Simulate to detect: a multi-agent system for community detection

Remy Cazabet
IRIT
Toulouse University
Toulouse, France
cazabet@irit.fr

Frederic Amblard
IRIT-UT1
University of Social Science
Toulouse, France
frederic.amblard@univ-tlse1.fr

***Abstract*—Community detection in social networks is a well-known problem encountered in many fields. Many traditional algorithms have been proposed to solve it, with recurrent problems: impossibility to deal with dynamic networks, sensitivity to noise, no detection of overlapping communities, exponential running time... This paper proposes a multi-agent system that replays the evolution of a network and, in the same time, reproduces the rise and fall of communities. After presenting the strengths and weaknesses of existing community detection algorithms, we describe the multi-agent system we propose. Then, we compare our solution with existing works, and show some advantages of our method, in particular the possibility to dynamically detect the communities.**

***Keywords*-Multi-agent simulation, Community detection, Dynamic networks, Social Networks**

I. INTRODUCTION

In the last few years, the democratization of the web 2.0 produced new kinds of network data. With millions of people interacting by means of digital media – Web 2.0 social networks like Facebook or Twitter, Media sharing website like Flickr or Youtube, message boards, ... – it becomes easier and easier to extract very large datasets of interaction and communication among individuals. As a consequence, much work has been done recently to study these networks. One of the most interesting and challenging problem deals with community detection.

A community in a network is frequently defined as a set of nodes that are more strongly connected among each others than with the remaining part of the network. This definition stays quite imprecise, because in many networks we know that there are some communities, we are able to identify them intuitively, but there is no precise definition for them. For example, think about your social network, as it could be extracted from Facebook: you are probably "friend" with many people from your family, and it is likely that most of these people are also connected the one to the others. But you may also have as "friends" some of your co-workers, which again know each other. Same thing with former classmates, and many other possible categories. In a perfectly classified world, these groups of people would be totally separated and their identification would be very easy, a simple identification of connected components. However in real life, some people from one community are also connected with people from another one.

Many algorithms exist to try to identify communities. One of the first very efficient solution was proposed by Girvan and Newman [1]. This method and most of the other ones proposed since then have a centralized and global approach. They first define a metric – often the modularity –, which, from a network and a given decomposition, gives a value which represents the "quality" of this decomposition. They then try to optimize this value for the network using different metaheuristics and/or adaptations of the metric.

However these methods have two major drawbacks:

- They are not able to deal with overlaps between communities
- They are not able to deal with dynamic networks

The first limitation comes from at least two reasons. First, the computation time becomes much larger when one allows communities to overlap, because the range of possible decompositions grows tremendously. Secondly, as the approach is global, it is not the local quality of each community that is evaluated, but rather the decomposition as a whole. It is then quite difficult using this way to find a metric able to compare, for instance with a network of 100 nodes, a solution having 10 communities of 10 nodes with another one having 20 communities of 10 nodes.

The second limitation, dealing with dynamic data, seems even harder to overcome with these traditional techniques. Indeed, all these algorithms are working on a snapshot of the network at a given time, or an aggregation of data over time. Some approaches [2] have tried to compute detections on several snapshots in time and then to compare the results. The difficulty is that the number of communities can change and many nodes can switch from a community to another. The recognition of the same community c at time t and $t + 1$ can be as hard as community identification itself. On top of that, when networks become larger, computation time increases exponentially, even with the fastest techniques. If one wants to follow the network evolution by comparing the evolution of communities at each time step, it implies to compute at each step a complete community detection. Such solution would become very expensive on a quickly evolving network. (Like for example Facebook, which has around half a million new users every day).

Some approaches, notably Palla [3], obtained interesting results with local methods. However this method stays too simple, and therefore can easily be fooled by tricky network configurations.

What we aim to do in this paper is to present a solution which apply the strengths of multi-agent systems – namely robustness, efficient distributed problem solving and adaptivity – to the field of community detection.

II. Description of the system

The multi-agent system we propose is composed of an evolving environment – the network – and a variable amount of agents.

A. Environment

The environment in which our agents live, evolve and die is described by the set of dynamical edges composing the network. Starting with empirical data on a specific dynamic network, we know the history of the environment and we simply aim at replaying it. In this respect it corresponds to a data-driven approach. More precisely it is described by an ordered set of time-stamped events that are either edge creation or edge removal. Initially, the environment is empty. Then, according to the ordered set of events, edges are added and removed until the whole history of the network has been played.

There are two kinds of agents evolving in this environment: the node agents and the community agents.

B. Node agents

Node agents are defined by their names (or labels) and the following characteristics:

- The list of other node agents they are linked with
- The list of communities they belong to and for each of them, the related value of $representativeness$ (detailed below).

The node agents have five actions:

- Create a new community
- Create a bond with another agent
- Remove a bond with another agent
- Ask to integrate a community
- Ask to ban a node from a community

1) Representativeness value: The value of $representativeness(i, c)$ of a node agent i to a community c is first computed when the agent node is added to a community. This value is defined as

$$\frac{nbNeighb(i, c)}{k_i}$$

where $nbNeighb(i, c)$ is the number of neighbors of agent i that also belong to c, and k_i represents the total number of neighbors of the agent i in the network, i.e. its connectivity.

The node agent then update this value each time that it creates or removes a bond with another agent.

2) Create a new community: Each time that a node agent creates a new bond with another agent, they decide whether or not to create a new community. This decision is taken by scanning their respective neighbors to find common ones. If a bunch of node agents detects that they now form a clique – all the agents are connected the one to all the others – of a minimal size, they create a community agent which initially includes all the node agents of the clique. The minimal clique required to generate a new community must be defined initially, it is the only parameter of our multi-agent system. It can be 3-clique if the graph is sparse, or 4-clique if the graph is denser.

3) Bond creation and removal: As explained beforehand, the aim of this paper is to replay the network creation in order to identify communities. Therefore, an ordered set of events available from empirical data will result in links creation or removal from the node agents following a data-driven approach. More precisely, if at time t, an edge is created between nodes $n1$ and $n2$, we order agent $n1$ to asks for the creation of a bond with $n2$ and $n2$ with $n1$. Therefore, $n1$ becomes a neighbor of $n2$ and vice versa.

Similarly, if we know that an edge is removed at a time t, we make the two concerned node agents remove their bond with the other.

4) Ask to integrate a community: As soon as an agent n creates a bond with another agent $n2$, he asks $n2$ the list of communities it belongs to. Then, for each community c to which n doesn't belongs to, he asks to integrate it. If his request is accepted by the corresponding community agent (described in next section), n joins the community c and adds it to his list of communities.

5) Ask to ban a node from a community: When agent n decides to remove a bond with $n2$, he checks if both of them belong to the same communities. If so, for each of these communities, n will asks to ban $n2$. The corresponding community agent will then decide whether or not to keep $n2$.

C. Community Agents

A community agent has the following attributes:

- A list of node agents that belong to this community
- A value of $seclusion$, which measures the quality of this community.

A community agent is also defined by the following possible behaviors:

- Decide for the integration of a node agent
- Decide for the banning of a node agent
- Decide whether or not to integrate another community
- Die

1) Value of seclusion: The value of $seclusion$ of the community is first computed when the community agent is created. This value measures the quality of the community, more precisely how well the community is separated from the remaining part of the system. This value is computed as

$$\sum_{n \in c} representativeness(n, c)$$

The smaller this value, the more the agents of the community have bonds with agents outside of it. However, a high value of $seclusion$ does not guarantee the quality of the community structure. Even if the nodes inside the community are scarcely connected between them, as soon as they have few bounds outside the community, its value of seclusion will be high. The quality of the community structure is guaranteed by the choices of the community agent when asked to add or remove a node.

2) Decide to integrate or not a node agent: When a community agent c receives a request from a node agent n asking for his integration, it first computes its potential belonging, $pb(c, n)$. This value is defined as

$$pb(c, n) = \sum_{n2 \in neighb(n)} representativeness(n2, c)$$

This value represents how strongly n is related to c. To evaluate if an agent should be integrated to it, the community agent compares $pb(c, n)$ to its value of $seclusion$. If n is linked to all agents in c, we have

$$pb(c, n) = seclusion(c)$$

The community integrates n if

$$pb(c, n) \geq \frac{seclusion(c)}{2}$$

The value of $pb(c, n)$ is, of course, related to the number of neighbors of the candidate agent. But it obviously depends also on which specific agents he is connected to. Indeed, in most of real networks, one observes a power law distribution of nodes' degrees, i.e. there are a lot of nodes with a small degree – few bonds – and few nodes – usually called hubs – with a lot of bonds. These hubs are good candidates to belong to several communities, but, according to our previous definition, with a small $representativeness$. Consequently, the choice of the community agent to integrate or not a node depends on the nodes he is connected to. If the agent n is only connected to hubs in c – therefore less representative of c, the community agent will not accept to integrate n. We consider that an agent belongs to a community when it is connected to its "core" agents, i.e. the ones who really represent it.

3) Decide to ban or not a node: This operation is very similar to the decision to integrate or not an agent. When a community agent c receives a request to ban an agent n, he computes the same test than for his integration. Therefore, if

$$pb(c, n) < \frac{seclusion(c)}{2}$$

n is banned from the community. As a consequence, c will also computes recursively the same test for all the neighbors of n. The rationale is that removing n from c is likely to reduce the potential belonging of its neighbors, which could be consequently banned from the community.

4) Decide to integrate or not another community: Each time a community agent c takes the decision to integrate or ban a node agent, the community could become more similar to another existing community. As the aim of community detection is to discriminate among possible communities, we integrated a mechanism that enables the merging of communities when they become too similar. In order to do so, the community agent c will asks to his agents the list of the communities they belong to. The community agent c then asks its integration to each one of these communities $c2$ when they are younger.

More precisely, c asks $c2$ the list of its members (agents), and computes the sum of the potential belonging of each of them. We named this sum the community likeness CL

$$CL(c, c2) = \sum_{n \in c2} pb(n, c)$$

If $CL(c, c2) \geq 0.75 * seclusion(c2)$, $c2$ dies and each of its agents ask for their integration in c. To summarize, if most of the agents from $c2$ could be integrated in c or are already part of it, $c2$ dies and most, if not all, of its nodes are integrated into c.

5) Death of communities: As a reminder, a community is created as soon as a clique of a minimal size s is detected, s being a parameter of the system. Apart of the preceding mechanism for the merging of communities that implies the death of some communities, when a community counts less than s agents, it is considered as non viable and therefore it dies.

III. Evaluation of the proposed algorithm

A. Computation time

As explained in introduction, the networks we are now able to obtain from Web 2.0 platforms and other digital resources are huge networks. Networks composed of millions of nodes and edges are now freely available, and their size is growing every day. Most of the existing community detection algorithms are not thought to deal with so large datasets, and their computation time grows exponentially with the size of the network. Even if a complexity is calculated for some of them, it stays theoretical because the computation time depends a lot on the density of the network considered, on the intrication of the existing communities, and so on and so forth.

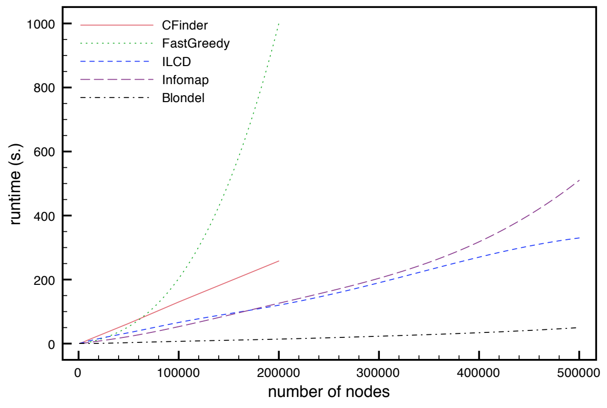

Figure 1. Comparison of the evolution of the speed with the size of the network for several well known community detection algorithms

On the contrary, with the proposed multi-agent system, all operations are made locally, which ensures that the complexity will not grow exponentially with the network size but rather linearly with the number of edges.

Practically, each addition of an edge $(n1, n2)$ to the network will generate the following actions:

- The node agents will ask for their integration to their neighbour's communities. But the only communities concerned are those which contains both $n1$ and $n2$, so very few compared to the exisiting ones on a large graph.
- The community agents decide whether or not they integrate the node agent. So, they only need to ask information to their members, again, a tiny portion of the agents from the total population.
- The modified communities (and only them) try to merge with other communities which have common nodes. Same remark as above.

The same analysis stands for edge removal.

Therefore it is clear that, as the network grows in size, the cost of each edge addition or removal will remain mostly similar, due to the local nature of the algorithm. The only thing that could slow down the computation time would be an important increase of the number of communities, with many agents belonging to several communities. In this case, the mechanism of communities merging ensures that similar communities will be merged, limiting this risk.

In figure 1, we compare some of the current fastest algorithms and our system. Tested algorithms are CFinder [3], FastGreedy [4], Infomap [5] and Blondel [6]. These tests were achieved on a recent personal computer with 4GB of memory. Implementations are either the ones given by their authors or the ones included in the iGraph open source library. They are implemented in C or C++. Only our algorithm and CFinder are implemented in JAVA. Our implantation can be downloaded on our website.

Benchmark graphs are built using a generator of graphs with community structure, the LFR Benchmark [7]. We generate graphs with the default settings (average degree = 15, max community size = 50), and make vary only the number of nodes. The number of edges changes linearly with the number of nodes.

1) FastGreedy: FastGreedy is one of the many algorithms that propose to optimize the modularity at the global level. We chose this method as it is presented as one of the fastest. To give an idea, the original method by Girvan and Newman [1] was not able to deal with graphs larger than a few thousands of nodes. We can see on the figure 1 that even with this really optimized method, the complexity grows exponentially considering the size of the graph, and therefore does not allow to study very large graphs.

2) CFinder: CFinder is the only well known method able to detect overlapping communities. Due to its local nature (the algorithm first detect all cliques of a given size and then merge all the ones that have only one different node), its runtime grows linearly with the size of the network. However, its main drawback is its memory usage. On our test, the algorithm exceeds our memory capacity before 300,000 nodes.

3) Blondel : Blondel is a method thought to be fast. We can see that it is way faster than all other methods. However, it is also a very naive method, with a greedy approach, which can be easily fooled by complex configurations. It tries to optimize the modularity by local assignation, but do not guarantee high values of modularity.

4) Infomap: Infomap is a more recent algorithm, proposing a new approach using random walks, and considered both as very quick and as efficient. We can see that its speed is quite comparable to our solution below 400,000 nodes, but from this limit, its slowly exponential evolution curve begins to penalize it.

5) iLCD: iLCD is the name given here to our system. As predicted, its complexity grows linearly with the network size, and seems to be rapid enough to study very large graphs.

B. Dynamic detection

Another characteristic of these large datasets now available, is their dynamic nature. By studying the logs of interactions on social networks, of exchange on media sharing platforms, of cell-phone calls, and so on and so forth, we have not only the information of who communicate with who, but also when and for how long. All existing methods to detect communities work only on static networks. When one wants to study the evolution of a network, one takes several snapshots of it, do static detection on them, and then try to compare the results [2]. However this method is strongly limited, as, on a quickly evolving network, it is nearly impossible to map the communities detected at time t with the ones detected at time $t + 1$. Furthermore, the detection made at time $t + 1$ does not take into account the results of the detection made at time t, therefore there is a major risk of inconsistency between the two results.

On the contrary, our multi-agent system tries to simulate the existence of communities at every step of the evolution of the network. Community agents are actually created at a given timestep, can live for a period of time, evolve by integrating or rejecting nodes, and then die. As we initially have the empirical data concerning the complete evolution of the network, the system produces as a readable output the complete evolution of the communities, as simulated by our system. (see [8] for more details)

One strength of the multi-agent systems often pointed out is their ability to adapt to real-time, changing problems. It is also the case here, as the system is not only able to detect communities on a given dataset, but could also be deployed to detect communities in real time on a large network at very small cost. As we shown in the previous part, the cost of one change of the environment (edge added or removed) is very small and stays the same as the network grows. Therefore, the system could be deployed on a very large social network like Facebook, and compute current communities in real time, updating them as soon as a new action is done on the network. This could leads to a lot of applications, from user assistance features (friend retrieval, friends organizer...) to business oriented services (targeted advertisement, new insights into the network's organization...)

C. Independent communities

As we explained in introduction, most of current community detection techniques are not able do detect communities with overlaps. Among the algorithms compared previously, which are the most considered currently, only one is able to detect overlapping communities (CFinder). The reason is that in most techniques, the network division is seen as a problem to solve at the global level. As a consequence, what will be detected as a community does not only depends on how well its members are linked, but also on how big is the whole graph, how dense it is, and whether or not the addition of a node in one community or another will improve or not the "quality of communities" at the global level.

On the contrary, in our multi-agent system, communities are autonomous entities that take or not the decision to integrate a node agent based only on their local view of the network and on their internal state. Therefore, if two communities consider that they should integrate the same node, both of them can do it and this particular node can belong to both communities. This ensures that each detected community really matches with reality, and is not an artifact caused by local variations of the global properties of the network.

The ability to detect overlapping communities is now widely recognized as an important feature when studying real world networks, especially social networks. To illustrate the importance of overlap detection, we generated with the LFR Benchmark (same generator as previously) some graphs with a structure as close as possible to the structure of a real social network, but with community structure that is known a priori. To find the parameters to give to the LFR Benchmark, we took inspiration from [9].In this paper, a large study has been done on several large Web 2.0 social networks, in order to know more about their properties. Especially, on several of them (YouTube, FlickR, LiveJournal and Orkut), there is a "group" structure, i.e. people can create groups and declare that they belong to one or several groups. Results differ from one social network to another, but the constants are:

- Most users belong to several groups
- Clustering is very high inside groups: nodes are linked to 50-90% of other nodes of the group.
- Group size remains quite small (depends strongly of the network, but in average around 20 - 50)
- Degree of nodes is quite high, with a power law distribution.

The LFR Benchmark does not allow to fit all these parameters. It also has some strong limitations, as for instance the equal repartition of edges: all nodes must have the same ratio of their edges inside and outside their communities, and must belong to the same number of communities, which is not realistic.

However, we chose as fixed parameters:

- The size of communities: between 20 and 30 nodes
- Each node must have 25% of its edges outside of all its communities (noise)
- The nodes degree distribution follows a power law
- Degree = 25 * number of communities to which it belongs to. It ensures a realistic number of edges with other nodes in the same community.

Moreover, we made vary only one parameter: the number of communities to which each node belongs to. Results are shown in figure 2.

The Y axis, which represents the quality of the detection, is a comparison between the results given by each algorithm and the correct decomposition in communities known by construction. To obtain this value, we used a version of the Normalized Mutual Information as described in [10]. A value of 1 means that the two sets of communities compared are exactly similar when 0 means they are totally different.

We compare our solution with CFinder, which is also able to deal with overlap, and with Infomap, which is considered as the most efficient algorithm but unable to deal with overlap. Unfortunately, CFinder was not able to run on all graphs. To run, it first needs to detect all cliques of a given size existing in the network. On dense graphs, this number becomes quickly enormous, and needs very large amounts of memory. However, the comparison with Infomap is enlightening: Infomap is the best algorithm when each node belongs to only one community, with a nearly perfect

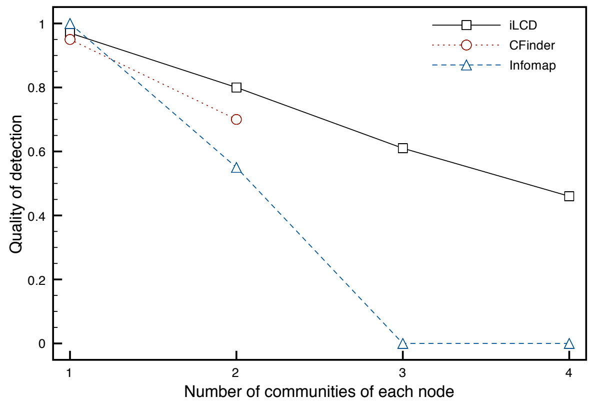

Figure 2. Comparison of the quality of the community detection made by several algorithms on generated graphs with an overlapping structure

detection. When each node belongs to two communities, the algorithm still manages to recognize communities, and just miss the "second belonging" of the nodes. However, when nodes belong to 3 communities or more, the algorithm seems unable to detect anything with success. It is not only that many nodes are misclassified: if we look more closely at the results given by the algorithm, we can see that nearly all nodes are in the same giant community.

Concerning our method, even with a very strong overlap, for instance with each node belonging to 4 different communities, our method is still able to do a reliable detection.

As a limit, we have to say that our algorithm is not as efficient with sparse communities. With the LFR benchmark, we can generate graphs with communities of, for example, 30 nodes, and a node's degree of 5. In this case, a node of a community will be linked to only a very small amount of other nodes of its community. Infomap and other "global" algorithms are still able to detect these sparse communities, because they are still "denser" than the remaining of the network. On the contrary, with what we have chosen as a definition of a community, the community must be recognizable in itself. However, this kind of sparse communities seems not really representative of what we found in real networks, and we should be able to detect them by searching for communities of communities.

IV. Conclusion

In this paper, we presented a multi-agent system aiming at simulating both the evolution of a network and the joint evolution of communities on it. By giving an existing network, the multi-agent system therefore performs efficient community detection along time.

We shown that this method is fast compared to existing community detection methods, allows to deal with overlapping communities and seems to give good results.

Its simulation nature also brings the new possibilities of dynamic and/or real-time detection on an evolving graph.

In a future work, we want to enhance this system on two sides:

- Playing with organization scales of our system by considering community agents as node agents and allowing them to act as such (by creating and removing edges with other community agents). By allowing them to make communities composed of communities, we could propose a solution for hierarchical community detection, which would give new insights into the network structure for very large graphs.
- By giving the possibility to node agents to create and remove bonds autonomously, after a learning period on empirical data for instance, we could generate realistic networks and try to predict the future evolution of a given network, while keeping a realistic community structure, which is another challenge in network science.

References

[1] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002. [Online]. Available: http://www.pnas.org/content/99/12/7821.abstract

[2] G. Palla, A. Barabasi, and T. Vicsek, "Quantifying social group evolution," *Nature*, vol. 446, no. 7136, pp. 664–667, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1038/nature05670

[3] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, Jun. 2005. [Online]. Available: http://dx.doi.org/10.1038/nature03607

[4] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, p. 066111, Dec 2004.

[5] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, Jan. 2008. [Online]. Available: http://www.pnas.org/content/105/4/1118.abstract

[6] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008. [Online]. Available: http://iopscience.iop.org/1742-5468/2008/10/P10008?ejredirect=migration

[7] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, no. 4, p. 046110, Oct. 2008. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.78.046110

[8] R. Cazabet, F. Amblard, and C. Hanachi, “Detection of overlapping communities in dynamical social networks,” in *IEEE International Conference on Social Computing*. IEEE, 2010, pp. 309–314.

[9] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC ’07. New York, NY, USA: ACM, 2007, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298311

[10] A. Lancichinetti and S. Fortunato, “Community detection algorithms: A comparative analysis,” *Physical Review E*, vol. 80, no. 5, p. 056117, Nov. 2009. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.80.056117