# 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 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, sensibility to noise, no detection of overlapping communities, exponential running time... This paper propose a multi-agent system replaying the evolution of a network and, in the same time, trying to mimic the birth, evolution and death of communities. After presenting the strengths and weaknesses of existing community detection algorithms, we describe our multi-agent system. Then, we compare our solution with existing works, and show some advantages of our method, in particular the possibility of doing dynamic detection of communities.**

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

## I. Introduction

Il the last few years, the democratization of the web 2.0 has produced a new kind of network data. With millions of people interacting by the means of digital media – Web 2.0 Social Networks like Facebook and Twitter, Media sharing website like Flickr or Youtube, message boards, ... – it is becoming easier and easier to extract very large datasets of interaction and communication among individuals. Therefore, much work has been done recently to study these networks. One of the more interesting and challenging problem is the one of community detection.

A community in a network is generally defined as a set of nodes strongly connected among them and more weakly to the remaining of the network. This definition stays quite imprecise, because in many networks we know that there are communities, we can recognize them, but there is no precise definition of them. Think for example of your social network, as it can be extracted from Facebook: you are probably "friend" with many people from your family, and most of these people likely are also connected to each other. But you maybe 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 perfect world, these groups of people would be totally separated and their identification would be very easy, a simple connected component identification. However in real life many people from one community are also friends with people from another.

Many algorithms exist to try to identify communities. The first efficient one was proposed by [1]. This method and most of the other ones proposed since then have a centralized, global approach. They 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 several metaheuristics and several variations of the metric.

However these methods all have two major drawbacks:

- They are not able to deal with overlap of communities
- They are not able to deal with dynamic networks

The first problem comes from at least two reasons. First, the computation time become much larger when allowing communities to overlap, because the range of possible decompositions growth by a tremendous amount. Secondly, as the approach is global, it is not the local quality of each community which is evaluated, but the decomposition as a whole, and it is hard this way to find a metric able to compare, for example, for a network of 100 nodes, a solution with 10 communities of 10 nodes and another one with 20 communities of 10 nodes.

The second problem, 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 compare the results, but this task is usually very hard because the number of communities can change and many nodes can switch from a community to another. Recognize the same community $c$ at time $t$ and $t+1$ can be as hard as community identification itself. On top of that, when network become large, computation time increase accordingly, even with the fastest techniques. If we want to follow the evolution of the network by doing regular community detection, it implies to compute every time a complete community detection, which will be very costly on a quickly evolving network as, for example, Facebook (around half a million new users every day).

Some approaches, notably [3] obtained interesting results with local methods. However these methods are only computational methods. Communities are not aware of the others, which can lead to similar, non-interesting communities. They

are also so simple that they can be easily fooled by tricky network configurations.

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

## II. Description of the system

Our multi-agent system 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 network, more specifically its edges. It must be seen as a kind of world for which we know the history, which we want to replay. This environment's history is described by a succession of edges created at specific time. Initially, the environment is empty. Then, at the first time, the first edge is added. Successively, all edges are added and then removed until the whole history of the network had been played.

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

### B. Node agents

Nodes agents are defined by their names and the following characteristics:

- The list of other agent nodes they have a bond with
- For each community to which they belongs, the value of $representativeness$ relative to them.

The node agents can have five types of actions:

- Try to create a new community
- Ask for the creation of a bond with another agent
- Remove a bond with another agent
- Ask to integrate a community
- Ask to reject a node from a community

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

$$\frac{nbNeighb(n, c)}{nbBonds(n)}$$

where $nbNeighb(n, c)$ is the number of neighbors of $n$ that also belongs to $c$ and $nbBonds(n)$ represents the total number of neighbors of $n$ in the network.

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

*2) Try to create a new community:* Each time a node create a new bond with another node agent, they decide or not to create a community. This decision is made by interrogating their respective neighbors to find common ones. If a bunch of nodes detect that they now form a clique – a bunch of nodes where all nodes are connected to all other nodes – of a minimal size, they automatically give birth to a community agent which initially include all nodes of the clique. The minimal size of the clique required to generate a new community must be defines initially, it is the only parameter of our multi-agent system. It can be 3 if the graph is sparse, or 4 if the graph is more dense.

*3) Bond creation and removal:* For the purpose of this paper, which is about detecting communities on a known network, node agents will be some kind of "zombies" or "puppets" agents, because we will tell them which of these two actions to do at which time. In a future work, we are planning to "give a will" to these agents, in order to create realistic networks or predict network evolution.

More precisely, if we know that, at time $t$, an edge is created between nodes $n1$ and $n2$, we order agent $n1$ to ask for the creation of a bond with $n2$ and $n2$ with $n1$. Therefore, the creation is automatically accepted and $n1$ 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 a node $n$ accepts a bond from another node $n2$, it asks $n2$ its communities and, for each community $c$ to which it doesn't belongs, it asks to integrate it. If this request is accepted, $n$ adds $c$ to the list of communities to which it belongs.

*5) Ask to reject a node from a community:* When $n$ decide to remove a bond with $n2$, it checks if both of them belongs to the same communities. If it is the case, for each of these communities, $n$ will ask to reject $n2$ from it. The community will then decide to keep or reject $n2$ from itself.

### C. Community Agents

A Community agent is defined by the following properties:

- A list of nodes. These are the components of this community
- A value of $seclusion$, which represents how this community is well defined.

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

- Decide to integrate or not a node to itself
- Decide to remove or not a node from itself
- Decide to integrate or not another community
- Die

*1) Value of seclusion:* The value of $seclusion$ of the community is first computed when the community is being born. This value represents how well this community is defined, or more precisely how well it is separated from the remaining of the system. This value is computed as

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

The smaller this value, the more the nodes inside the community have edges with nodes outside of it. However, a high value of $seclusion$ is not a guarantee of the quality of the community structure. Even if the nodes inside the community are scarcely connected between them, as soon as they have few bonds 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:* When a community $c$ receive a request from a node $n$ to belong to itself, it first compute 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 well how strongly $n$ is related to $c$. To evaluate if a node should be integrated to it, the community just compare $pb(c, n)$ to its own value of $seclusion$. If $n$ is linked to all nodes in $c$, we will have

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

The community integrate $n$ if

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

The value of $pb(c, n)$ is, of course, related to the number of nodes to which the candidate node is connected. However it also depends a lot on which specific nodes he is connected to. Indeed, in most of real networks, as predicted by the power law of nodes' degree, there are a lot of nodes with a small degree – few bonds – and some nodes – that we call hubs – with a lot of bonds. These hubs usually belongs to several communities, but, according to our previous definition, with a small $representativeness$. Consequently, the choice of the community to integrate or not a node will depend on the nodes to which it is connected. If the only nodes of $c$ to which $n$ is connected are hubs – therefore not representative of $c$, the community will not accept to integrate $n$. If a node really belongs to a community, it must be connected to its "core" nodes, the ones really representative of it.

*3) Decide to remove or not a node:* This operation is very similar to the choice of integration of a node. When the community $c$ receives a request to remove a node $n$, it compute the same test as for integration, and if now

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

$n$ is removed. However in this case, $c$ will also do recursively the same test on all its nodes that are neighbors of $n$. (Remove $n$ from $c$ is likely to reduce the potential belonging of neighbors of $n$ to $c$, which could allow $c$ to remove them from itself.)

*4) Decide to integrate or not another community:* Each time a community $c$ take the decision to add or remove a node, it can become more similar to another existing community. As we want to keep only communities that are different, and even not too alike, communities have a mechanism to merge with other communities if they become nearly equivalent. To do so, and to limit the number of communities to test, the community $c$ will asks to the nodes it includes the list of their other communities. For each of these communities $c2$ – which therefore have at least one node in common with $c$ – that are also younger than it, $c$ will try to integrate them.

To do so, $c$ will asks $c2$ the list of its nodes, and compute the sum for each of these nodes of their potential belonging. We call this sum the community's likeness,

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

If $CL(c, c2) \geq 0.75 * seclusion(c2)$, $c2$ die and all its nodes ask to be integrated in $c$. To sum it up, if most of the nodes of $c2$ could be integrated in $c$ or are already in it, $c2$ die while most of its nodes are integrated into $c$.

*5) Death of communities:* To be born, a community must be composed of a clique of a given size $s$, which is a parameter of the system. It is the minimal size under which a community is not considered as viable. As a consequence, if, after a node removal, a community counts less than $s$ intern nodes, it dies.

## III. Advantages of the multi-agent system over traditional methods

### A. Computation time

As explained in introduction, one thing that has changed with the new kind of networks we are now able to get by the means of Web 2.0 platforms and other digital resources is their size. Networks composed of millions of nodes and edges are now freely available, and their size will continue to grow. Most of the community detection algorithms were not thoughts to deal with so large datasets, and their running time grows exponentially with the size of the network. Even if a complexity is given for some of these algorithms, it stays theoretical because the actual computation time depends a lot of the density of the network, of the easiness to identify

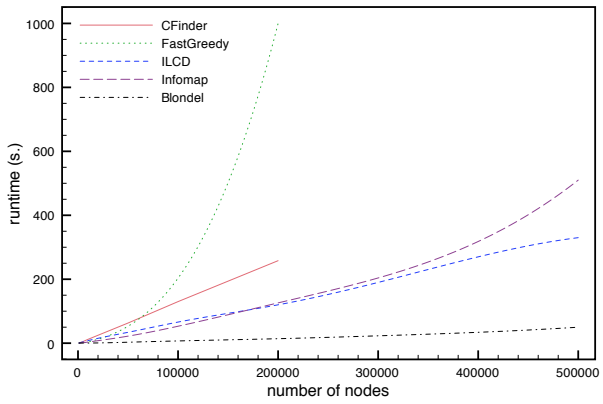


Figure 1. Comparison of the evolution of the speed with the size of the network for several well known community detection algorithms

communities (optimization by simulated annealing...), and so on and so forth.

On the contrary, with this multi-agent system, all operations are made at the local level, which can allow us to ensure that the complexity will not grow exponentially with the size of the network but more linearly with the number of edges.

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

- The nodes will ask to be integrated in the communities of their neighbor. Only the communities to which the two nodes belong are concerned, which is very few on a large graph.
- The communities compute if they should integrate the node. To do so, they only need to ask information to the nodes they are composed of. Again, a very tiny portion of the nodes are concerned on a large graph.
- The modified communities (and only them) try to merge with other communities with common nodes. Same remark as above.

The same analysis can be done for edge removal, with the same findings.

It is therefore clear that, as the network grows in size, the cost of each edge addition or removal will stay globally similar, due to the local nature of the algorithm. The only thing that can make the computation slower is an important increase of the number of communities, with many nodes belonging to several communities. However, the mechanism of community 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 done 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 were made using a generator of graphs with community structure, the LFR Benchmark [7]. We generated graphs with the default settings (average degree = 15. Max community size = 50), and made 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 proposing to optimize the value of 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 that even with this really optimized method, the complexity stays exponential, and does not allows 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 growth linearly with the size of the network. However, its 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 guarantees high values of modularity.

*4) Infomap:* Infomap is a recent algorithm, proposing a new approach using random walks, and considered as both very quick and efficient. We can see that its speed is quite comparable to the one of our method under 400,000 nodes, but then, its slowly exponential evolution curve began to penalize it.

*5) iLCD:* iLCD is the name given here to our system. As predicted, it has a complexity linear with the growth of the network, and seems to be quick 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 communicated with who, but also when and for how long. All existing methods to detect communities work only on static networks. When one want to study the evolution of a network, he 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's nearly impossible to recognize the communities detected at time $t$ in the detection made at time $t+1$. Furthermore, the detection made at time $t+1$ do not take into account the results of the detection made at time $t$, therefore there is no continuance between the two results.

On the contrary, our multi-agent system tries to simulate the existence of communities at each and every step of the evolution of the network. Community agents are actually being born at a time, can live for a period of time, evolve by integrating or rejecting nodes, and then die. As we initially had the complete evolution of the network, the system produce as a readable output the complete evolution of the communities, as simulated by our system.

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 overlap. Among the algorithms compared previously, which are the most considered currently, only one is able to detect overlapping communities (CFinder). This is because 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 do not depend only of how well its members are linked, but also of how big is the whole graph, how dense it is, and if setting 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 based only on their local view of the network and their internal state. Therefore, if two communities consider that they should integrate the same node, they just do it and this node now belongs to two communities. This ensures that each detected community really matches to a reality, and is not an artifact caused by a local variation 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 of the structure of a real social network, but with a known community structure. To find the parameters to give to the LFR Benchmark, we inspired ourselves of [8]. In this paper, a large study has been done on several large Web 2.0 social networks, in order to know more of their properties. Especially, on several of them (YouTube, FlickR, LiveJournal and Orkut), there's 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 do not allows fitting all these parameters. It also has some strong limits, as for example 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 must be between 20 and 30 nodes
- Each node must have 25% of its edges outside of all its communities (noise)
- Power law of node's degree distribution
- Degree = 25* number of communities to which it belongs. It ensures a realistic number of edges with other nodes of the same community.

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

The Y axe, 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 of the network. To obtain this value, we used a version of the Normalized Mutual Information as described in [9]. A value of 1 means that the 2 sets of communities compared are exactly similar when 0 means they are totally different.

We compare to our solution CFinder, which is also able to deal with overlap, and Infomap, which is considered as the most efficient of algorithms 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 present in the network. On dense graphs, this number become quickly enormous, and needs very large amounts of memory.

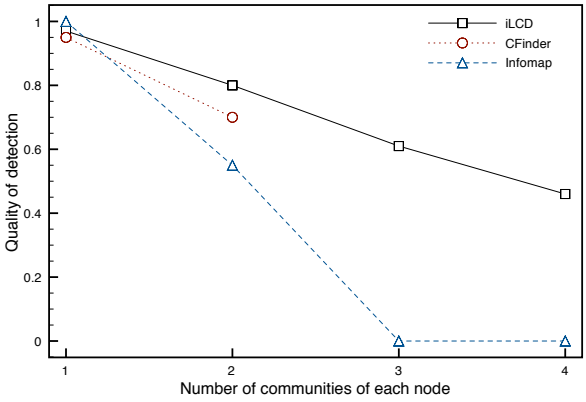


Figure 2. Comparison of the quality of the community detection made by several algorithms on generated graphs with an overlapping structure

However, the comparison with Infomap is enlightening: Infomap is the best algorithm when each node belongs only to one community, with a nearly perfect detection. When each node belongs to 2 communities, the algorithm still manages to recognize communities, and seems just to miss the "second belonging" of the nodes. But when nodes belong to 3 communities or more, the algorithm seems unable to detect anything with success. It's 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.

On the other side, even with a very strong overlap, 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 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 simulate 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 dealing 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:

- By considering communities themselves as nodes and allowing them to act as so (by creating and removing edges with other community agents), and allowing to make communities composed of communities, we could do hierarchical community detection, which gives new insights into the network structure for very large graphs.
- By giving the possibility to nodes to create and remove bonds, based for example on their previous acts, we could generate realistic networks or try to predict the future evolution of a given network, by keeping a 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] 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

[9] 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