Communities in evolving networks: definitions, detections and analysis techniques

Thomas Aynaud[a], Jean-Loup Guillaume[a], Qinna Wang[b], Eric Fleury[b]

[a]LIP6 – CNRS – *Université Pierre et Marie Curie*
4 place Jussieu
75005 Paris, France
firstname.lastname@lip6.fr
[b]*addresse de Qinna*

Abstract

Complex networks can often be divided in dense sub-networks called communities. These communities are crucial in understanding the underlying structure of these networks and may have applications in data mining or visualisation for instance. We expose here a survey of recent advances in the definition, the detection and the analysis of these communities in the particular case of evolving networks.

Keywords: Community, evolving network, complex network, tracking, multi-slice

1. Introduction

Real world complex systems are very often composed of several interacting objects. These interactions can be modeled by a network whose nodes are the objects themselves, linked together if they do interact. For example, the web is a network of web pages connected through hyperlinks and the brain is a network of neurons connected through synapses. It has been shown that, even if these networks represent drastically different objects, they share similar properties. For instance, the distances are often low and they behave locally almost like cliques.

Understanding the underlying structure of these interaction networks may have a major impact in the understanding of the systems they model and one classic way to describe this structure is to consider it as composed of several *communities*. A community is intuitively a group of nodes *sharing something* like a group of friends in a social network or a group of web pages dealing with the same subject for instance. Formalising the definition

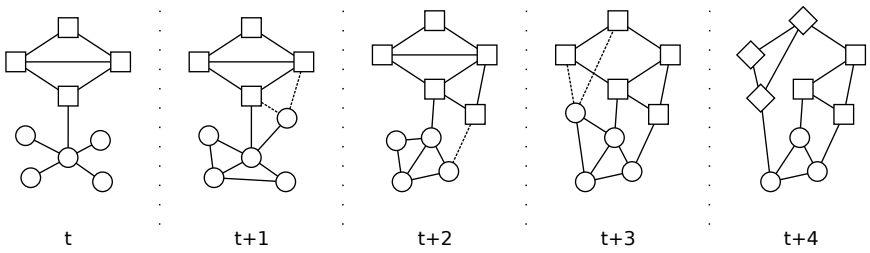

Figure 1: Snapshot graphs

of communities and finding them efficiently is a difficult task and many algorithms have been proposed in the last decade. Communities are often defined using the network topology as groups of nodes with many links inside the groups but a few links between them. Then, finding communities is similar at looking for a partition of the nodes that maximises a *quality function* which captures this idea of dense groups. One such quality function widely used is the modularity[1]. Other definitions have been proposed like the Clique Percolation Model or the various statistical models. Readers can refer to the surveys [2, 3, 4] for more details. Nowadays, although several definitions still exist, detecting communities in networks can be done really efficiently and algorithms produce very interesting results. However, most studies focus on static networks and the usual approach is to collect data during a long period and then to aggregate all these date to create a large static network. By doing this, many information are lost since real data are always evolving: pages appear or are updated constantly on the web, people start new relationships, etc. Thus, recent researchs have been conducted to investigate the case of communities in evolving networks.

An evolving network is often defined as a sequence of static networks, each of them representing the state of the network at different timestamps (see figure 1). Since each snapshot is a static graph, the first approach to compute communities on an evolving network is to use a classical algorithm on each snapshot more or less independently. The approaches in this direction are described more precisely in Section 2. Using static algorithms in evolving networks triggers some complex issues that will also be described in Section 2. To circumvent these issues, some algorithms directly suited for dynamic networks have been proposed. They are described in Section 3. Another related issue concerns incremental algorithms: instead of studying a given dynamic network, such approches consider that data arrive as a continuous stream. One typical case is a search engine that wants to adapt

its results everytime new pages are discovered but does not want to recompute the whole community decomposition. These specific algorithms are explained in Section 4. Finally, we present in Section 5 some results which have been obtained with analysis techniques of evolving communities.

2. Using static algorithms on several snapshots

As the problem of community detection is well known on static graphs, it is natural to rely on already studied solutions for the evolving case. Therefore, many attempts have been made to use static algorithms on dynamic networks: for each timestep, a partition of the nodes is computed and the main issue is then to characterise the evolution of the communities: what happened to a community between the two timesteps. Indeed, a community may merge with another one, split or disappear for example. When we try to identity some common parts between two partitions, we are trying to solve a *matching problem*. In the following, we will also use the term of tracking when we are trying to characterize the evolution of given community. Figure 2 presents such a matching.

One underlying issue is related to the stability of static algorithms. Indeed they are often non deterministic and few modifications, or even no modifications, of the input network may lead to many changes in the resulting community partition. This is validated in [5] where the authors compare three classical community detection algorithms on different evolving network and show that they are not stable at all. Partitions between two consecutive timesteps are compared using a partition edit distance that counts the number of nodes that must to be moved to match one partition to the other (more details in Section 5.1). There is regularly more than 10% of the nodes that change of community even after a very small modification of the network (one node removed). Thus, the communities vary more than the network and performing a matching between the partitions of two consecutive time stamps is very hard since we cannot distinguish between the modifications due to the evolution and the ones due to the instability of the algorithm itself. A stabilisation of the Louvain algorithm for modularity optimisation is proposed which achieves high decomposition quality with very good stability.

2.1. Partitions matching using set theory

The first intuitive idea to perform the matching between partitions is to use set theory and rules or methods to decide whether to sets of different partitions are similar or not. For example, if two communities of successive

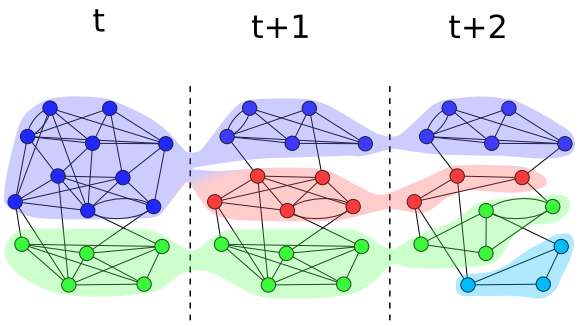

Figure 2: Three snapshot of a given evolving graph with a fixed matching between the communities of different time steps.

snapshots share many nodes, they are related. The main problem is that given two partitions, one can find many different valid matchings between these partitions. This is illustrated in Figure 3. Therefore the matching problem can be rephrased as the maximization of a quality function that tells whether or not a matching is good.

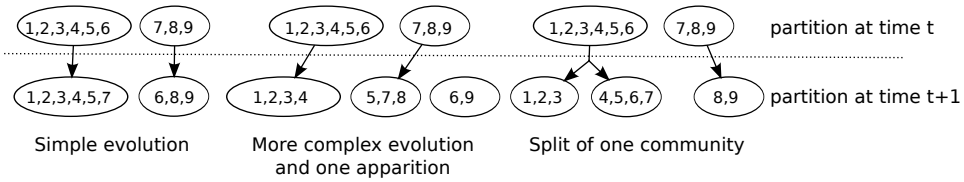

Figure 3: The matching problem

The first attempt to solve the matching problem in the context of evolving communities is [6]. The authors study a co-authoring network using the size of the intersection of communities: two communities at successive time steps are matched if they share enough nodes, i.e if the size of their intersection is high. More precisely, the *match value* between two communities is defined as:

$$match(C, C') = min\left(\frac{|C \cap C'|}{|C|}, \frac{|C \cap C'|}{|C'|}\right)$$

This value ranges between 0 and 1. The bigger the intersection is, i.e. the closer the two communities are, the higher it is: a value of 1 means that they are the same and 0 that they are disjoint. The authors derive from this

the match of a community C at time t as the community C' at time $t+1$ which maximises this value and they use a classical hierarchical clustering to study the *CiteSeer* database. Due to the instability of the community detection algorithm used, many communities disappear or are strongly modified between consecutive timesteps. Thus, communities are initially impossible to follow since the changes are caused by the algorithm and not by true changes in the dataset. This issue is solved by considering only the stable communities which are defined as communities which still exist when the input network is slightly modified. Such communities are called *natural communities* and while the process eliminates many communities, it allows a first study of the few surviving ones. The algorithm is validated by observing the evolution of communities on the *CiteSeer* database of some already known phenomena in science like the apparition of the add-hoc networks research community.

This matching methodology has been generalised in [7] where the authors define many similar rules to deal with other cases of communities evolution: merge, split, appearance and disappearance (see Figure 3). Given a community C at time t, the authors define $match(C)$ as the community at time $t+1$ whose intersection with C is the largest if and only if this intersection is larger than a given threshold. If there no community at time $t+1$ whose intersection is larger than the threshold then $match(C) = \varnothing$. Then they use a set of rules to track communities:

- $C \in P_t$ becomes $C' \in P_{t+1}$ if $C' = match(C)$ and $\forall C'' \in P_t \neq C, C' \neq match(C'')$.
- $C \in P_t$ splits into multiple communities $C_1, C_2, ..., C_k \in P_{t+1}$ if $\forall i$, $C_i \cap C$ is large enough and $(C_1 \cup C_2 \cup ... \cup C_k) \cap C$ is large enough.
- $C \in P_t$ has merged with others communities into $C' \in P_{t+1}$ if $C' = match(C)$ and $\exists Z \in P_t \neq C, C' = match(Z)$.
- $C \in P_t$ disappears if none of the cases above hold.
- $C' \in P_{t+1}$ appears if $\forall C \in P_t, C' \neq match(C)$.

These rules are quite natural and are easily understandable, but are not very satisfactory. Indeed, the term "large enough" is not a formal definition, the threshold is hard to choose, there exists many normalisations of the intersections size: for instance we could decide that two communities are similar when 70% of one is inside the other or when they share 80% of the

nodes. This framework, is called MONIC and many variations of the set of rules have been proposed in [8, 9, 10, 11, 12]. Anyway, finding a consensus on a minimal set of rules seems impossible and the various parameters are generally not easy to choose.

In [13], the authors use the same kind of solutions but instead of computing the *match* between communities they propose to follow communities using few important nodes that they called *ore nodes*. The idea is that core nodes should be more stable than border nodes that might belong to several overlapping communities and which create noise when then move. They thus select for each community some representative core nodes nodes and track them between different timesteps. The limitations of the community tracking still hold and furthermore we have to define core nodes. In [13] the core nodes are nodes v which verify $\sum_{n \in neighbours} degree(v) - degree(n) > 0$ whereas in [14] they are nodes whose degree is higher than a given k. There are many centrality measures which all want to classify nodes relatively to their importance and chosing among those is difficult and may depends on targeted application.

2.2. Using the community detection algorithm to perform the matching

Instead of computing the matching after having computed the communities, it is possible in some contexts to perform the matching during the community detection. This is done in [15], where the authors use a specific community definition and an associated algorithm on the union of the networks at time t and at time $t + 1$. Their definition is such that each community of the union network contains both communities of the network at time t and at time $t + 1$. Therefore, communities at time $t + 1$ that are grouped with communities at time t on the union graph are considered to be evolution of them (see figure 4). The authors validate this approach to study a mobile phone network and a co-authorship network.

However, this specific technique relies on the definition of communities and a strong hypothesis must be fulfilled by the community detection algorithm: if links are added to the network, then the communities can only grow, merge or remain unchanged. This property ensures that detected communities on the union network contain full communities of both timestamps t and $t+1$ and not just fraction of communities which would make the matching impossible. This property is barely fulfilled and suppose a very constrained definition of what a community is (overlapping cliques in [15]). This technique also requires parameters and rules to deal with the cases where several communities of different snapshots are grouped (merge and split).

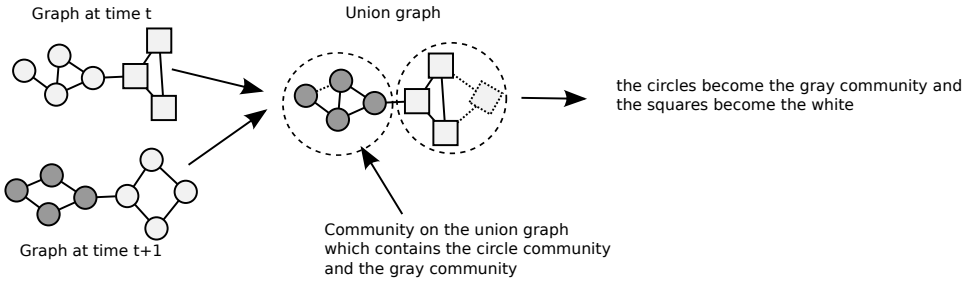

Figure 4: Using the union graph to compute evolution

2.3. Community evolution network

Another way to use a community detection algorithm to perform the matching is to build a temporal network representing the relations between communities at each time step. The communities are then computed on this network and it therefore gives communities that span across several snapshots. Thus, such algorithm are decomposed in two phases: first, each snapshot is decomposed in static communities and then these static communities are joined in temporal communities that span over several time steps. The static communities for each snapshot are sometimes called community instances to distinguish them from community (see Figure 5).

In [16], *community instances* are computed with a classical divisive hierarchical clustering [1] at each timestep. Then, the authors build an *evolution graph* where nodes are the community instances and links join community instances of different timesteps with a weight based on the MONIC matching and a threshold based on the time distance. Finally, this evolution graph between the communities instances is also decomposed with the same algorithm and the decomposition found on this graph define the final communities. Since they contain community instances of several timestep, these communities span over given period and a given node of the initial graph can belong to different communities at each timestep. Thus, the authors propose a metric to decide the *involvement* of a node to a community on the evolution graph.

In [17] the authors use the same kind of two steps decomposition. They first compute community instances which are then packed in a *tensor*. Instances are then weighted to indicate how important they are to the considered final community but also to take into account the importance of a community at a given timestep. Finally, they obtain a function to optimise to obtain fuzzy communities that span across several timesteps as they

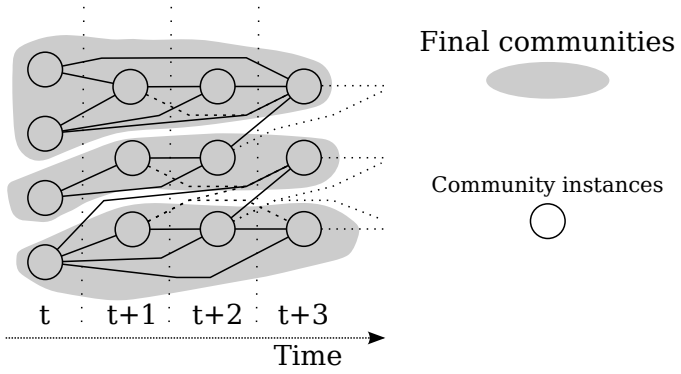

Figure 5: An example of evolution graph with community instances (nodes) and temporal communities (in grey).

are the combination of community instances of several timesteps. They finally propose an process based on constrained optimization to optimize the function and validate with the classical NEC-blog dataset.

In [18], the authors use a similar framework but redefine communities using several assumptions: it should be a group of people seeing each other often, seeing other people rarely and that does not change a lot during its lifetime. Moreover, member of a community tend not to change of community frequently. With given community instances (or groups), the authors propose a quality function to assign nodes and instance communities to communities. Morevoer, nodes may change of community during the time and this association is penalised with several costs that are summed:

- α is the *individual cost* when a node changes of community between two successive timestamps. This reflects that one node must not change too often.
- β is the *group cost* when a node and its group are not in the same communities at a given time.

Finally, they add a colour cost which is proportional, for each node, to the number of communities it belongs to during the whole time so as to ensure that one particular node does not belong to many communities. Optimising this function is NP-Complete, but algorithms with proved approximation factors are proposed in [19].

2.4. Conclusion

None of the above mentionned techniques seems perfect. The two main problems are first the instability of the algorithms and when problem is solved or neglected, classical technique based on the size of the intersection of communities needs many rules and parameters which are difficult to choose. As approaches using static algorithms are limited, many different algorithms have proposed to use directly the temporal information during the community detection and not afterwards.

3. Using temporal information directly to find better communities

3.1. Modification of the quality function

Since classical community detection are based on static quality functions, modularity for instance, a solution can be to modify such quality functions to integrate evolution.

The first attempt was presented in [20] which splits the quality function in two terms: a part for the quality of the current snapshot and a part to ensure stability:

$$Q = Q_{snapshot} + \alpha Q_{stability},$$

where $Q_{snapshot}$ is a static quality function, $Q_{stability}$ is a term which evaluates the distance of the new partition with the precedent and α is a parameter that characterises the importance given to the stability. This new quality function allows to obtain a series of more interesting clusterings and to reduce the number of artifacts caused by the optimisation algorithm. The authors then use this new quality function to extend some algorithms like *k-means* and hierarchical agglomerative algorithm.

This idea has been extended in [21] in which the authors propose to use an overall quality term instead of a stability term. In this case the partition found at time t does not have to be close to the partition at time $t-1$ but have to be a good partition at time t and a fairly good partition at time $t-1$.

Rather than just considering two consecutive time steps, an extension of this stabilization, presented in [22] proposes to find only one partition that is always good. To do this, the average quality over time is optimised. If the quality function is the modularity and if the network does not change too much, it is possible to achieve on average a quality which is very close to the optimal one. As there is only one partition, there is no stability issue. But this does not means that there is no evolution as the nodes may not exist at all timestamps. In this cas, the evolution is more inside the

communities than between them, giving different insight than traditional community tracking.

The stability problem has also been considered in [23]. Instead of modifying the quality function to integrate smoothness, the authors compute communities at time $t-1$ and if two nodes are in the same community at time $t-1$ and are also connected at t, then the weight of the link between them is increased or decreased of a given factor α to increase the smoothness of a modularity optimisation algorithm. If the algorithm considers the weight as a similarity factor, then the weight is increased, while if the weight is a distance, it is decreased.

3.2. Computing directly with models

Another advance has been done in [24, 25] in which the authors formulate the quality function as a non negative matrix factorisation that jointly optimise the quality and the stability of the communities. Then, they propose algorithms to optimise this function. They test their framework, called FacetNet, using an extension of the modularity suited for overlapping communities and use it to study several dataset such as the dblp dataset, the NEC-blog dataset and some synthetic ones. Their framework is one of the most general since it handles both overlapping and evolving communities. It is also one of the few that do not separate the detection of the communities and the detection of the evolution (i.e. the matching problem).

In [26], the authors study a dynamic modification of the stochastic bloc model to generate graphs and obtain insight on the structure. The stochastic block model generates a network by adding node one by one. When a node is added, it is assigned to a community following a given probability π and then links are added following a probability that depends on the community of the two extremities. They change the community assignment to depend on the previous assignment and finally evaluate the most likely value for the various parameters. Those parameters represent the community assignment and give insight in the dynamic. They propose ways to estimate the parameters and show that the results are good by computing the modularity at each timestep and comparing results with ground truth via the mutual information.

3.3. Temporal sliced graphs

The temporal information can also be coded in the graph itself. Given several snapshots of an evolving network, they may be placed side by side in one temporal graph. So, nodes that exist at several timestamps appear several times in this graph. Each snapshot is a slice of the dynamic network.

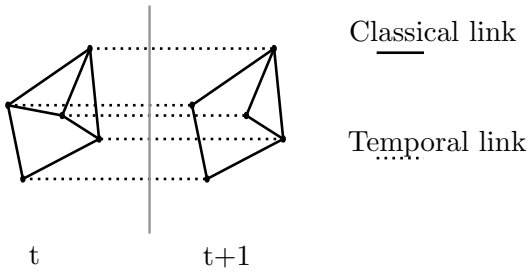

Figure 6: Example of a temporal graph: the same network is shown side by side at two timestamps and communities are computed on this union graph.

Then *temporal links* between nodes in different slices can be added, typically between a node at time t and the same node at time $t+1$. The result is one network with two kind of links: the links that really exist at some moment in the evolution of the network and the links between slices (see Figure 6).

The first attempt is [27] where the authors build a temporal sliced graph and then use the classical community detection algorithm Walktrap [28] on it. The communities on this graph contain thus nodes of various timestamps and are consequently *communities over time*. The matching problem is automatically solved since communities span over timestamps.

This idea of placing several snapshots side by side is extended in [29] where they propose a more general way to connect slices. They also propose a modification of the modularity integrating directly the slices and use it to detect multi-scale communities and dynamic communities.

3.4. Conclusion

Several ideas have been proposed to study dynamic communities using snapshots or more complicated ideas. One major lack is now validation tools. Indeed there exist almost consensual quality functions for static network that people can use to evaluate their algorithms and some tests networks that are randomly generated, see [30] for instance or real dataset with known ground truth, see [31] for instance. However, such general quality functions or validation tools do not exist for evolving networks. One new random model based on the static one of [30] and one based on [1] are proposed respectively in [12] and in [23] but they are not yet widely used.

4. Incremental/Online algorithms

Incremental or online algorithms aim to process data stream instead of a full snapshot. The input is thus a sequence of events on the network and the algorithm tries to maintain a fairly good community decomposition by updating its current decomposition instead of computing one from scratch. This is useful in case of real time monitoring of huge dataset such as the Internet or the web for instance. Furthermore if updates are computed efficiently, then incremental algorithms can achieve better performances than computing communities on each snapshot independently.

In [32], the authors propose an incremental algorithm to compute communities using eigenvalues of the network. By just considering the changes, they are able to compute communities ten time faster on the new snapshot than by considering the whole network with a small quality cost. They hence modify the spectral clustering algorithm to incrementally find the eigenvalues and eigenvectors.

A similar kind of algorithm is proposed in [33]. They first define a distance between nodes of the network and then the neighbourhood of a node as the topological balls of radius ϵ. They consider only the nodes whose neighbourhood size is bigger than a given threshold considered as core vertex. Nodes who belong to the neighbourhood of a core vertex are border vertex, and others are noise vertex. The communities are then the transitive union of neighbourhoods that share nodes. This defines a static community detection algorithm similar to the classic DBSCAN clustering algorithm and they propose techniques to update the neighbourhoods and the community when nodes or links are added or removed.

Two modularity optimisation, the Louvain Algorithm and the Fast Greedy algorithm, are modified in [34] to handle small modifications. Both algorithms are heuristics that greedily change nodes of community and merge communities to optimise the modularity. Considering that they already have a partition, the authors apply similar heuristics but they fix nodes not affected by the network change in their previous community. Then, only a few nodes, depending on the event, are movable, and the algorithm is therefore must faster than computing on the whole network.

Finally, another incremental algorithm is proposed in [35]. Instead of reacting to events, the authors encode the input network in a network between communities: each community corresponds to two nodes, linked with a weight of the sum of the intra links weight. There is a link between two communities whose weight is the number of links connecting them. This

results in a much smaller network than the original network. Then, when a change occurs, the authors detect which communities will be modified and split them. They can then use a classic algorithm on this network and, since the network is really smaller, they can achieve very good performances if there are few modifications. This method is also very general and can be applied to many algorithms since only the input network is modified.

5. Analysis of the evolution

Analysing the communities in evolving networks raises new issues. Indeed, new properties directly related to the evolution emerge, like the life expectancy or the growth speed of communities for instance. In general there is no real consensus on the interesting metrics and many of the definitions are quite similar, with only some modifications of the normalisation process for example.

5.1. Stability and event detection

Evaluating the distance between two clustering is already a well known problem and it has been deeply analyzed outside of the context of evolving communities. Many metrics such as the Rand index, the mutual information, and many more have been defined and compared. We redirect the reader to [] for more details and we will only focus on some results related to the stability for evolving communities.

5.1.1. Characterising changes

Measuring the stability is important but another issue consists in measuring how fast the events occurs and how do the communities change. In [36] the authors study a series of four snapshots of the web crawled from 1999 to 2002 and define some parameters at a given timestep to describe how quickly a community evolves, see table 1.

Similar values are defined in [7] to study the dynamic of the communities tracked with the MONIC framework. The authors propose the survival ratio (the proportion of communities at time t that survive at time $t + 1$), the absorption ratio (the proportion of communities that are absorbed) and the pass-forward ratio (the sum of both the survival and absorption ratio) to analyse the communities inside the ACM library.

The modifications within a community are related to its *popularity* defined in [9] as the number of nodes joining the community minus the number of nodes leaving it. In the dblp dataset, it appears that the XML community

Name	Definition	Representation
growth rate	$\frac{\lvert c' \rvert - \lvert c \rvert}{\delta_t}$	evolution speed of the size of the community
novelty	$\frac{\lvert c' - c \rvert}{\delta_t}$	number of new nodes in the community
disappearance rate	$\frac{\lvert c' - c \rvert}{\delta_t}$	number of disappearing nodes in the community
merge rate	$\frac{\lvert c' \cap C - c \rvert}{\delta_t}$	number of absorbed nodes from other communities by merging
split rate	$\frac{\lvert c \cap C' - c' \rvert}{\delta_t}$	number of splits from c

Table 1: Properties proposed in [36] with c a community at time t, c' the corresponding community at time $t + \delta_t$, C the union of all the communities at time t and C' the union of all the communities at time $t + \delta_t$.

starts being very popular in 2000 according to this metric which corresponds to a known historical event and tend to prove the interest of the measure.

5.1.2. Detecting events and major changes

The detection of events is another stability related issue. The objective here is to be able to distinguish between the normal transformations and the abnormal ones.

In [37], the authors compute communities on several snapshots and follow the minimum description length principle: any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally. A community is then a grouping of nodes which allows a good compression of the graph: it minimises the description length. They build a list of consecutive description of communities. If a community decomposition is close to the previous ones, then it can be effectively compressed with them and the compressed size does not change a lot. Conversely, if the new decomposition vary a lot from the previous, adding it to the list will not be a lot more effective than compressing it separately. The authors therefore use this principle to detect changes in the community structure rather than describing it.

5.2. Node evolution in communities

An orthogonal issue to the evaluation of the communities evolution consists in studying how nodes move from community to community. To study this evolution, [9] defines some metrics as the *sociability* or the *influence* of a node. The sociability is defined as the number of communities it belongs to during its whole life and the authors use this metric to make link prediction

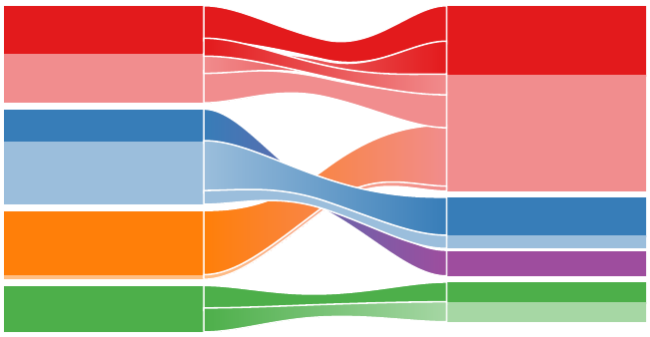

Figure 7: Example of mapping between communities (Extrait de leur papier pour le moment, il faut en refaire une)

within the dblp dataset and some clinical test results and it appears to be very efficient.

They also define the influence of a node, which should characterise if a node make other nodes join or leave a community when it does. Thus, let's n be a node, *companions* be the set of nodes that join or leave a community when n does and m the number of join and leave of a community that n does. Then, the influence of n is : $influence(n) = \frac{|companions|}{m}$. This makes followers of influential nodes being influential, so they add a few rules to curb this effect: let's n' be the node which has the more interaction with n, if the degree of n' is higher than the degree of n or the influence of n' is higher than the influence of n, then the influence of n is set to 0. It excludes followers, but only the high influence nodes have an interesting value, and it disallows to influent node to interact a lot together.

Finally, [38] proposes a visualisation tool to study community evolution. Their *alluvial diagrams* highlights structural changes between two consecutive timesteps by representing communities as blocks and drawing flows between them (see figure 7).

5.3. Real case studies

Finally, a few real case studies have been done which propose insights and explanations on various community behaviours.

First, [39] analyses the general characteristics of a blog network from 1999 to 2002 and study the behaviour of intra community links. They use the notion of burstiness defined by Kleinberg in [40]. The intra community links tend to appear more and more as burst (a large amount of links quickly

added together) and there are more and more communities which also tend to grow. This result is historical and not general as their dataset corresponds to the apparition of blog-space but it is a valuable insight about the blog emergence.

In [41], the authors study two datasets. The first one is extracted from the science repository *dblp* and the second from the news and communication website *livejournal*. They do not detect communities on them, but use ground knowledge to build a kind of clustering. In the dblp dataset, authors are grouped if they attend the same conferences and in livejournal, users can themselves join discussion groups. Authors are wondering if there are some structural reasons that lead a node to join a group and why some groups are growing. To explain this, they define many features of a node and they build a decision tree according to these features. In livejournal, the most discriminating feature is the proportion of friends in community who are friends with each others. They build similar trees to predict community growth. They also wonder whether people bring subjects inside groups or if the subjects appear in groups and people join such groups afterwards to discuss the subjects.

Finally, [15] studies the life expectancy of communities, depending on their size and stability. The life expectancy is the time between the appearance of a community and the time when it disappears. They use their clique percolation method to detect communities and their evolution on a network of co-authorship and on a network of phone call and discover that there are two cases, depending on the size of the communities. They show that small communities need an high stability (the ξ parameter defined at section **??**) to survive and that big communities need a smaller stability to survive. In other terms, the small ones have to be very stable if they want to survive longer, admitting very few changes, whereas big communities must adapt themselves to survive and cannot rely on a very strong core of members.

References

[1] M. E. J. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical Review E 69 (2004) 26113.

[2] M. A. Porter, P. J. Mucha, J.-p. Onnela, Communities in Networks, World Wide Web Internet And Web Information Systems (????) 0–26.

[3] S. E. Schaeffer, Graph clustering, Computer Science Review 1 (2007) 27–64.

[4] S. Fortunato, Community detection in graphs, Physics Reports (2009).

[5] T. Aynaud, J.-L. Guillaume, Static community detection algorithms for evolving networks, in: Wireless Networks, volume 2010, pp. 508–514.

[6] J. Hopcroft, O. Khan, B. Kulis, B. Selman, Tracking evolving communities in large linked networks, in: National Academy of Sciences of the United States of America, volume 101, National Acad Sciences, 2004, p. 5249.

[7] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, R. Schult, Monic: modeling and monitoring cluster transitions, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA, 2006, pp. 706–711.

[8] T. Falkowski, M. Spiliopoulou, J. Bartelheimer, Community dynamics mining, in: Proceedings of 14th European Conference on Information Systems (ECIS 2006), Citeseer, Goteborg, 2006.

[9] S. Asur, S. Parthasarathy, D. Ucar, An event-based framework for characterizing the evolutionary behavior of interaction graphs, in: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2007, p. 921.

[10] M. Oliveira, J. Gama, Bipartite graphs for monitoring clusters transitions, Advances in Intelligent Data Analysis IX M (2010) 114–124.

[11] M. Oliveira, J. Gama, Understanding Clusters Evolution, in: Workshop on Ubiquitous Data Mining, volume D, pp. 16 – 20.

[12] D. Greene, D. Doyle, Tracking the evolution of communities in dynamic social networks, in: Advances in Social Networks Analysis and Mining (ASONAM), volume 2010, IEEE, 2010, pp. 1–13.

[13] Y. Wang, B. Wu, N. Du, Community Evolution of Social Network: Feature, Algorithm and Model, Science And Technology (2008).

[14] M. Beiró, J. Busch, Visualizing communities in dynamic networks, in: Latin American Workshop on Dynamic Networks, volume 1.

[15] G. Palla, A.-L. Barabasi, T. Vicsek, Quantifying social group evolution, Nature 446 (2007) 664–667.

[16] T. Falkowski, M. Spiliopoulou, Users in volatile communities: Studying active participation and community evolution, Lecture Notes in Computer Science 4511 (2007) 47.

[17] Y. Chi, S. Zhu, X. Song, J. Tatemura, B. L. Tseng, Structural and temporal analysis of the blogosphere through community factorization, Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07 (2007) 163.

[18] C. Tantipathananandh, T. Berger-Wolf, D. Kempe, A framework for community identification in dynamic social networks, Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07 (2007) 717.

[19] C. Tantipathananandh, T. Berger-Wolf, Constant-factor approximation algorithms for identifying dynamic communities, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, volume Id, ACM, 2009, pp. 827–836.

[20] R. Kumar, A. Tomkins, D. Chakrabarti, Evolutionary clustering, in: In Proc. of the 12th ACM SIGKDD Conference.

[21] X. Song, Y. Chi, B. L. Tseng, D. Zhou, K. Hino, Evolutionary spectral clustering by incorporating temporal smoothness, in: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA, 2007, pp. 153–162.

[22] T. Aynaud, J. Guillaume, Long range community detection, in: Latin American Workshop on Dynamic Networks.

[23] M. Kim, J. Han, A particle-and-density based evolutionary clustering method for dynamic networks, Proceedings of the VLDB Endowment 2 (2009) 622–633.

[24] B. L. Tseng, Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, Facetnet: a framework for analyzing communities and their evolutions in dynamic networks, Social Networks (2008) 685–694.

[25] B. L. Tseng, Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, Analyzing communities and their evolutions in dynamic social networks, ACM Transactions on Knowledge Discovery from Data 3 (2009) 1–31.

[26] T. Yang, Y. Chi, S. Zhu, Y. Gong, R. Jin, A Bayesian Approach Toward Finding Communities and Their Evolutions in Dynamics Social Networks, ? (????) 990–1001.

[27] M. B. Jdidia, C. Robardet, E. Fleury, Communities detection and analysis of their dynamics in collaborative networks., in: ICDIM, IEEE, 2007, pp. 744–749.

[28] P. Pons, M. Latapy, Computing communities in large networks using random walks, Journal of Graph Algorithms and Applications 10 (2006) 191–218.

[29] P. Mucha, T. Richardson, K. Macon, M. Porter, Community structure in time-dependent, multiscale, and multiplex networks, science 876 (2010) 10–13.

[30] A. Lancichinetti, S. Fortunato, F. Radicchi, Benchmark graphs for testing community detection algorithms, Physical Review E 93046110 (2008).

[31] W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33 (1977) 452–473.

[32] H. Ning, W. Xu, Y. Chi, Y. Gong, T. Huang, Incremental spectral clustering with application to monitoring of evolving blog communities, In SIAM Int. Conf. on Data Mining (2007).

[33] T. Falkowski, A. Barth, M. Spiliopoulou, Studying community dynamics with an incremental graph mining algorithm, in: Proc. of the 14 th Americas Conference on Information Systems (AMCIS 2008), pp. 1–11.

[34] R. Görke, P. Maillard, C. Staudt, Modularity-Driven Clustering of Dynamic Graphs, Experimental Algorithms Cl (2010).

[35] T. Dinh, I. Shin, N. Thai, M. Thai, A General Approach for Modules Identification in Evolving Networks, Dynamics of Information 40 (2010) 83–100.

[36] M. Toyoda, M. Kitsuregawa, Extracting evolution of web communities from a series of web archives, in: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, ACM New York, NY, USA, 2003, pp. 28–37.

[37] J. Sun, C. Faloutsos, S. Papadimitriou, P. Yu, Graphscope: parameter-free mining of large time-evolving graphs, in: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA, 2007, pp. 687–696.

[38] M. Rosvall, Mapping change in large networks, PLoS One (2010) 1–9.

[39] R. Kumar, J. Novak, P. Raghavan, A. Tomkins, On the bursty evolution of blogspace, World Wide Web 8 (2005) 159–178.

[40] J. Kleinberg, Bursty and Hierarchical Structure in Streams, Data Mining and Knowledge Discovery 7 (2003) 373 – 397.

[41] L. Backstrom, D. Huttenlocher, J. Kleinberg, X. Lan, Group formation in large social networks: membership, growth, and evolution, Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (2006) 54.