Figure 1 explains the relation between cardinality of MDS and attachment degree. The MDS size is inversely proportional to the attachment degree $i.e.$ it decreases when the attachment degree grows. This is a rational result because it is expected that the cardinality will decrease when the graph density grows. For example in full-connected graph, the MDS cardinality is 1. It is worth to indicate that the minimal obtained rate of MDS represents 7% of the entire population of nodes (374 nodes in MDS from 5000 when $AttachDeg = 97$).

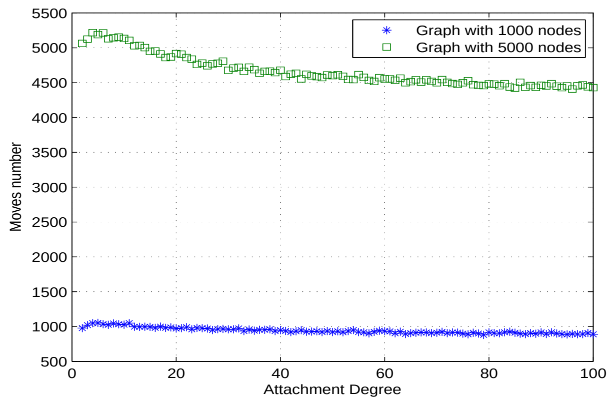

Fig. 2. Time of convergence of MDS according attachment degree

Figure 2 shows the necessary number of moves in order to converge to the legitimate configuration. It illustrates (nearly) a constant plot which means that the time of convergence (number of moves) and graph density (attachment degree) are independent. Observe that the top values of the plots (moves number) are located in the positions of the maximal values of cardinality shown in figure 1. Although Hedetniemi [9] has proved that MDS algorithm stabilizes in $O(n^2)$, simulations show a convergence in $O(n)$. This later complexity has been proved by Turau [10].

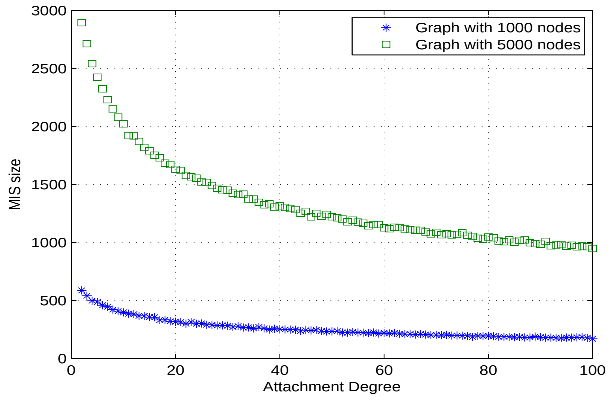

Fig. 3. Size of MIS according attachment degree

2) MIS: In figure 3, for graphs of 5000 nodes, the MIS size is in the interval $[950, 2900]$. As for graphs of 1000 nodes, the MIS has size in $[170, 590]$. When the graph density (attachment degree) grows, the cardinality of MIS decreases $i.e.$ size of MIS is inversely proportional to the attachment degree. According figures 1 and 3, MDS sets are always smaller than MIS.

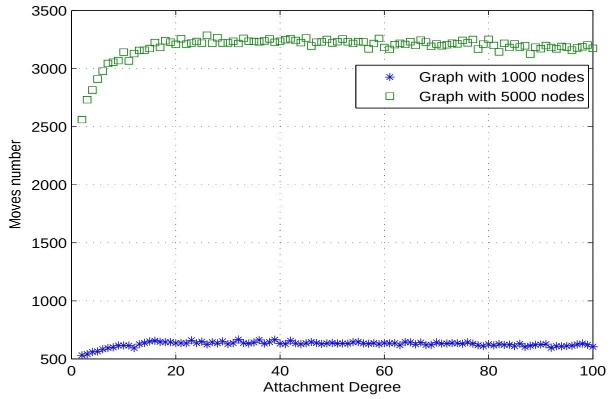

Fig. 4. Time of convergence of MIS according attachment degree

Figure 4 shows the time of convergence for MIS stabilization. When the degree attachment exceeds 10, time of stabilization becomes almost constant. For graphs of 5000, the number of moves is less than 3300 moves, and less than 700 for graphs of 1000 nodes. These results ensure the formal proof that MIS stabilizes in $O(n)$ [8].

3) Coloring: Figure 5 illustrates that 7 to 52 colors are needed for coloring scale-free graphs of 1000 nodes. Number of colors is proportional to the graph density (attachment degree). For graphs of 5000, we need a number of colors between 8 and 71. Note that theoretically, the maximal value of colors is $(\Delta+1)$ where Δ is the maximum degree in the graph [15]. Our tests on scale-free graphs show that values given by simulations does not reflect perfectly theoretical results. In the presence of the hubs, Δ will be very high whereas the number of colors, given in experiments, is very small comparing to $(\Delta + 1)$. For example, graphs of 5000 nodes (with attachment degree of 100) give easily hubs having more than 900 neighbors whereas the maximal number of colors will not exceed 80. Figure 6 ensures theoretical complexity

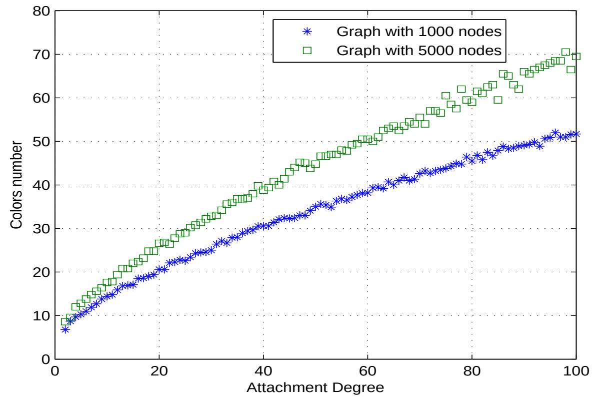

Fig. 5. Number of colors according attachment degree

obtained in [15] that grundy coloring converges in $O(n)$.

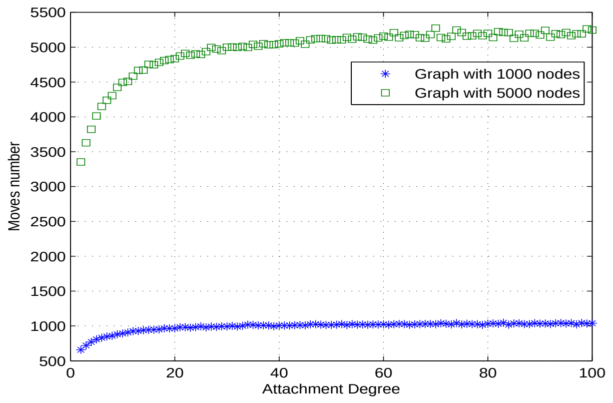

Fig. 6. Time of convergence of nodes coloring according attachment degree

B. Comparison

In this section, we try to observe the behavior of self-stabilizing algorithms on random graphs and scale-free graphs. We use graphs having sizes of 1000, 2000, ...,10000 nodes. For more accurate results, we generate graphs having nearly the same density (number of edges). For example, the keyword used for curves legend: *Scalefree10* is used for scale-free graphs and *Random10* for random graphs where in both cases, most of nodes have likely 10 neighbors. The same concept is adopted for *Random50* and *Scalefree50* where every node has high probability to be adjacent to 50 neighbors. These values have been chosen according statistics given in previous works [2], [5] where many of real scale-free networkx have an average degree approximated to the taken values.

1) MDS: Random graphs give sets smaller than scale-free graphs as shown in figure 7. In fact, up to now, we have no explanation why random graphs give cardinality smaller than scale-free graphs. In other side, plots given in this figure confirm previous results that: more the graphs are dense, smaller MDS are given. However, for the time of convergence,

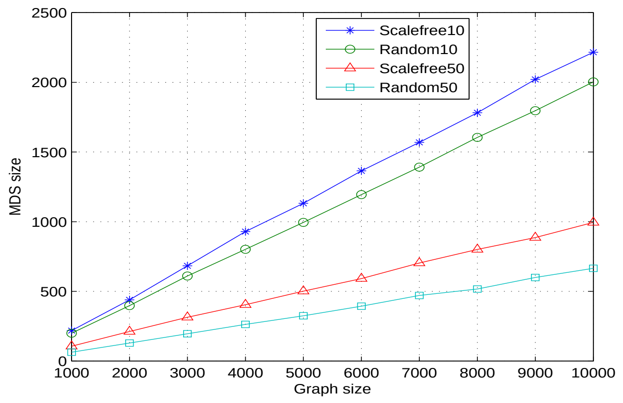

Fig. 7. MDS size in random graphs and scale-free graphs

figure 8 illustrates that calculating MDS in scale-free graph is faster than random graph. This is due to the cardinality where MDS given by random graphs are smaller than scale-free graphs, thus optimal sets need more time to reach legitimate configuration.

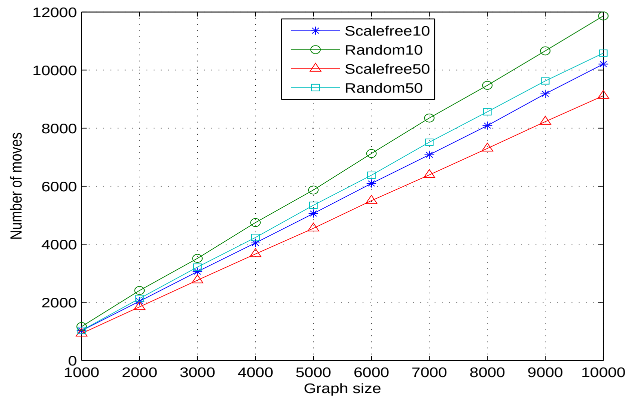

Fig. 8. Time of convergence (MDS) in random graphs and scale-free graphs

2) MIS: Recall that in MIS, we aim to maximize the set of independent nodes unlike MDS where the set is minimized. In figure 9, scale-free graphs give MIS sets bigger than random graphs.

Figure 10 shows that MIS of random graphs is found quickly than scale-free graphs. More the MIS is bigger (maximal), self-stabilizing algorithm needs more time to stabilizes. Bigger sets will need an increasing number of moves to be found.

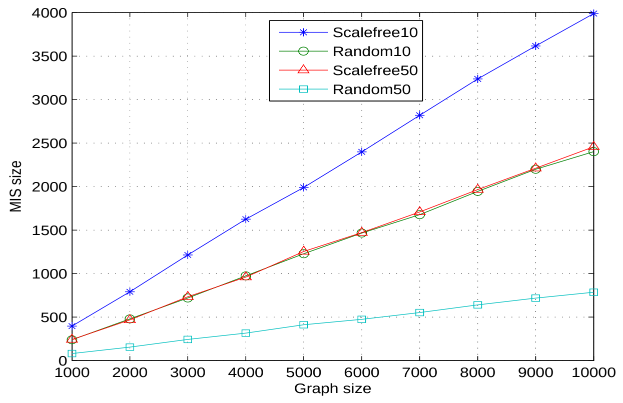

Fig. 9. MIS size in scale-free and random graphs

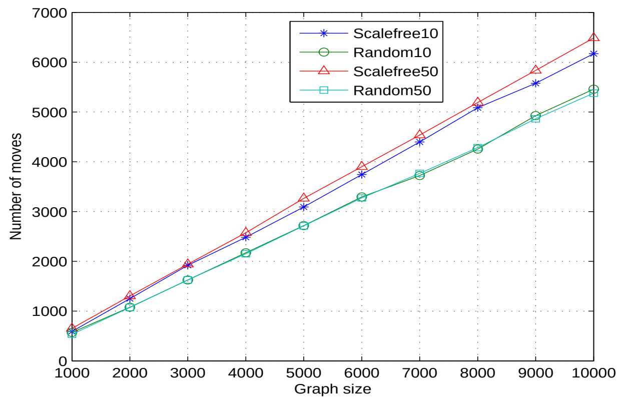

Fig. 10. Time of convergence (MIS) in scale-free and random graphs

3) Coloring: The important observation given by figure 11 is the constant number of colors needed in random graphs whatever the size of graphs. The number of colors is independent from graph growing because the average degree remains constant whatever the size of the graphs. It shows that for random graphs with an average degree of 10, we need at most 8 colors and for random graphs with an average degree of 50, we need only 20 colors. For the scale-free graphs, the number of colors is proportional to the attachment degree. It increases slowly when the attachment degree grows. Generally, scale-free graphs with large scale of nodes, contains hubs of high degree. According [15] where the number of colors is dependent to Δ, more the graph grows, hubs will use a greater number of colors.

In figure 12, scale-free graphs find colors quickly than random graphs. It is clear when the interval of choices (colors) is large, it will be easy to find the colors.

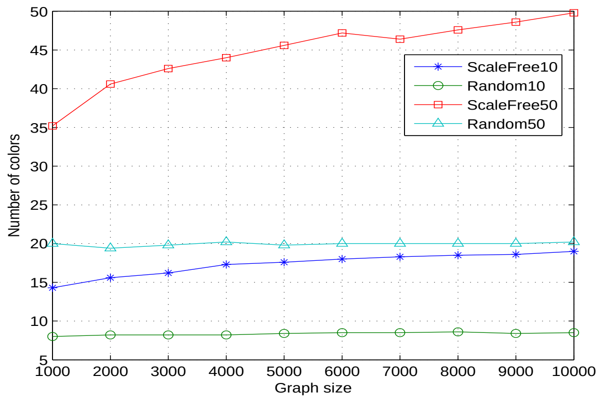

Fig. 11. Colors number in scale-free and random graphs

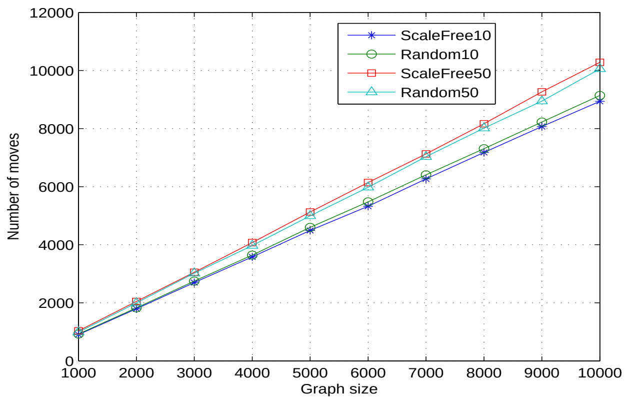

Fig. 12. Coloring convergence in scale-free and random graphs

C. Evolution

In the previous simulations, graphs are supposed static. Before activating the self-stabilization in static graphs, we assign to each node a random state. After that the algorithm starts until it converges to the legitimate global configuration. However, in this section, we try to show that uniform self-stabilizing algorithms can be integrated easily in Scale-free graphs during the evolution process. The graph is supposed stable (in legitimate configuration) before the attachment of a new node. When the new node joins the existing graph, the self-stabilizing algorithm starts execution in order to move to the legitimate global configuration again.

In the experiments, we have generated a graph of 10000 nodes integrating self-stabilization (using the above three algorithms). According Algorithm 1, after constructing the initial subgraph of 10 nodes (attachment degree=10), the evolution process begins immediately with the coming of 11^{th} node. The self-stabilizing process starts at the moment the 11^{th} node joins the graph. At this moment, the algorithm has to converge to the correct configuration (the graph is now formed with 11 nodes). Every time there is new node who joins the existing graph, self-stabilizing process is reacting to correct the graph configuration because there is a probability that the new node dis-stabilizes the legitimate configuration. The alternative process, add new node / reaction of self-stabilization, is executed from the joining of the 11^{th} node to the 10000^{th} node.

Figure 13 shows a partial interval of the alternative process, add node / reaction of self-stabilization, between the coming of nodes 5010 and 5020. Self-stabilizing algorithms give different results. Generally, there is two cases. In the first case , the new node does not influence the global configuration of the graph where there is no executed moves. In this case where the number of moves is not growing, the new node joins the graph selecting a random state without activating its own rules of self-stabilization which means that the new node is in a correct state at least with its neighborhood. In the second case, the number of moves grows when just one new node joins the existing graph. The new node dis-stabilizes the global configuration by executing at least one move which means that the node or one of its neighbors has enabled a rule in order to correct the global configuration. The maximal number of moves executed, for a new node addition, is given by node 5019 in the MDS algorithm where 4 moves are executed for the stabilization of the algorithm (from 9956 to 9960). For 10 added nodes, the coloring problem is the algorithm that needs a greater number of moves for stabilization (18 moves from 7060 to 7078) whereas the MIS algorithms needs 5 moves (smaller number) in the interval of the 10 new nodes. Other

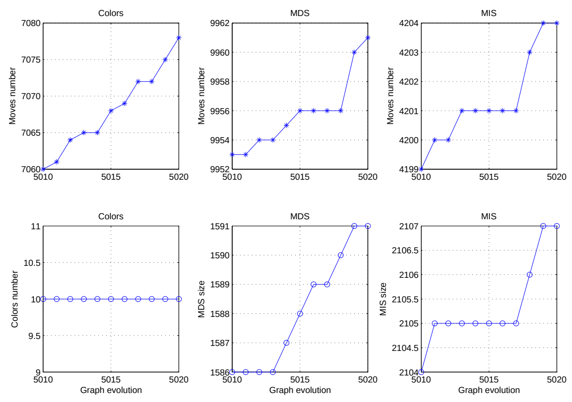

Fig. 13. Self-stabilization for 10 new nodes attaching with degree 10 on scale-free graph of 5010 nodes

results shown by figure 13 are the following. The coloring problem still uses a constant number of colors in this interval. MDS cardinality is increased by including new 5 nodes in the MDS from the 10 possible. The size of MIS grows by adding 3 new nodes in MIS from 10 nodes possible.

Table II gives some results extracted from the entire evolution from the 11^{th} node to the 10000^{th} node. Although these results are calculated for evolutionary graphs, it will be possible to compare them with static graphs. For the convergence time, static graphs seems faster than evolutionary graphs (according plots $ScaleFree10$ in figures 8, 10 and 12). Recall that dynamic graphs stabilizes gradually $i.e.$ a new

TABLE II
RESULTS OF SELF-STABILIZING ALGORITHMS DURING GRAPH EVOLUTION

	MDS		MIS		Coloring	
Graph	Moves	Size	Moves	Size	Moves	Colors nbr
11	12	5	1	5	7	3
1000	1912	297	849	418	1442	9
2000	3989	627	1733	858	2852	10
3000	6024	936	2525	1280	4237	10
4000	7958	1252	3398	1698	5650	10
5000	9924	1588	4191	2097	7038	10
6000	12043	1877	5071	2499	8364	10
7000	14138	2187	5909	2889	9787	10
8000	16235	2487	6774	3305	11190	10
9000	18332	2809	7593	3721	12587	10
10000	20317	3139	8493	4144	13949	10

graph with n nodes corrects its configuration knowing that the old graph of $n-1$ is in legitimate configuration. Only the new joining could move to an illegitimate configuration. However in the static graphs, the global legitimate configuration must be achieved after starting from an unknown global random state. Moreover, comparing table II with figures 7, 9 and 11 shows that static graphs gives smaller MDS sets. Whereas dynamic graphs produce maximal sets for MIS and give smaller number of colors for the coloring problem.

VII. CONCLUSION

In this paper, self-stabilization is tested under scale-free graphs using a set of well-known algorithms. We have illustrated also how self-stabilization works under graph evolution. This later demonstration is important if we know that most of real networks are constructed under evolution. From a point of view application, our work shows that self-stabilization is suitable for distributed systems where the structure is in continuous change according to the evolution process introduced by Barabasi.

Our simulation tests give various results under scale-free graphs, particularly for the problems of: graph coloring, detecting minimal dominating set and finding the maximal independent set.

In the future, we hope to test self-stabilization on graphs with more dynamic structures such as nodes leaving. We will try to interpret some results like the reason random graphs produce smaller sets of MDS and MIS than scale-free graphs.

ACKNOWLEDGEMENTS

The authors would like to thank N. Guellati, C. Titouna and the referees for their valuable comments in order to improve the paper.

REFERENCES

[1] N. Guellati and H. Kheddouci, “A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 406–415, 2010.
[2] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” *Rev. Mod. Phys.*, vol. 74, no. 1, pp. 47–97, 2002.
[3] B. Neggazi, N. Guellati, M. Haddad, and H. Kheddouci, “Efficient self-stabilizing algorithm for independent strong dominating sets in arbitrary graphs,” *International Journal of Foundations of Computer Science*, vol. 26, no. 06, pp. 751–768, 2015.
[4] A.-L. Barabasi and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[5] X. F. Wang and G. Chen, “Complex networks: small-world, scale-free and beyond,” *IEEE circuits and systems magazine*, vol. 3, no. 1, pp. 6–20, 2003.
[6] S. Dolev, A. Israeli, and S. Moran, “Self-stabilization of dynamic systems assuming only read/write atomicity,” *Distributed Computing*, vol. 7, pp. 3–16, 1993.
[7] S. Dubois and S. Tixeuil, “A taxonomy of daemons in self-stabilization,” *CoRR*, vol. abs/1110.0334, 2011.
[8] S. K. Shukla, D. J. Rosenkrantz, and S. S. Ravi, “Observations on self-stabilizing graph algorithms for anonymous networks,” in *PROCEEDINGS OF THE SECOND WORKSHOP ON SELF-STABILIZING SYSTEMS*, 1995, pp. 1–7.
[9] S. Hedetniemi, S. Hedetniemi, D. Jacobs, and P. Srimani, “Self-stabilizing algorithms for minimal dominating sets and maximal independent sets,” *Computer and Mathematics with Applications*, vol. 46, no. 56, pp. 805–811, 2003.
[10] V. Turau, “Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler,” *Information Processing Letters*, vol. 103, no. 3, pp. 88–93, 2007.
[11] D. Li, L. Liu, and H. Yang, “Minimum connected r-hop k-dominating set in wireless networks,” *Discrete Mathematics, Algorithms and Applications*, vol. 1, no. 1, pp. 45–57, 2009.
[12] K. Alzoubi, P.-J. Wan, and O. Frieder, “Maximal independent set, weakly connected dominating set, and induced spanners in wireless ad hoc networks,” *International Journal of Foundations of Computer Science*, vol. 14, no. 2, pp. 287–303, 2003.
[13] Y. Ding, J. Z. Wang, and P. K. Srimani, “A linear time self-stabilizing algorithm for minimal weakly connected dominating sets,” *International Journal of Parallel Programming*, vol. 44, no. 1, pp. 151–162, 2016.
[14] Y. Ding, J. Wang, and P. Srimani, “Self-stabilizing selection of influential users in social networks,” in *17th International Conference on Computational Science and Engineering*. IEEE, 2014, pp. 1558–1565.
[15] S. T. Hedetniemi, D. P.Jacobs, and P. K.Srimani, “Linear time self-stabilizing colorings,” *Information Processing Letters*, vol. 87, no. 5, pp. 251–255, 2003.
[16] L. Kuszner, “Tools to develop and test self-stabilizing algorithms,” *http://kaims.eti.pg.gda.pl/ kuszner/self-stab/main.html*, 2005.