Multi-Step Community Detection and Hierarchical Time Segmentation in Evolving Networks

Thomas Aynaud
UPMC Univ Paris 06, UMR 7606, LIP6, F-75252, Paris, France
thomas.aynaud@lip6.fr

Jean-Loup Guillaume
UPMC Univ Paris 06, UMR 7606, LIP6, F-75252, Paris, France
jean-loup.guillaume@lip6.fr

ABSTRACT

Many complex systems composed of interacting objects like social networks or the web can be modeled as graphs. They can usually be divided in dense sub-graphs with few links between them, called *communities* and detecting this underlying community structure may have a major impact in the understanding of these systems. We focus here on evolving graphs, for which the usual approach is to represent the state of the system at different time steps and to compute communities independently on the graph obtained at each time step.

We propose in this paper to use a different framework: instead of detecting communities on each time step, we detect a unique decomposition in communities that is relevant for (almost) every time step during a given period called the *time window*. We propose a definition of this new decomposition and two algorithms to detect it quickly. We validate both the approach and the algorithms on three evolving networks of different kinds showing that the quality loss at each time step is very low despite the constraint of maximization on several time steps.

Since the time window length is a crucial parameter of our technique, we also propose an unsupervised hierarchical clustering algorithm to build automatically a hierarchical time segmentation into time windows. This clustering relies on a new similarity measure based on community structure. We show that it is very efficient in detecting meaningful windows.

Categories and Subject Descriptors

[**Data**]: Social networks, Graphs; [**Algorithms/Models**]: Clustering, Classification

General Terms

Community detection, evolving graphs, social networks, complex networks, web, clustering

Introduction

Many complex systems are modeled by graphs, or complex networks, which are particularly suited for representing large groups of interacting objects. For instance, the web is a graph of web pages interacting through hyperlinks and the brain is a graph of neurons connected through synapses. Even if they model systems of very different types, these graphs share common non-trivial properties like a low average distance and a small global density balanced with a high local density. This last property means that there are few links inside these graphs but that links are grouped, which suggests that there exist groups of nodes with many links inside the groups but few between them. These groups of nodes are called *communities* and finding and understanding this underlying structure may have a major impact in the global understanding of the modeled systems. For instance, the detection of these communities can be used for graph visualization [2] or to mine various kinds of graphs: they can be groups of interest in a social network [4, 33], web pages dealing with the same subject [10], proteins that share a common function in a metabolic network [36] or modules in a software source code [18].

In the last decade, many definitions have formalized this concept and many algorithms have been proposed to compute these communities in large graphs using only topological information in an efficient way [12, 25, 27]. The Louvain Method [7] in particular allows to detect communities quickly and efficiently with enlightening results.

Complex networks model systems that are usually evolving (pages appear or disappear on the web, contacts between people evolve over time...). However, most of these graphs are still studied as static graphs, *i.e.* without considering time. These studies therefore overlook much information that would be crucial to fully understand phenomena taking place in the network. Now that more and more data containing temporal information becomes available, algorithms devoted to evolving graphs are required. However, finding and analyzing communities on evolving graphs raises new complex issues. For example, an evolving graph can be seen as a sequence of static graphs, but it raises the issue of deciding what happens between snapshots.

In this paper, we propose a novel approach that consists in detecting communities that are satisfying on a given long period. To achieve this, we detect only *one* partition in communities for a given period that will be relevant for every time step of this period and will thus represents the structure of these snapshots at once. Obviously, considering one unique clustering that is relevant for several snapshots cannot be as good on a given snapshot as partitions found by a static algorithm that specifically optimizes it for this particular snapshot, but we will show that we can find partitions having equivalent quality even during long periods. Relying on an existing static definition allows us to quantify the loss of quality and to validate more efficiently our results. This unique decomposition will represent a more global structure since it will remain valid for a long time called the *time window*.

Choosing a relevant time window is of course an issue since the graph may have a given structure for some time and then change during another period. For instance, a contact graph is different between day and night since people are often with colleagues during daytime and with family during evening and night. Furthermore, the structure during the day also evolves with meetings, lunches or breaks: time windows can therefore contain smaller natural windows. We propose to identify automatically such changes by building a hierarchical decomposition of time in which each period can be associated to a clustering of the graph.

This paper contribution is thus twofold: first a new definition of multi-step communities with two algorithms to detect them on very large datasets and second an automatic method to hierarchically extract meaningful time windows. This paper is organized as follows: we first introduce some background and related works and then we define precisely the concept of multi-step communities. In section 3 we propose two algorithms to efficiently detect such communities and, in section 4, we evaluate these algorithms on three evolving graphs with very different features. Finally, we present and validate the hierarchical time decomposition in meaningful time windows, before concluding and presenting the perspectives of this work.

1. BACKGROUND AND RELATED WORK

1.1 Communities

Detecting communities in complex networks has raised a lot of interest and many algorithms have been proposed. Usually, people try to find a partition π of the nodes which can be evaluated by a *quality function* that gives a score to π. The *modularity* [20] is a widely used quality function which compares, for each community, the proportion of links inside the community with a null model claiming that meaningful communities are more densely connected than a null model would expect. The classical null model is a random graph with the same degree sequence as the graph under study. Using this null model, with L the total number of links, l_s the number of links inside a community s and d_s the total degree of s, the modularity of a partition π is:

$$Q(\pi) = \sum_{s\in\pi} \frac{l_s}{L} - \left(\frac{d_s}{2L}\right)^2$$

Since the modularity is null when all nodes are grouped in a single community, interesting modularities are always positive and the higher the modularity, the better the partition. Readers can refer to [6] for tighter bounds and a more precise study of the modularity and to [26] for expected values on random graphs. The modularity's main drawback is that it favors large communities [6]. However, it is very natural for studying graph structure and it can be easily extended to more complex cases, such as directed or weighted graphs and overlapping communities [17, 21].

Finding the partition that maximizes the modularity on a given graph is NP-hard and many approximation algorithms have been proposed (see [12, 25, 27] for very complete surveys). Modularity is one of the few quality functions that can be optimized on huge graphs. Indeed, networks of billion links have been partitioned in a few hours [7] and this efficiency is required in order to study evolving graphs composed of many huge static graphs. Since we are using heuristic algorithms, a low value of the modularity can mean either that the algorithm does not succeed in finding meaningful structure or that this structure does not exist. With this limitation in mind, we will use in this paper the Louvain Method, which is the fastest modularity optimization algorithm [7] producing very high quality results. We describe it in more details in section 2.2.

1.2 Communities in evolving graphs

Evolving graphs can generally be described as a sequence of static graphs where several modifications occur between consecutive snapshots. On such graphs, two main approaches have been followed to study the evolution of communities: computing communities for each snapshot and then tracking communities among them, or using the temporal information directly during the detection, *i.e.* detecting communities on the evolving graph itself.

The first approach is much simpler since it consists in community detection at each time step, which can be done with any algorithm. However, it requires the tracking of the communities *i.e.* following what happens between two consecutive snapshots. A community may remain stable, split, appear, disappear, or merge with another one for instance. The intuitive method is to compare two communities of consecutive time steps with rules based on the size of their intersection [1, 14, 29]. These rules can be used conjointly with the clustering algorithm [23] or simplified by tracking only specific core nodes such as the ones defined in [34] which would be more representative of their community than others. Many workarounds are required because of stability issues: the static algorithms used on each snapshot are often non-stable and hence produce different results even if the input graph does not change. This produces noise that makes the tracking very difficult. To solve this issue, one may consider only stable communities [14] or use a very constrained algorithm [23].

The second approach consists in directly integrating time into the computation. Different techniques have been proposed, such as probabilistic models [32], modified algorithms like streaming algorithms [22], modification of static algorithms [3], use of new objects like sliced networks [19] and finally new definitions of communities [31]. A new definition proposed in [16] splits the quality function in two terms: one for the quality of a snapshot partition and one for the stability. It can be quickly summarized in defining the quality Q_{dyn} as $Q_{dyn} = Q_{snapshot} + \alpha Q_{stability}$ where $Q_{snapshot}$ is a static quality function, modularity for instance, $Q_{stability}$ is a stability term and α is a parameter that allows changing the importance of the stability. Instead of a stability term, [28] extends this idea and proposes to add an overall quality term. The partition found at time t does not have to be close to the partition at time $t-1$ but must be a good partition at time t and a fairly good partition at time $t-1$.

We propose in this paper to extend this idea by computing communities that are almost always good on a given time period.

Finally, extracting interesting time windows based on structural changes has attracted attention mainly to perform event detection by studying how much new information is brought by the new snapshot [30] and by discovering correlated spatio-temporal changes [9]. We will extend the notion of time windows to a hierarchy of windows that do not have to be consecutive in section 4, based on our notion of multi-step communities.

2. MULTI-STEP COMMUNITIES

2.1 Definition

We consider here that an evolving graph is a succession of static graphs, each of them representing the state of the complex network at a given time. The evolving graph G on a set of snapshots $S = \{1, 2, ..., n\}$ is $G = \{G_1, G_2, ..., G_n\}$ with $G_i = (V_i, E_i)$ the snapshot i with nodes V_i and edges E_i. We denote by $V = \cup_{i \in \{1,...,n\}} V_i$ the set of all the nodes and a clustering, *i.e.* communities, is a partition of V. We also define a time window T as a subset of the possible snapshots, *i.e.* $T \subseteq S$. In many situations, not all snapshots have the same importance and we may assign a weight w_i to snapshot i. A possible use of weights is to consider snapshots that are not regularly spaced in time: for instance if snapshot i represents the state of the network for a long period, then w_i could be related to the length of the period. It is also possible to give an increasing importance to recent snapshots using a weight like: $w_i = \frac{i}{n}$. Then, we define $Q_{avg}(G, \pi, T)$ the average modularity of the partition π of V for a given time window T as:

$$Q_{avg}(G, \pi, T) = \frac{1}{\sum_{i \in T} w_i} \sum_{i \in T} w_i . Q(G_i, \pi)$$

Where $Q(G_i, \pi)$ is the modularity of the partition π on the static graph G_i considering only the nodes in V_i. We call it the static modularity to differentiate it from Q_{avg}, the average modularity, which applies on evolving graphs and is defined on several snapshots. Detecting communities means finding a partition that maximizes the average modularity. As it is a NP-complete problem [8], we will actually try to find partitions that have the higher average modularity possible.

2.2 Detection algorithms

We propose two algorithms to find a partition of high average modularity on a given time window. The first one consists in building a new graph that is an average representation of the evolving graph and then detecting static communities on it. The second method is a modification of the Louvain Method which is a static modularity optimization algorithm.

To be more precise, the Louvain Method is a hierarchical greedy algorithm designed to optimize the modularity on a static graph (weighted or not). It is composed of two phases, executed alternatively. Initially, each node is in a singleton community. Next, during phase 1, nodes are considered one by one [1]. Each one is placed in its neighboring community (including its own community) that maximizes the static modularity gain. This phase is repeated until no node is moved (the obtained decomposition is therefore a local maximum). Then, phase 2 consists in building a new graph between the communities found during phase 1: there is a node in the new graph for each community and, for two communities C and C', there is a link of weight w where $w = \sum_{v,v' \in C \times C'} weight(v, v')$. There is also a loop on C of weight $\sum_{v,v' \in C \times C} weight(v, v')$ [2]. The algorithm then executes phase 1 and 2 alternatively until the static modularity no longer increases.

[1]The results depend on the order in which nodes are considered. It does not really matter in our results and we use one predefined order for each graph.

[2]This ensures that the partition found after phase 1 has the same static modularity as the partition of the new graph where each node is put in its own community.

2.2.1 *Sum-method*

Rather than directly optimizing the average modularity, we first propose the sum-method to optimize the static modularity on an average representation of the evolving graph. Given an evolving graph $G = \{G_1, G_2, ..., G_n\}$ and a time window $T \subseteq \{1, ..., n\}$, we build a new weighted graph, called the sum graph, which is the union of all the snapshots in T: each edge of the sum graph is weighted by the total time during which this edge exists in T. Since the sum graph is a static weighted graph, we can apply the Louvain Method on it, or any other classical algorithm, and use the result as multi-step communities.

However, the metric optimized this way is slightly different from the average modularity. Given a partition π, we denote by l_{tc} the number of links inside the community c at time t, L_t the number of links of G_t and d_{tc} the total degree of the community c at time t. The total weight of links inside community c is therefore $\sum_{t \in T} l_{tc}$, the total weight of the sum graph is $\sum_{t \in T} L_t$ and the total weighted degree of the community c is $\sum_{t \in T} d_{tc}$. Then, the static modularity of the partition π on the sum graph is equal to:

$$Q_{sum}(\pi, G, T) = \sum_{c \in \Pi} \frac{\sum_{t \in T} l_{tc}}{\sum_{t \in T} L_t} - \left(\frac{\sum_{t \in T} d_{tc}}{2. \sum_{t \in T} L_t} \right)^2$$

If we come back to the definition of average modularity and after reordering the summation, we have:

$$Q_{avg}(\pi, G, T) = \frac{1}{\sum_{i \in T} w_i} \sum_{c \in \Pi} \left(\sum_{t \in T} \frac{l_{tc}}{L_t} - \sum_{t \in T} \left(\frac{d_{tc}}{2.L_t} \right)^2 \right)$$

Therefore the static modularity on the sum graph Q_{sum} is not exactly the average modularity Q_{avg} and we will next propose a second method to directly optimize this quantity. Nevertheless, this first method is very fast since sum graphs are usually much smaller than the sum of the size of all snapshots and we will see that it is a good first approximation since the obtained results have a high average modularity.

2.2.2 *Average-method*

To optimize the average modularity we will modify the Louvain Method. Two reasons explain the efficiency of the Louvain Method. First, when it must decide into which community a node should be moved, it can compute quickly the static modularity gain of each possible move. Second, the size of the considered network is quickly reduced at phase 2 and therefore all subsequent phases are really fast. We must change two elements to adapt the Louvain Method to optimize the average modularity during a time window T: the computation of the quality gain in the first phase and how to build the network between communities in the second one.

One must note that the difference between two partitions is the sum of the differences of the static modularity for each snapshot. Therefore, the average modularity gain can also be easily computed locally: it is the average of the static gains for each snapshot of T. The transformation of the network into a network of communities can be modified in the following way: given a partition π of V, we apply the same transformation as for the Louvain Method on every snapshot of T independently (with different weights for each snapshot) to obtain a new evolving network between the communities of π. We call this algorithm the average-method.

Executing the average-method does not take more time in worst cases than executing the classic Louvain Method on each snapshot of T since it consider less nodes during phase 1. Therefore, it remains applicable to huge datasets composed of thousands of nodes and thousands of time steps. An implementation of this algorithm is available at [13]. We will next analyze the results of these two algorithms.

3. RESULTS ANALYSIS

To validate our approach and the two associated algorithms, we use three evolving graphs with very different characteristics, which show the advantages and the limitations of our approach.

Blogs dataset. During four months, about six thousands blogs have been monitored to track posts, comments and citation links between blogs. We study here the aggregated graph between blogs. We start with an empty graph and every day we add the blogs and links seen this day. The obtained evolving graph grows slowly and regularly during 120 time steps. For more details about the measurement, see [11].

Mrinfo dataset. The second graph represents the topology of multicast routers on the Internet measured with the `mrinfo` tool. This tool allows asking to a multicast router all its neighboring multicast routers. Every day, `mrinfo` was run on a first router and then recursively on every neighbor in a breadth first search fashion. This is not a social networks, but our algorithms and definitions are valid on any kind of evolving graphs and internet topology share many properties with social networks [35, 5]. Thus we consider it as another interesting dataset and include it in our study. The measurement has been performed during several years, yielding a dynamic map of the multicast routers topology. For more details about the measurement, see [24]. We focus here on year 2005, which represents 365 snapshots containing about 3100 nodes on average. The evolution of this graph is divided in three very distinct phases. The first phase spans from the beginning up to day 52. During this phase the graph is very unstable and contains many events. The second phase is between days 52 and 117 and is more stable. Finally, the third phase lasts from day 117 to the end. This last phase is similar to the first one but is longer and contains fewer events.

Imote dataset. The last graph is a sensor graph representing proximity between participants of the Infocom 2005 conference. Participants received a sensor device capable of recording the presence of nearby devices. The experiment started at the conference registration on the evening before the conference itself, then lasted three nights and days. The measurement ended at the end of the third morning. The measurement is very noisy since bluetooth devices often fail to detect each other. People move and so links are quickly changing, resulting in a graph with many transformations. It is composed of 41 nodes and 25000 snapshots, which are not regularly distributed in time. This allowed us to weight snapshots with the effective duration between them. Note that the graph is very stable during night, with only a few connections, and very dynamic during days. For more details see [15].

The three graphs have very different characteristics. Mrinfo and Blogs are quite stable and Imote is very dynamic; Imote contains many snapshots and Blogs only a few. Finally, Blogs is growing (no link is ever removed) and therefore the last snapshot contains all the information. We will see in the sequel that the two algorithms provide results which are related to these characteristics.

3.1 Average modularity

To study the efficiency of the algorithms we compare the quality of several partitions during the whole measurement and the time window T is then $\{1, ..., n\}$. The s-partition is the result of the sum-method (see Section 2.2.1) and the a-partition the result of the average-method (see Section 2.2.2). We have also computed the partitions found by the classic Louvain Method for every snapshot independently. These are the static partitions. As they have the best static modularity we may expect, we compare their average static modularities to the values obtained with both our methods.

	s-partition	a-partition	Average static
Blogs	0.6 (2min 20s)	0.61	0.62 (2min 40s)
Mrinfo	0.91 (28s)	0.92	0.923 (34s)
Imote	0.38 (16s)	0.39	0.56 (8s)

Table 1: Average modularities of studied partitions during whole measurement and time of the optimization on a regular desktop personal computer. a-partitionnever takes more than a few seconds.

Table 1 presents the obtained modularities for the 3 methods. While a-partition and s-partition have very similar quality, the a-partition is always better. As the s-partition is faster to compute, it remains interesting, but from now on we will always use the a-partitions and call them multi-step partitions.

Finally, one can see that the multi-step partition has a comparable average modularity to the maximum one in the Blogs and Mrinfo graphs. We insist that this maximum cannot be achieved because this would require a snapshot so that the unique partition optimized for every snapshot has a better static modularity on this snapshot than the partition specifically optimized for it without consideration of others. This is not realistic since you cannot find higher static modularity by adding more constraint. If the evolving graph is composed of only one snapshot, both methods are exactly the classic Louvain Method.

The most difficult case for our algorithm is when the community structure changes drastically, for instance when a few nodes that are strongly connected stop being connected and change their links to nodes outside their community. It happens in the Imote graph and we will see more precisely in the sequel some cases where the algorithm finds a good structure and some where it fails.

3.2 Static modularity of multi-step partitions

To obtain more precise information about the quality of partitions, Figure 1 presents the static modularity of these partitions over time. We have also added the best static partition which is the partition among all the static partitions which has the highest average modularity. Their average modularity are 0.6 for Blogs, 0.61 for Mrinfo and 0.34 for Imote. Excepted for Blogs, the best static partitions are not good compared to s-partition and a-partition.

In the Blogs graph, multi-step partition and static partition quality are regularly getting closer confirming that static and average structure are getting closer. Similarly, the best static partition challenges other approaches. This is because this graph is always growing. Therefore, the last snapshots are very similar to the sum graph and then the last static partitions have a high average modularity.

In the Mrinfo graph, the multi-step partition has almost always the same quality as the static. Thus, we capture the overall structure of this graph with only one partition. It works even during phase 2 between days 52 and 117. It seems that parts of the set of nodes change during this period, and thus this event does not affect the partition of the other nodes. The best static partition here clearly fails in discovering a global structure since it is good only during a few snapshots. This shows the interest of looking for a relevant multi-step partition on the whole measurement rather than just selecting one snapshot which cannot be representative of the whole.

In the Imote graph, it appears that the structure changes a lot over time and there is a clear day/night effect. The multi-step partition is a lot more effective during night, since the graph is almost static and groups are clearly separated. Conversely it fails during the day when the structure is less stable and thus more difficult to identify. The best static partition suffers from same default and has a lower quality.

Finally, the efficiency of the average modularity depends on graph characteristics. With Mrinfo and Imote, the multi-step partition gives new information since there is no representative time of the average structure whereas for Blogs detecting the structure of the last snapshots is sufficient.

3.3 Nodes existence

An interesting insight is the existence or not of nodes during the time window. The partition is unique and thus static, but the graph evolves. Nodes and links appear and disappear and although we only have one partition, communities still evolve. Some nodes may even never exist at the same time but be strongly connected to a very stable core in a multi-step community. To represent this, we have drawn a picture for every community where each line corresponds to a node and each column to a time step (see Figure 2). If the $n-th$ node of the community exists at time t, the point (t, n) is white and conversely the point is black if it does not exist. To have a result which does not look noisy and is understandable, we must order the nodes carefully. We have chosen one simple order which gives good results: nodes are first ordered according to their first appearance time

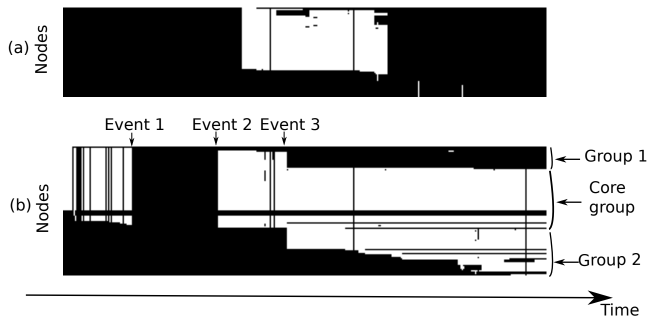

Figure 2: Nodes existence diagrams in Mrinfo dataset for two interesting communities.

since nodes that appear together are often similar, and then sorted according to the number of snapshots during which they exist (it appears that this approach puts together nodes involved in the same events).

This graphical representation is not interesting for all graphs and communities since we cannot display many nodes nor many snapshots. Furthermore the multi-step partitions are not always meaningful and then trying to display them is not very useful. In Blogs, communities grow slowly and there is nothing really interesting to show. For Imote, there are too many snapshots divided in phases where we do not find relevant average structure so we do not represent it. The most interesting results are obtained on Mrinfo, see Figure 2.

The community (a) illustrates the simplest case: all nodes exist almost during the same time except a few outliers. Nodes appear and disappear together which is revealing that the grouping is relevant. The community (b) is a more complex case and allows the identification of sub-clusters: a core group composed of nodes that are the most present and two groups that exist at different moments. Thus at the beginning, group 1 and core group exist and disappear during phase 2 between days 52 and 117 before reappearing for a few days. Then, group 1 disappears and group 2 replaces it.

Such drawings illustrate that even if the partition is unique, nodes may exist or disappear during the measurement and therefore the communities are evolving. This reveals new information about community structure, hence looking locally inside a community allows us to study an interesting evolution.

4. TIME WINDOW AND HIERARCHY

Until now, we have always optimized the average modularity over the whole experiment time and we have seen that in some cases this leads to communities of lower interest since the average structure changes too much. We will now exploit the fact that we can choose the time window T. It can be an interval if we are interested only in consecutive snapshots, but it is not mandatory. For example, one may imagine a repeated social structure every day, which would therefore not be composed of consecutive snapshots, and detecting it would be interesting.

4.1 Hierarchy

In Imote the structure changes between days and nights, but it changes also during days with several sessions, lunches, keynote, etc. Thus, time windows can naturally contain

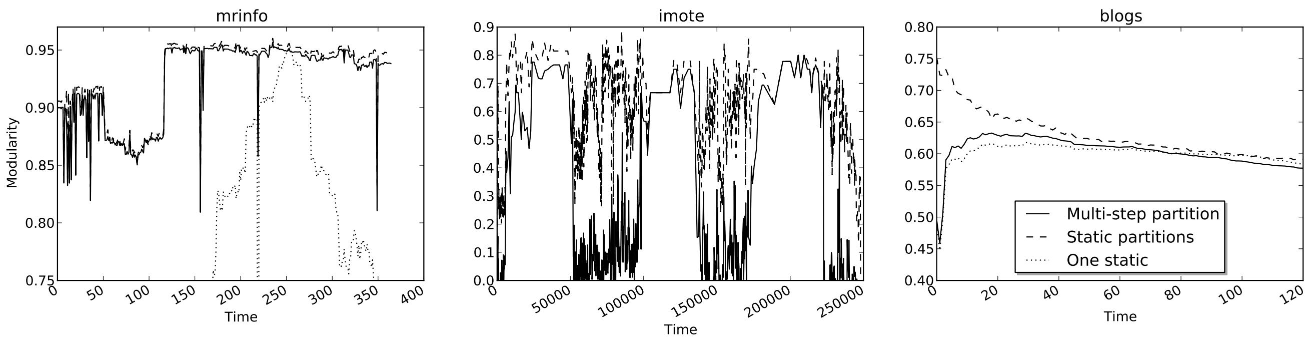

Figure 1: Static modularities of the studied partitions on each snapshot.

smaller windows. This kind of hierarchical segmentation is usually represented as a tree in which the leaves are the individual time steps, which are hierarchically grouped using a similarity criterion. We propose an agglomerative hierarchical time clustering algorithm (algorithm 1): at the beginning, every snapshot is alone and we group recursively the two most similar windows until no possibility remains. This algorithm requires a similarity measure to decide which time windows should be merged.

Algorithm 1 Hierarchical time window merge algorithm.

```
 1: $G$ the initial graph
 2: $L$ the list of potential snapshots, initially empty
 3: for all snapshots $t$ of $G$ do
 4:     Compute the communities $\pi_{\{t\}}$ on the snapshot $t$
 5:     Put $\{t\}$ in $L$
 6: end for
 7: while $L$ not empty do
 8:     Find $T_i$ and $T_j$ in $L$ which maximize $Sim(T_i, T_j)$
 9:     Remove $T_i$ and $T_j$ of $L$ and add $T = T_i \cup T_j$ in $L$
10:     Output that $T_i$ and $T_j$ have been merged
11:     Compute $\pi_T$ the communities of $G$ on the window $T$
12: end while
```

4.1.1 Similarity

To define a similarity between time windows, we will rely on the associated partitions. We claim that if two time windows are structurally similar, the multi-step partition of one, which summarizes its structure, will be a relevant decomposition for the other and conversely. This can be evaluated for two time windows T_i and T_j associated to the partitions π_i and π_j by the similarity:

$$Sim(T_i, T_j) = Q_{avg}(G, \pi_i, T_j) + Q_{avg}(G, \pi_j, T_i)$$

We do not perform direct comparison of the partitions because of the instability of the decomposition algorithms as discussed earlier and in [3]. Conversely, the modularity is very stable and therefore we use it to test if both time windows have a similar structure.

Some extensions of the algorithm are possible. For example, one may want to force the merged time windows to be consecutive: in this case, the results are easier to interpret but repeated structures will be missed if they are not consecutive. In the following we will also forbid the merge of time windows whose similarity is negative [3]. With these rules, the algorithm can produce several disjoint trees rather than a simple one.

This algorithm can produce time segmentation quickly. If the evolving graph is made of n snapshots, there are initially n community detections to perform and one more community detection for each merge, *i.e.* at most n more. Thus, there are $2n$ executions of the sum-method (but most on smaller time windows than the whole dataset) and each partition must be evaluated on each snapshot, which is equivalent to iterating over all the edges. The exact complexity of this algorithm relies on the complexity of the Louvain Method, which is unknown. For instance, the times segmentation takes 1 hour and 30 minutes for Mrinfo, 25 minutes for Blogs and 8 hours for Imote (the high number of snapshots induces a very large tree). If we use the constraint of merging only consecutive time windows, these lengths drop to 17 minutes for Mrinfo, 11 minutes for Blogs and 1 minute 50 seconds for Imote.

4.2 Time window hierarchies

Validation of the results is an important problem and is now performed experimentally. We have checked that time windows identified by the algorithm correspond to understandable phenomena in the graphs. This validates both the algorithm and the similarity function. We have built the tree for each dataset with and without forcing time windows to be consecutive. Note that if we force time windows to be consecutive, a strong modification of the structure during one time step prevents time windows before and after this event to be merged and results in a split. This validation process is limited, but there do not exist an objective benchmark for dynamic community detection.

With the Mrinfo dataset using consecutive time windows (Figure 3 top), the highest level, which contains the largest time windows, is more granular than the 3 phases described before. Indeed, there are many events during the first phase which therefore cannot be identified as one consecutive time window. The group of snapshots between 52 and 116 corresponds to phase 2 and the time windows after correspond to the phase 3 also separated in four sub-windows by small events. The first level hence reflects almost exactly what we were expecting and we can explain many separations at

[3]the trivial partition where all nodes are in one community has a modularity of 0, so a negative modularity describes a worse decomposition!

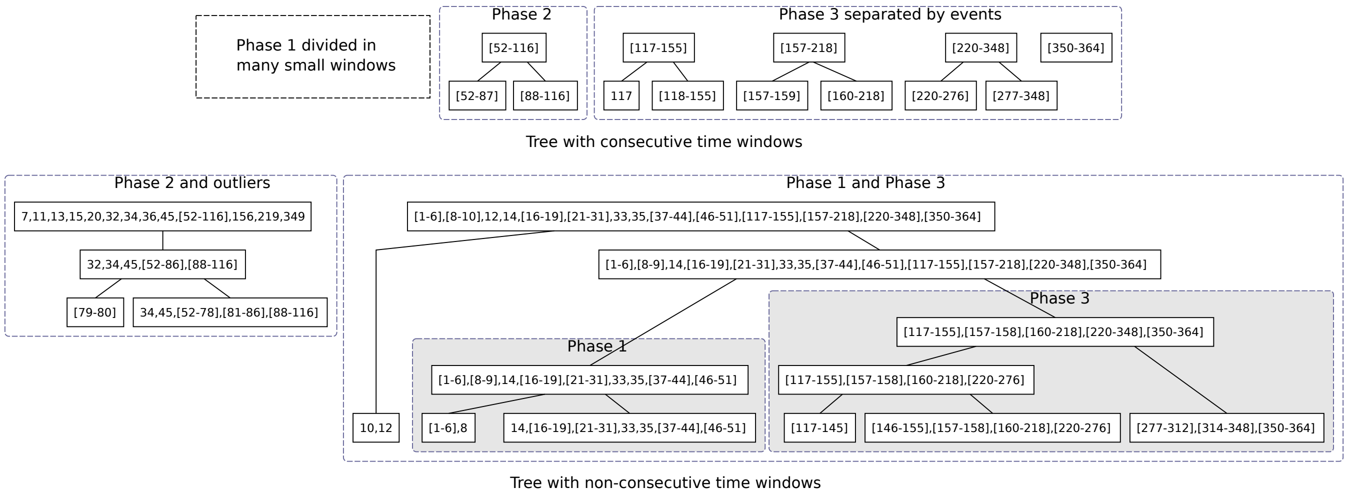

Figure 3: First levels of the hierarchical time decomposition on the Mrinfo dataset. Numbers in tree nodes represent time windows. For instance, "32,34,45,[52-86],[88-116]" corresponds to the time window containing days 32, 34, 45, from 52 to 86 and from 88 to 116. We have grouped time windows into blocks with potential explanations when possible.

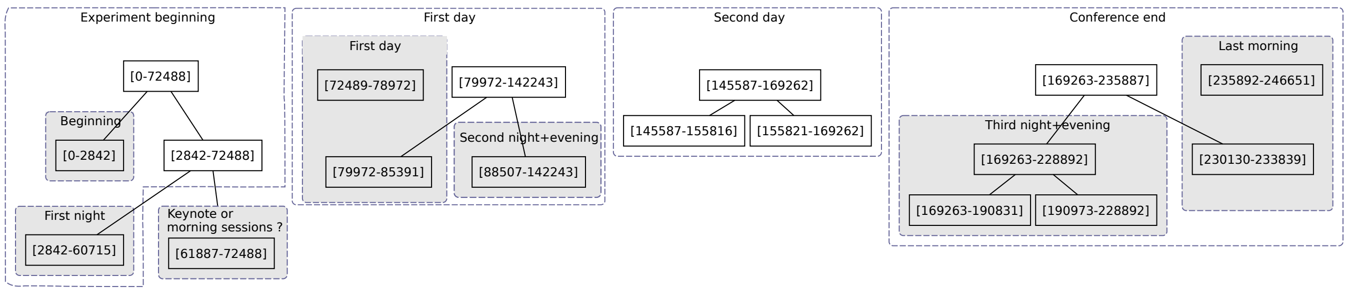

Figure 4: First levels of the hierarchical time decomposition on Imote dataset using consecutive time windows. Times are in seconds.

lower levels.

When considering the decomposition obtained with non-consecutive time windows (Figure 3 bottom), the effect caused by events disappears. The first level is composed of two groups: one containing phase 2 and some outliers and one containing phase 1 and 3, which are quite similar structurally. This last group is then subdivided into phase 1 and phase 3. Sub-levels are separated by events. Moreover, without any constraint of consecutive grouping, we remark that time windows are often almost consecutive except for events, showing that the algorithm tends to merge similar time windows anyway.

Imote dataset presents similar results. Using consecutive time windows, the highest level contains several time windows that correspond to real moments (see Figure 4). The first one corresponds to the beginning of the experiment containing three groups: 45 minutes after the beginning, a first long period corresponding to the first night and then another 2 hours and a half, which may correspond to the breakfast and keynote speech (we do not have the exact experiment timing so we can only conjecture). Then there is a small group that may correspond to a part of the day 1. The next window is composed of another part of the day 1 and of the second night. The following time window contains mostly the second day, which is then divided in two parts that al-

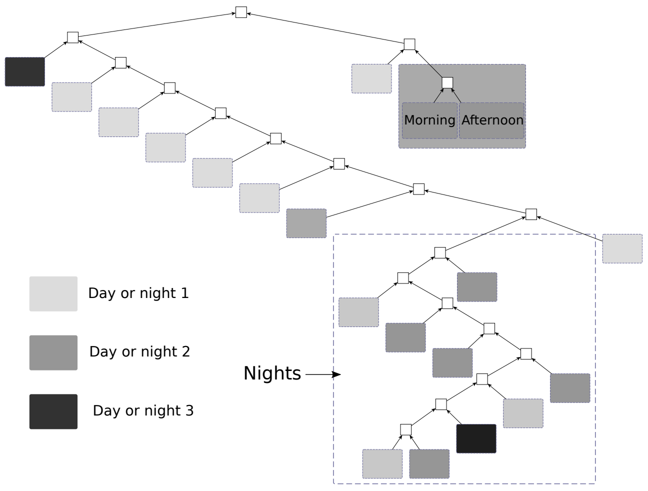

Figure 5: Biggest tree of the hierarchical time segmentation of the Imote dataset using non-consecutive time windows. Blocks represent big subtrees to be able to represent the whole structure.

most match the morning and the afternoon. The next time window is composed of the last night and a part of the last morning. The last window contains the remaining moments of the last morning. Thus, the time segmentation algorithm seems to detect easily explainable time windows.

We present a sketch of the structure of the largest tree (the others are very small) of the decomposition built using non-consecutive time windows on Figure 5. The tree is huge and there are many sub-levels thus we can only give an idea of the whole structure and each block is in fact a big sub-tree. Sub-trees are composed of snapshots of the same day, showing that graph structure gives meaningful information. The two biggest groups are the day 2 and the nights. Day 2 is then divided into the morning and the afternoon and the night sub-tree contains many sub-trees depending on the day.

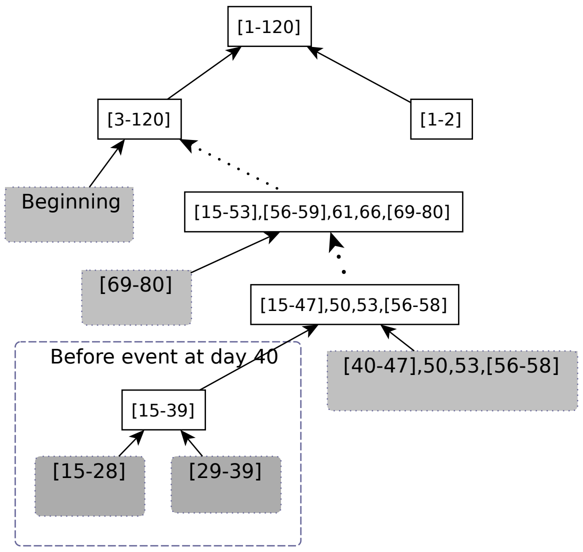

Figure 6: Tree of the hierarchical time segmentation of Blogs. Gray blocks represent sub-trees and dashed arrows long paths with only one snapshot added at each step.

With Blogs, consecutive and non-consecutive trees are very similar. They are unbalanced and almost degenerate into a list which is consistent with the fact that the graph regularly grows: there is no clear notion of time windows. We hence draw only a sketch of the non-consecutive tree on figure 6. There are only few long time windows. The highest windows contain the first snapshots, which are the most different from the average structure. We do not succeed in interpreting the other windows because of a lack of knowledge about the measurement. We only know that a measurement event happened at day 40 that can be seen in the two lowest windows.

Finally, results seem very promising. Time segmentation is often meaningful with graphs that exhibit an evolution we can divide in time windows. Both approaches, using consecutive and non-consecutive time windows are complementary: consecutive time windows are easier to interpret and produce directly interesting results but non-consecutive are less sensitive to events and can therefore detect some long range or periodic connections.

5. CONCLUSIONS AND PERSPECTIVES

We have shown that communities can be detected not only on one given snapshot but also on a given long period by optimizing an average quality function. We have proposed two algorithms to optimize this quality: one is very fast but the other produces better results and remains applicable to huge datasets with many large snapshots.

The new temporal quality function is very general and weights allow to emphasize some snapshots or to consider their duration. We used here the modularity as the basic quality function but others could have been used. Using a known static quality function is an asset to validate our results: if the static quality function is valid and we have the same quality on one snapshot, then our new method is also valid on this snapshot. Having communities that span on several snapshots and diagrams like the nodes existence over time provide new analysis tools to obtain insights about lives of communities.

Optimizing on a given period raises the issue of the relevant length of time windows since using a long period is not always satisfying. We have proposed a method to extract automatically interesting time windows that scales to evolving graphs of thousands of nodes and snapshots. Non-consecutive time windows can be detected using an agglomerative hierarchical time clustering algorithm which uses a similarity function based on the snapshots structure: if two snapshots have the same structure, the multi-step partition of one will be relevant on the other and conversely.

These results give new insights on the complex networks under study. The construction process of Blogs has a significant impact on the conclusions we can draw and maybe some links should be removed some time after their creation to obtain a temporal structure that is more meaningful on such web graphs. Our algorithms are most successful on the Mrinfo and Imote datasets where clear phases exist, which can be detected and described. This experimental validation is still limited and this is a major concern for all dynamic community detection algorithms: there exist no consensual benchmark or dataset with ground truth. As several algorithms now exist, we should develop such tools for objective comparison.

Further studies will also focus on other similarity functions. The current similarity function measures both how similar and modular time windows are and normalization might correct this default. Another limitation of the new average modularity is due to the summation itself. The sum is commutative and thus the order of the snapshots has no influence on the results. This means that there is no causality and that, for instance, the succession of snapshots G_1, G_2, G_3 is strictly equivalent to the succession of snapshots G_3, G_2, G_1. As a first approximation, it is all right, but it would be great to adapt the metric to deal with causality.

Acknowledgments

This work was partly funded by a grant from the *Agence Nationale de la Recherche*, with reference ANR-10-JCJC-0202 and by the project WebFluence #ANR-08-SYSC-009.

6. REFERENCES

[1] S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *Proc. of the 13th ACM SIGKDD*, pages 913–921. ACM, 2007.

[2] D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale visualization of small world networks. In *Proc. IEEE Symposium on Information Visualization*, pages 75–81, 2003.

[3] T. Aynaud and J.-L. Guillaume. Static community detection algorithms for evolving networks. In *WiOpt Workshop on Dynamic Networks*, pages 508–514, 2010.

[4] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. *Proc. of the 12th ACM SIGKDD*, pages 44–54, 2006.

[5] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509, 1999.

[6] M. Barthélemy and S. Fortunato. Resolution limit in community detection. *Proc. of the National Academy of Sciences of the United States of America*, 104(1):36–41, 2007.

[7] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *J. Stat. Mech*, 10008:1–12, 2008.

[8] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing Modularity is hard. *ArXiv Physics e-prints*, 2006.

[9] J. Chan, J. Bailey, and C. Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems*, 16(1):53–96, 2008.

[10] Y. Chi, S. Zhu, X. Song, J. Tatemura, and B. L. Tseng. Structural and temporal analysis of the blogosphere through community factorization. *Proc. of the 13th ACM SIGKDD*, pages 163–172, 2007.

[11] J.-P. Cointet and C. Roth. Socio-semantic Dynamics in a Blog Network. *2009 International Conference on Computational Science and Engineering*, (6):114–121, 2009.

[12] S. Fortunato. Community detection in graphs. *Physics Reports*, (486):75–174, 2010.

[13] J.-l. Guillaume. Web Page, http://jlguillaume.free.fr/www/programs.php.

[14] J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Tracking evolving communities in large linked networks. In *PNAS*, volume 101, pages 5249–5253. National Acad Sciences, 2004.

[15] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. *Proc. of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking - WDTN '05*, pages 244–251, 2005.

[16] R. Kumar, A. Tomkins, and D. Chakrabarti. Evolutionary clustering. In *In Proc. of the 12th ACM SIGKDD*, pages 554–560. ACM Press, 2006.

[17] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11), 2007.

[18] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *Proc. 6th Intl. Workshop on Program Comprehension*, pages 45–53, 1998.

[19] P. Mucha, T. Richardson, K. Macon, and M. A. Porter. Community structure in time-dependent, multiscale, and multiplex networks. *Science*, 876:10–13, 2010.

[20] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):26113, 2004.

[21] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *J. Stat. Mech.*, 2009:P03024, 2009.

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

[23] G. Palla, A.-L. Barabasi, and T. Vicsek. Quantifying social group evolution. *Nature*, 446:664–667, 2007.

[24] J. Pansiot, P. Mérindol, B. Donnet, and O. Bonaventure. Extracting Intra-Domain Topology from mrinfo Probing. In *Passive and Active Measurement*, 2009.

[25] M. A. Porter, P. J. Mucha, and J.-P. Onnela. Communities in Networks. *Notices of the American Mathematical Society*, 56:1082–1097, 2009.

[26] J. Reichardt and S. Bornholdt. When are networks truly modular? *Physica D Nonlinear Phenomena*, 224:20–26, 2006.

[27] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1:27–64, 2007.

[28] X. Song, Y. Chi, B. L. Tseng, D. Zhou, and K. Hino. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proc. of the 13th ACM SIGKDD*, pages 153–162. ACM, 2007.

[29] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. Monic: modeling and monitoring cluster transitions. In *Proc. of the 12th ACM SIGKDD*, pages 706–711. ACM, 2006.

[30] J. Sun, C. Faloutsos, S. Papadimitriou, and P. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proc. of the 13th ACM SIGKDD*, pages 687–696. ACM, 2007.

[31] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. *Proc. of the 13th ACM SIGKDD*, pages 717–726, 2007.

[32] B. L. Tseng, Y.-R. Lin, Y. Chi, S. Zhu, and H. Sundaram. Analyzing communities and their evolutions in dynamic social networks. *ACM Transactions on Knowledge Discovery from Data*, 3(2):1–31, 2009.

[33] M. L. Wallace, Y. Gingras, and R. Duhon. A new approach for detecting scientific specialties from raw cocitation networks. *J. Am. Soc. Inf. Sci. Technol.*, 60:240–246, 2009.

[34] Y. Wang, B. Wu, and N. Du. Community Evolution of Social Network: Feature, Algorithm and Model. *Science And Technology*, (60402011), 2008.

[35] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442,

1998.

[36] J. Zhao, H. Yu, J. Luo, Z. Cao, and Y. Li. Hierarchical modularity of nested bow-ties in metabolic networks. *BMC bioinformatics*, 7(386), 2006.