OLCPM: An Online Framework for Detecting Overlapping Communities in Dynamic Social Networks

Souâad Boudebza, Rémy Cazabet, Faiçal Azouaou, Omar Nouali

OLCPM: An Online Framework for Detecting Overlapping Communities in Dynamic Social Networks

Souâad Boudebza[a,*], Rémy Cazabet[b,d], Faiçal Azouaou[a], Omar Nouali[c]

[a] *Ecole nationale Supérieure d'Informatique, BP 68M, 16309, Oued-Smar, Alger, Algérie. http://www.esi.dz*
[b] *Sorbonne University, UPMC University, CNRS, LIP6 UMR 7606, Paris, France.*
[c] *Division de Recherche en Thorie et Ingnierie des Systmes Informatiques,CERIST, Rue des Frres Aissiou, Ben Aknoun, Alger, Algrie.*
[d] *Univ Lyon, Universite Lyon 1, CNRS, LIRIS UMR5205, F-69622 France.*

Abstract

Community structure is one of the most prominent features of complex networks. Community structure detection is of great importance to provide insights into the network structure and functionalities. Most proposals focus on static networks. However, finding communities in a dynamic network is even more challenging, especially when communities overlap with each other. In this article, we present an online algorithm, called OLCPM, based on clique percolation and label propagation methods. OLCPM can detect overlapping communities and works on temporal networks with a fine granularity. By locally updating the community structure, OLCPM delivers significant improvement in running time compared with previous clique percolation techniques. The experimental results on both synthetic and real-world networks illustrate the effectiveness of the method.

Keywords: Community Detection, Temporal Network, Dynamic, Overlapping, Social Network, Clique, Label Propagation

2010 MSC: 00-01, 99-00

*Corresponding author
Email address: s_boudebza@esi.dz (Souâad Boudebza)

1. Introduction

The analysis of complex networks is a fast growing topic of interest, with applications in fields as various as neural networks, protein networks, computer networks or geographical networks. One of the most prominent application domain is social network analysis.

The study of social networks can be traced back to the beginning of the 19th century, since the initial work on sociometry [1]. This subject has gained new momentum in recent years, mainly due to the advent of the information age and internet, which has led to the extensive popularity of online social networks, producing large social datasets that can be studied by researchers. The goal of social network analysis is to analyze relationships among social entities and to understand the general properties and features of the whole network, typically by means of graph theory. Nodes in the graph represent social actors within the network (people, organization, groups, or any other entity) and edges characterize social interactions or relations between nodes (friendship, collaboration, influence, idea, etc.).

One of the most prominent features of social networks is their community structure, characterized by the existence of nodes collections called communities, where nodes within a collection tend to interact more with each other than with the rest of the network [2]. Individuals within the same community often share similar properties, such as interests, social ties, location, occupation, etc. Therefore, the ability to detect such communities could be of utmost importance in a number of research areas, such as recommender systems [3][4], email communication [5], epidemiology [6], criminology [7], marketing and advertising [8, 9], etc.

There are many challenges facing community detection. One of the most important, in particular for social networks, is *overlap of communities*: in such networks, individuals often belong to several social groups. For instance, individuals often belong to familial and professional circles; scientists collaborate with several research groups, etc. The second challenge lies in the fact that

real-world communities are time-evolving. The community structure changes as the social entities and their interactions evolve. These changes can be modeled as addition and removal of nodes and edges from the graph. For instance, in online social networks like Facebook, changes are introduced by users joining or withdrawing from the network, or by people adding each other as "friend". These changes may lead to a significant transformation of the network community structure. Palla et al.[10] propose six types of events which may occur during the evolution of communities: birth, growth, shrink, merge, split, and death. The communities can grow or shrink, as members are added or removed from an existing community. As time goes by, new communities can be born, and old communities may disappear. Two communities can become closely related and merge into a single one, or, conversely, a single community can split into two or more distinct ones.

1.1. Rationale for an online version of the Clique Percolation Method

A growing number of methods have been proposed to reveal overlapping and evolving community structures [11, 12]. One of the most prominent of these methods was proposed by Palla et al.[10]. The clique percolation method (CPM) [13] is used to extract the community structure at each time step of an evolving network. Then, communities in consecutive time steps are matched.

The CPM method, thanks to its community definition, has interesting properties compared with other popular methods such as Louvain and infomap [14, 15]:

- It is deterministic, i.e., two runs on two networks with the same topology will yield the same results.
- Communities are defined intrinsically, i.e., each community exists independently from the rest of the network, unlike methods using a global quality function such as the *modularity* [16], that suffer from resolution limits [17] binding the size of communities to the size of the network.
- Communities can overlap, i.e., a node can be part of several communities.

These properties represent an advantage when working with social networks and with dynamic networks. In particular, a well-known problem with the discovery of evolving communities is the so-called instability of methods [18], which can be summarized as follows: because community detection methods are unstable, the difference observed in the partition between two consecutive periods of the network might be due either to significant changes in the network or to random perturbations introduced by the algorithm itself. This problem is due to (1) the usage of stochastic methods, as two runs on very similar (or even identical) networks can yield very different results if the algorithm reaches different local maximum, (2) non-intrinsically defined communities, as a modification of a community might be due to changes introduced in an unrelated part of the network.

Given these observations, CPM appears as a natural candidate to be used for dynamic community detection. The method adapting CPM to the dynamic case [10], however, suffers from at least two weaknesses for which we propose solutions in this article, one due to CPM itself, and other to its adaptation to the dynamic case:

- All cliques need to be discovered anew at each step, both in the new graph snapshot and in a joint graph between snapshots at t and $t-1$, which is computationally expensive for networks with many steps of evolution.
- Nodes must belong to a cliques of size at least k to be part of a community, and as a consequence, some nodes might not be affected to any community. As most social networks have a scale-free degree distribution, a large number of nodes remain without a community.

To circumvent these issues, we propose a new two-step framework for detecting overlapping and evolving communities in social networks. First, built upon the classical algorithm CPM, we introduce an Online CPM algorithm (OCPM) to identify the core nodes of communities in real time. To do that, we propose to use *stream graph* as a network model. At every change in the network, the community structure is updated at the local scale. This allows significant

improvements in computational complexity compared with dynamic CPM [10]. Second, to deal with the coverage problem of CPM, we propose a label propagation post-process (OLCPM)and thus, nodes not embedded in any community will be assigned to one or more communities.

The rest of the paper is organized as follows: section 2 discusses the related work on overlapping and evolving community detection algorithms. In Section 3, we present the different types of dynamic networks and introduce a fully dynamic network model. Section 4 presents the OLCPM framework of dynamic community detection: OCPM algorithm and Label propagation based post process. Experimental results are described in section 5.

2. Related work

In this section, we first introduce the Clique Percolation Method (CPM) [13] and its dynamic version [10], on which our proposal is built on. Then, we present a brief overview of some relevant research work on overlapping and dynamic community detection.

Palla et al.[10] were among the first to propose an approach for dealing with dynamic and overlapping community detection. Their approach has two main steps: i) static community identification and ii) community matching. In the first step, the CPM method [13] is used to extract the community structure at each time step. In this method, a community is defined as the union of all *k-cliques* (complete subgraphs of size k) that can be reached from each other through a series of adjacent $k-cliques$ (sharing $k-1$ nodes). In the second step, communities are matched between consecutive snapshots. The following process is used: for each pair of consecutive snapshots, a joint graph is created, containing the union of nodes and links from both networks. CPM is then applied to the resulting graph. The communities in the joint graph provide a natural connection between communities in the consecutive snapshots. If a community in the joint graph contains a single community in each corresponding snapshot, then they are matched. If the joint graph contains more than one

community from either snapshot, the communities are matched in descending order of their relative node overlap. Overlap is computed for every pair of communities from the two snapshots as the fraction of the number of common nodes to the sum of the number of nodes in both communities.

The work of Palla et al. [10] falls into the category of community matching approaches, i.e., methods with a static community detection step and a matching step. Most of the earliest algorithms proposed for dynamic community detection were following a similar approach, with variations in the method used for detection in each snapshot (MOSES in [19]), Louvain in [14], etc.) and for community matching (Jaccard Coefficient in [19], Core nodes in [20], etc).

In recent years, several authors have proposed methods based on a different approach, allowing to work on dynamic graphs provided as a stream. In this case, there are too many modifications of the network to run a complete algorithm at each step. Therefore, these methods update communities found at previous steps based on local rules. Below, we introduce examples of such methods. More details can be found in [12].

- Xie et al.[21] extended LabelRank [22] algorithm which is a stabilized and deterministic variant of Label propagation algorithm [23] to deal with evolving communities in dynamic networks. The extended algorithm called LabelRankT is based on a conditional update rule by which only nodes involved in change between two consecutive snapshots are updated.

- Nguyen et al.[24] proposed AFOCS, an adaptive framework for detecting, updating and tracing the evolution of overlapping communities in dynamic mobile networks. During the initialisation step, AFOCS identifies all possible basic network communities which represent the densely connected part of the network, whose internal density is greater than a certain level, and merge those with the highest overlaps with each other. In a second step, AFOCS adaptively update the community structure, as the dynamic network evolves in time.

- Cazabet and Amblard [25] proposed an online algorithm called iLCD.

In this work, the dynamic network is considered as a sequence of events (adding or removing edges). iLCD is using a multi-agent system: each community is an agent on the network, which can integrate or reject nodes. The agents are bounded by a certain number of operating rules, like updating existing communities, creating new communities or merging similar ones. Communities can be updated at each apparition or deletion of links.

- Rossetti et al.[26] defined TILES, which also proceeds in a streaming fashion, i.e., dynamics of the network is described as flows of interactions (also called perturbations) between users where nodes and edges can be created or removed over time. Each perturbation is considered as a fall of domino tile: every time a new interaction appears in the network, TILES updates the community locally and then propagates the changes to the node surroundings to adjust the neighbors' community memberships.

A weakness of these algorithms is the absence of any guarantee that the communities found represent an optimal solution at the global level, because communities at each step are based on communities found in a previous step by applying a set of local rules. More precisely, these methods suffer from the risk of community drift, in which the solution can be dragged away from an originally relevant solution. Another consequence is that communities found by these algorithms at step t depend on the particular sequence of previous graph modifications: the same graph produced by a different graph's history would yield a different partition.

On the contrary, due to the nature of the definition of communities in CPM, we are able in this article to provide an algorithm that handles a flow of changes with local modifications, while guaranteeing that the same state of the graph will always yield the same community structure.

3. Dynamic Network Model

Various temporal models have been proposed to deal with dynamic networks. We distinguish three broad approaches:

- **Aggregated graphs** model the dynamic network as a single static network by aggregating all contacts between each pair of nodes in a single edge. This representation does not allow longitudinal analysis, for instance tracking the evolution of communities.

- **Series of snapshots** model the evolving network through a series of snapshots, each of which is a static network representing contacts that exist at the corresponding time, or during the corresponding time window. The main issue of this approach is to determine the 'right' number of time windows, i.e., the temporal granularity. Tracking communities across network sequences can be difficult if important temporal information is lost between snapshots.

- **Temporal networks** conserve all known temporal information. There are two main models: series of contact and interval graph [27]. In a sequence of contact, interaction is represented as a triple (i, j, t) where i and j are the interacting entities and t is the time when the relationship is activated. In an interval graph, interaction is represented as a quadruplet $(i, j, t, \delta t)$ which means that i is involved in contact with j from t to δt. In these models, only the temporal information about interactions is represented, there is no temporal information about nodes.

In the following, we introduce our own formalism for evolving graphs, which is better suited to deal with *stream graphs*, i.e., graphs whose modifications occur as a flow, not necessarily known *a priori*. This formalism has the same expressivity as interval graphs.

3.1. Stream graph

Networks are often represented by a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges between nodes. We represent dynamic graphs as an ordered sequence of events, which can be node addition, node removal, edge addition or edge removal. We use the following notations:

- *Inserting or removing a node* is represented as triples (v, e, t), where v is the node, e is the event observed among $\{+, -\}$(insert $(+)$ or remove$(-)$), and t is the time when the event occurs.
- *Inserting or removing an edge* is represented as quadruplets (u, v, e, t), where u and v are endpoints of the edge, e is the event observed among $\{+, -\}$(insert $(+)$ or remove$(-)$), t is the time when the event occurs.

Note that this formalism, for edges, is identical in nature to an interval graph, but is more convenient for stream algorithms, as new operations can be added at the end of the ordered sequence of events without affecting previous ones.

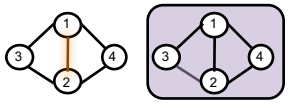
4. OLCPM Framework

Our framework comprises two main steps. First, we propose to adapt the classical algorithm CPM [13] for static overlapping community detection to deal with evolving networks. We propose an online version of CPM called OCPM (Online CPM). This algorithm is based on analyzing the dynamic behaviors of the network, which may arise from inserting or removing nodes or edges, i.e., every time a change is produced in the network, we update locally the community structure alongside the involved node or edge.

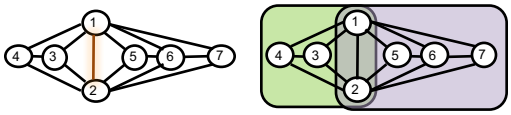
As stated earlier, CPM may not cover the whole network, i.e., some nodes have no community membership. To deal with this problem, we assume that the communities corresponding to OCMP contain core nodes, and we propose a way to discover the community peripheral nodes. In the second step of our framework, we extend OCMP using label propagation method and we propose OLCPM (Online Label propagation CPM). These proposals will be presented in detail in the next section.

4.1. OCPM: Online Clique Percolation Method

This section proposes the first step of our framework OLCPM, an online Clique Percolation Method (OCPM). This method takes two inputs:



(a) Example with k=3



(b) Example with k=4

Figure 1: Examples of adding an edge with both endpoints outside any community. (*a*) Example for $k = 3$: when the edge$(1, 2)$ is added, a new community $\{1, 2, 3, 4\}$ is created from two adjacent k-cliques $\{1, 2, 3\}$ and $\{1, 2, 4\}$. (*b*) Example for $k = 4$: the insertion of edge$(1, 2)$ leads to the creation of two communities $\{1, 2, 3, 4\}$ and $\{1, 2, 5, 6, 7\}$ from respectively two groups of not-adjacent k-cliques $\{\{1, 2, 3, 4\}\}$ and $\{\{1, 2, 5, 6\}$,$\{1, 2, 6, 7\}\}$

.

- SE, chronologically ordered sequence of events which models networks modification, following the format: (n, e, t) or (i, j, e, t) as defined in section 3.1
- the parameter K, which determines the clique size; it is an integer value greater than or equal to 3

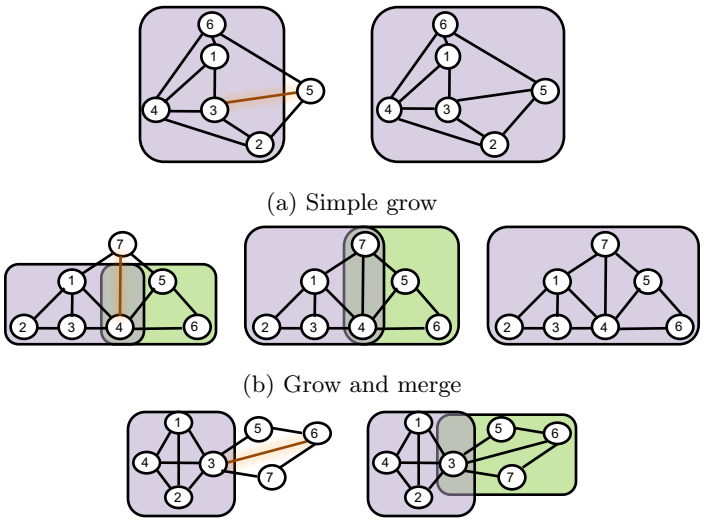
The OCPM method maintains after each modification three elements:

- $G(V, E)$ the current state of the network
- AC the set of currently alive communities
- DC the set of dead communities

It is therefore possible to know the community structure status at every network modification step.

4.1.1. Definition of the OCPM algorithm

Note: To facilitate the readability of the paper, we decided to put all formal algorithms in the **Appendix**, and to only include the rationale of these



(a) Simple grow

(b) Grow and merge

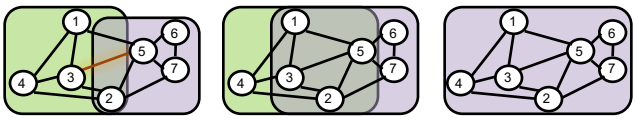
(c) New community

Figure 2: Example of adding an edge with an external endpoint and internal one(for $k = 3$). (a) The community $\{1, 2, 3, 4, 6\}$ grows with node 5 when adding edge $(3, 5)$. (b) When the edge $(4, 7)$ is added, the communities $\{1, 2, 3, 4\}$ and $\{4, 5, 6\}$ grow with node 7, and then merged. The resulting community takes the identity of the one that contains more nodes.(c) By adding edge $(3, 6)$, a new community $\{3, 5, 6, 7\}$ is created.

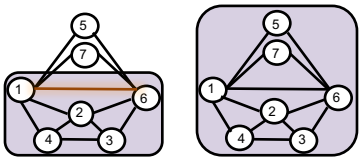
algorithms in the body of the article. Please refer to the **Appendix** for further details.

The core of the OCPM algorithm can be defined by an algorithm that updates the current state of all variables according to a sequence of events SE, as detailed in Algorithm 1. The task carried out by the algorithm depends on the type of event encountered:

- **Add a new node**: adding an isolated node n has no influence on the community partition. In this case, only n is added to the graph G and no other action is performed until the next event.
- **Add a new edge**: when a new edge (i, j) appears, we add this edge to the graph G. According to the type of edge, we distinguish two cases:



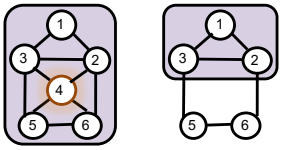
(a) Grow and Merge



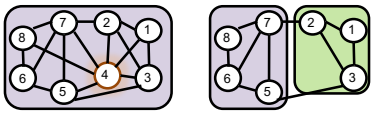
(b) Grow

Figure 3: Examples of adding an edge with two internal endpoints(k=3). (a) The communities $\{1, 2, 3, 4\}$ and $\{2, 5, 6, 7\}$ grow with the nodes of adjacent k-cliques $\{\{1, 3, 5\}, \{2, 3, 5\}\}$ formed when adding the edge $(3, 5)$, and then merged. (b) The community $\{1, 2, 3, 4, 6\}$ grows with the nodes of adjacent k-cliques $\{\{1, 7, 8\}, \{1, 5, 8\}, \{1, 2, 8\}\}$ formed when adding the edge $(3, 5)$.

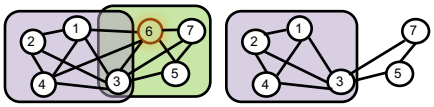
- When inserting an external edge, i.e., both its endpoints are outside any community, we check if one or more new k-cliques (KCliques() function Algorithm 6) are created. If it is the case, we gather all adjacent k-cliques one to the other. Then, for each group of adjacent k-cliques, we create a single community. Figure 1 shows two examples of adding external edges and the changes it brings to the community structure. (See Algorithm 3)
- In all other cases, i.e., when a new edge appears with one or two internal extremities, we check all k-cliques created when adding this edge and not belonging to any community. Then, all adjacent k-cliques are grouped together and for each group, we check if there are other adjacent k-cliques included in any community to which belongs any node in this group. If they exist, the corresponding communities will grow with the nodes of this group and they can eventually be merged (Merge()function Algorithm 7). Otherwise, a new community appears containing nodes of this group. Figures 2 and 3 depict some examples of adding edges with one or two internal endpoints and the changes to the community structure. (Algorithm



(a) Shrink



(b) Shrink and Split

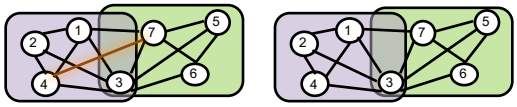


(c) Death

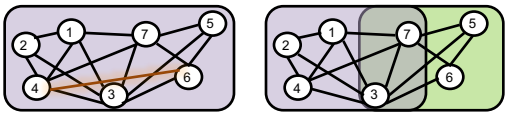
Figure 4: Example of removing internal node (k=3 for (a) and (b), k=4 for (c)). (a) When removing the node 4, the members $\{4, 5, 6\}$ leaves out the community $\{1, 2, 3, 4, 5, 6\}$.(b) When removing the node 4, the community $\{1, 2, 3, 4, 5, 6, 7, 8\}$ shrinks, i.e., it loses this node and all its edges, and then splits into two communities: $\{5, 6, 7, 8\}$ and $\{1, 2, 3\}$. (c)By removing the node 6, the community $\{1, 2, 3, 4\}$ shrinks and the community $\{3, 5, 6, 7\}$ dies

2)

- **Delete node**: In this case, we remove the node from the graph G, and all its edges are removed as well. If the node is external, i.e., it doesn't belong to any community, the community structure is not affected and no action is performed until the next event. When the removed node belongs to one or more communities, we check for each community to which this node belongs whether it still contains at least a k-clique after the node is removed. This community dies if it loses all k-cliques(see figure (c) 4). Otherwise, the community shrinks, i.e., it loses this node and all its associated edges. Here, we distinguish two cases:
  - The community may remain coherent and the community structure doesn't change(see figure (a) 4).
  - The community may become disconnected and therefore, it will be



(a) No change in the community structure



(b) Community split

Figure 5: Examples of removing internal edge (k=4). (a) The community structure doesn't change when removing the edge $(4, 7)$. (b) When removing the edge $(4, 6)$, the community splits into two small communities, each of which contains a group of adjacent k-cliques in the original community.

break up into small communities (see figure (b) 4).

The split function (Algorithm 8) deals with these two cases. After the community shrinking, its structure is recalculated keeping the principle of CPM -checking all maximal cliques of size not less than k. The resulting community having the largest number of nodes keeps the identity of the original one, where the others have new identities.

The Algorithm 4 describes this case.

- **Delete edge**: First, we remove the edge from the graph G. The removal of an edge with two endpoints belonging to the same community(ies)(called internal edge) follows the same mechanism as internal node removal: the communities to which belong the two extremities of this edge may split or die. For each of them, we check whether it still contains k-cliques. If so, we use the function Split (Algorithm 8) to check whether or not the community is divided into smaller parts. Otherwise, this community dies (see Algorithm 5). Figure 5 shows two examples of removing internal Edge and the changes that it brings to the community structure.

  For all other types of edges, the community structure doesn't change.

Here, we detail some functions used in our algorithm:

- **Kcliques**(): (Algorithm 6) This function takes a set of nodes SN as input parameter and returns all maximal cliques of size not less than k containing this set. In order to optimize the performance of our algorithm, k-cliques are locally launched in the subgraph including the set SN and all common neighbors among its members.

- **Merge**(): (Algorithm 7) This function is used for merging adjacent communities. The resulting community takes the identity of the one with the highest number of nodes.

- **Split**(): (Algorithm 8) This function is used for splitting a community if possible. It takes as input a community and creates from it one or more communities. We proceed as follows: first, we identify all maximal cliques of size not less than k in this community and we aggregate adjacent k-cliques with each other. Then, for each of the aggregated k-cliques, we create a new community. The community which has the largest number of nodes keeps the identity of the original one.

Table 1 summarizes the actions which can be carried out by OCPM according to graph events.

Event		**Actions**
Add new node		-
Add new edge	External	Birth
	Other	Grow+[Merge], Birth
Delete Node	External	-
	Internal	Shrink+[Split], Death
Delete Edge	Internal	Split, Death
	Other	-

Table 1: Actions that can be performed according to graph events. Brackets denotes events that can only follow the preceding community event.

4.1.2. Complexity of the algorithm

Instead of computing all k-cliques for the whole network at each event occurring in the network, OCPM updates the community structure on the local scale, and thus only the community structure alongside the node or the edge involved in the event is recomputed. For certain events, like adding or deleting an isolated node or deleting an external edge, the community structure doesn't change and hence, the computational time saving reaches its maximum. For instance, if we have n k-cliques when such event is produced, the computational time savings will be n times the average time for calculating k-cliques. For other events, the computational time saving is also significant. See section 5.1 for an empirical evaluation of the complexity.

4.1.3. Community tracking process

One of the difficulties when tracking the evolution of communities is to decide which community is a continuation of which. Our framework allows a trivial matching in the case of *continuation* (no merge or split) of communities. In the case of merge and split, deciding which community keeps the original identity is a well-known problem with no consensus in the literature [12]. In OCPM, we took the simple yet reasonable decision to consider that the *largest* community involved in a merge or split have the same identifier as the merged/split one. This strategy can be replaced without altering the algorithm logic. A more advanced process could be added to solve problems of *instability*, e.g. communities merging and quickly splitting back to their original state.

4.2. OLCPM: Online Label propagation CPM

This section describes the second step of our framework. A post-processing based on label propagation is set out on the output communities of OCPM to discover the peripheral nodes. This module is called OLCPM (Online Label propagation CPM).

There is a twofold reason for using a post-process extending core-communities found by OCPM:

- In a network evolving at fast path, one can update core-communities efficiently after each event, and run the post-process only when the current state of communities needs to be known, thus saving computation time
- It is known that the periphery of communities is often not well defined and unstable. As seen earlier, and because OCPM is deterministic and it searches for core-communityies, it reduces this instability problem. By using the label propagation mechanism only as a post-process for analysis, communities at t do not depend on the periphery of communities that might have been computed at $t-1$, but only on the stable part found by OCPM.

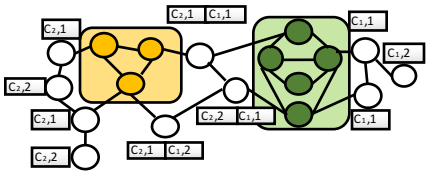
4.2.1. OLCPM algorithm

First, each core-community (community found by OCPM) spreads to neighboring peripheral nodes (nodes not covered by OCPM) a label containing its identity and a weight representing the geodesic distance (the length of the shortest path) between this neighboring node and any other node in the core-community. Each peripheral node has a local memory allowing the storage of many labels. Label propagation process is based on breadth-first search (BFS). When all labels have been shared, nodes are associated with all communities with which they have the shortest geodesic distance. Note that nodes can, therefore, belong to several communities, if they are at the same distance of community found by OCPM. This algorithm is defined formally in Algorithm 9.

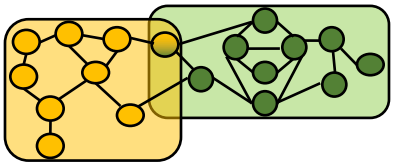
Figure 6 presents an illustration of this process.

5. Experiments

In this section, we begin by evaluating the effectiveness of OCPM algorithm. Thus, we compare the time complexity of OCPM with the dynamic version of CPM [10]. Second, we are interested in the quality of the communities that OLCPM is able to find, considering both synthetic and real-world networks.



(a) Label spreading step



(b) Community structure after label analysis (k=3)

Figure 6: Peripheral community updates by OLCPM. (a) Label spreading step. (b) Community structure after label analyses (for K=3). Green nodes are members of the community $C1$; Yellow nodes are members of the community $C2$; uncolored nodes have no affiliation.

5.1. Measuring OCPM complexity gain for highly dynamic networks

In this section, we compare the empirical complexity of the original dynamic version of CPM (hereafter, DyCPM)[10] and our proposed version (OCPM). We generate synthetic dynamic networks, and compare how the running time of both algorithms vary with the properties of the network and of its dynamic. Note that we compare OCPM only with CPM because both algorithms try to solve the *same problem*, i.e, they have the same definition of communities. Other streaming algorithms introduced in section 2 have an *ad hoc* definition of communities introduced together with the method, and does not have the same properties, such as being deterministic and not being dependent on the network history. Their complexity is, in theory, similar to the one of OCPM (local updates at each modification).

5.1.1. Generation of dynamic networks with community structure

We propose a simple process to generate dynamic networks with realistic community structure. First, a static network is generated using the LFR benchmark [28], the most used benchmark for community detection. Then, for this

network, we generate a step by step evolution. In order to conserve the network properties (community structure, size, density), we define an *atomic modification* as the following process:

1. Choose randomly a planted community as provided by LFR
2. Select an existing edge in this community
3. Select a pair of nodes without edges in this community
4. Replace the selected existing edge by the selected not-existing one.

We define a step of evolution as the combination of a atomic modifications. In order to test the influence of the number of modifications between steps, we test different values of a.

Note that we use synthetic networks instead of real networks at this step since:

- We are only interested in measuring time complexity of algorithms. Synthetic networks are mostly criticized for having unrealistic community structures, while here we are mainly interested in the size and rate of evolution of the networks.
- It allows controlled experiments. With real evolving networks, changes in the structure/size of the network could affect computation time at each step, and we could not control the number of modifications between snapshots, or vary the size of networks while keeping constant properties.

5.1.2. Experimental process

The LFR benchmark [28] is, as of today, the most widely used benchmark to evaluate community detection methods. It is known to generate realistic networks with heterogeneous degrees and community sizes.

It has the following parameters : N is the network size, k is the average degree of nodes, $kmax$ the maximum degree, $t1$ and $t2$ are power-law distribution coefficients for the degree of nodes and the size of community respectively, μ is the mixing parameter which represents the ratio between the external degree of

the node with respect to its community and the total degree of the node, $minc$ and $maxc$ are the minimum and maximum community size respectively, On is the number of overlapping nodes , Om is the number of community memberships of each overlapping node.

In order to obtain realistic networks, we first generate an original network with n nodes using the LFR benchmark, with fix parameters $k = 7$, $maxk = 15$, and $\mu = 0.4$. Other parameters stay at their default values. In order to test the influence of the network size, we test different values of n.

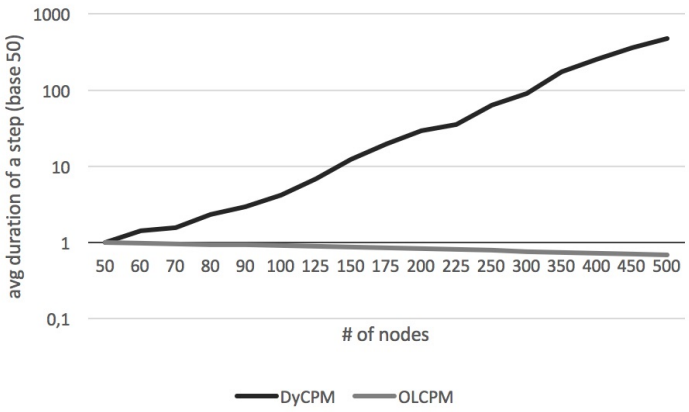


Figure 7: Evolution of time complexity when varying the size of the network (number of nodes), and keeping other parameters constant (average node degree, community, size, etc.). DyCPM complexity increases exponentially with the size of the network, while OLCPM one stays constant or slightly decreases. Expressed in base 50, i.e, 10 on the vertical axis means 10 times slower than with 50 nodes.

As can be seen in figures 7 and 8, the complexity of both algorithms depends on very different parameters. With OLCPM, the time needed to update communities after a modification step does not increase proportionally to the size of the network at any given time, but increases linearly with the number of atomic modifications.

On the contrary, the complexity of DyCPM depends on the properties of the static network, but not on the number of atomic modifications between steps.

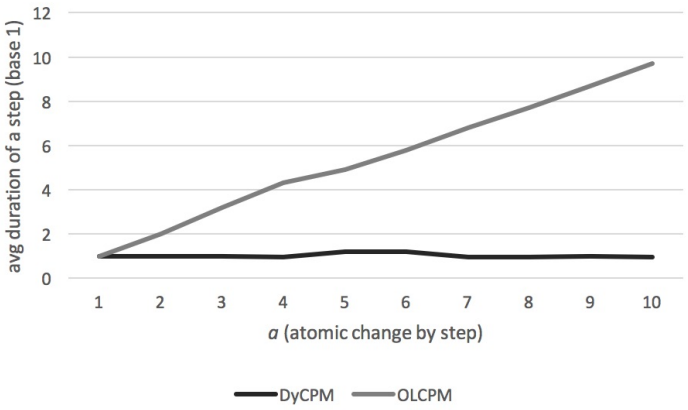


Figure 8: Evolution of time complexity when varying the number of atomic changes by step. DyCPM complexity is independent relatively to a while OLCPM's complexity increases linearly with a Time.

As expected, OLCPM is appropriate to deal with stream graphs, in which modifications are known at a fine granularity, as the cost of each update is low. On the contrary, DyCPM is appropriate to deal with network snapshots, i.e., a dynamic network composed of a few observations collected at regular intervals.

5.2. Measuring OLCPM communities quality

To quantify the quality of communities detected by OLCPM framework, we used both synthetic and real-world networks with ground truth community structure. We remind the reader that communities found by DyCPM and OCPM are identical, the difference lies only in the label propagation post-process of OLCPM.

Normalized Mutual Information (NMI) is used as the measurement criterion. This measure is borrowed from information theory [29] and widely adopted for evaluating community detection algorithms. It measures the similarity between a ground truth partition and the one delivered by an algorithm. As the original definition is only well defined for *partitions* (each node belong to one and only one community), a variant of the NMI adapted for *covers* (nodes can belong

to zero, one or more communities) have been introduced in [30]. This variant is the most used in the literature for comparing overlapping communities. We used the original implementation by the authors [1]. The NMI value is defined between 0 and 1, with a higher value meaning higher similarity.

5.2.1. Static Synthetic networks

We use the LFR benchmark [28] to generate realistic artificial networks.

We use two different network sizes, *small networks*(1000 nodes) and *large networks*(5000 nodes), and for a given size we use two ranges for community size: *small communities*, having between 10 and 50 nodes and *large communities*, having between 20 and 100 nodes. We generate eight groups of LFR networks.

In the first four networks, μ ranges from 0 to 0.5 (steps of 0.1) while Om is set to 100 for small networks and 500 for large networks (5000 nodes). In the other networks, μ is fixed to 0.1 and On ranges from 0 to 500 (steps of 100) for small networks and from 0 to 2000 (steps of 500) for large networks. All these networks share the common parameters: $k = 10$, $maxk = 30$, $t1 = 2$, $t2 = 1$, $On = 2$. The parameter settings are shown in table 2.

Network group ID	**N**	**minc**	**maxc**	μ	**On**
N1	1000	10	50	0-0.5	100
N2	1000	20	100	0-0.5	100
N3	5000	10	50	0-0.5	500
N4	5000	20	100	0-0.5	500
N5	1000	10	50	0.1	0-500
N6	1000	20	100	0.1	0-500
N7	5000	10	50	0.1	0-2000
N8	5000	20	100	0.1	0-2000

Table 2: LFR parameter setting

[1] `https://sites.google.com/site/andrealancichinetti/software`

CPM and OLCPM are run for $k = 4$. The NMI values of communities detected by CPM and OLCPM are depicted in figure 9. Note that communities found by CPM and OCPM are identical, therefore the observed differences are only due to the post process.

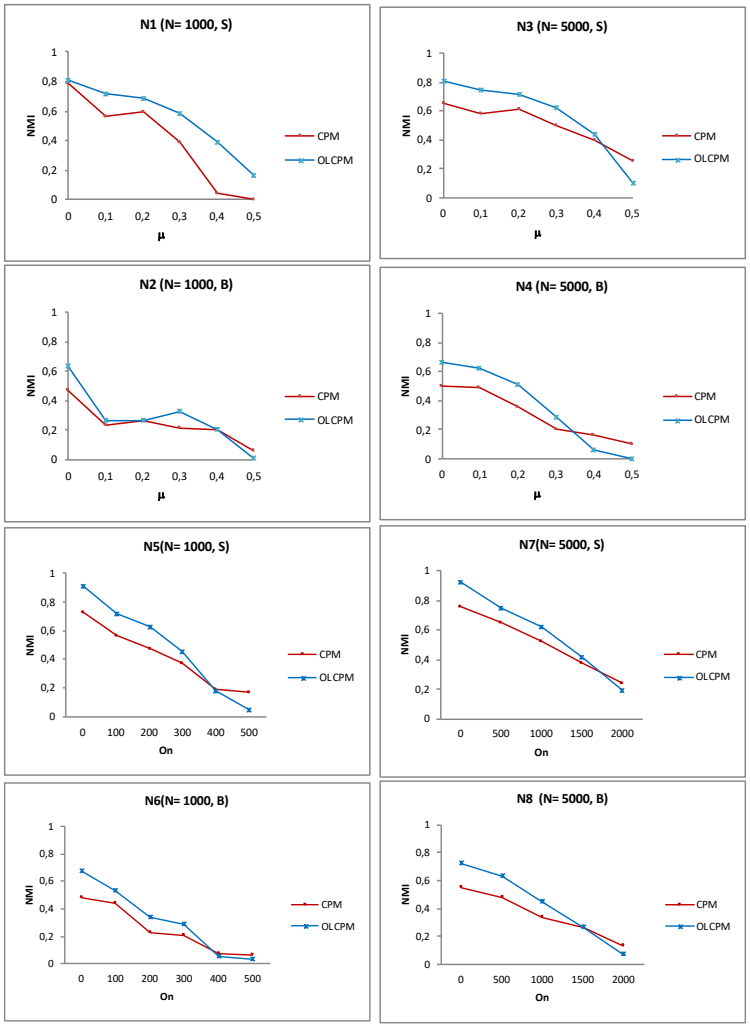


Figure 9: Performance of CPM and OLPM for $k = 4$ on the LFR benchmark networks. The plots show the NMI scores as a function of the mixing parameter μ (upper half plots) and of the number of overlapping nodes On (lower half plots) for different network sizes (small networks in the left hand plots and large networks in the right hand plots) and different community sizes ((S) ranges from 10 to 50 and (B) ranges from 20 to 100).

In most cases, OLCPM achieves the highest results, except for the two cases where: (1) the community structure becomes very fuzzy ($On >= 400$ for small networks or $On >= 1500$ for large networks) or (2) the value of μ is large (greater than 0.3). In these cases, OLCPM performs similar or slightly worse than CPM. When the community structure becomes too fuzzy for CPM, the irrelevant core-communities provided are probably worsened by the post-process.

As a conclusion, we can consider that in situations in which CPM finds meaningful communities in a network, the proposed post-process improves the solution.

5.2.2. Dynamic Real-world networks

In order to evaluate the community detection results of our framework OLCPM on real temporal networks, we leverage a high-resolution time-varying network describing contact patterns among high school students in Marseilles, France [31]. The dataset was collected by the SocioPatterns collaboration using wearable sensors, able to capture proximity between individuals wearing them. The dataset was gathered during nine days (Monday to Tuesday) in November 2012. Data collection involved 180 students from five classes. Proximity relations are detected over 20-second intervals. Data collection involved students' classes corresponding to different specializations: 'MP' classes focus more on mathematics and physics, 'PC' classes on physics and chemistry, and 'PSI' classes on engineering studies. These classes represent the expected ground truth community structure.

We construct a dynamic network composed of 216 snapshots, each corresponding to 1 hour of data. Nodes correspond to students, and there is an edge between two nodes in a snapshot if the corresponding students have been observed in interaction at least once during the corresponding period. (Please refer to the original article [31] for details about the meaning of *interaction*. To sum up, two students are in interaction if they stand face-to-face at a distance between 1 and 1.5 meters.)

We compute the communities at each step using both DyCPM and OLCPM

(Communities yielded by DyCPM and OCPM are identical). Then, for each snapshot, we compute the NMI according to [30]. Results are displayed in Figure 10. We show results for k=3 and k=4, which yield the best results.

The average NMI over all snapshots is provided in Table 3.

Algorithm	DyCPM k=3	DyCPM k=4	OLCPM k=3	OLCPM k=4
Average NMI	0.024	0.004	0.059	0.044

Table 3: Average NMI scores of OLCPM and DyCPM [10] for $k = 3$ and $k = 4$ on SocioPatterns collaboration networks [31].

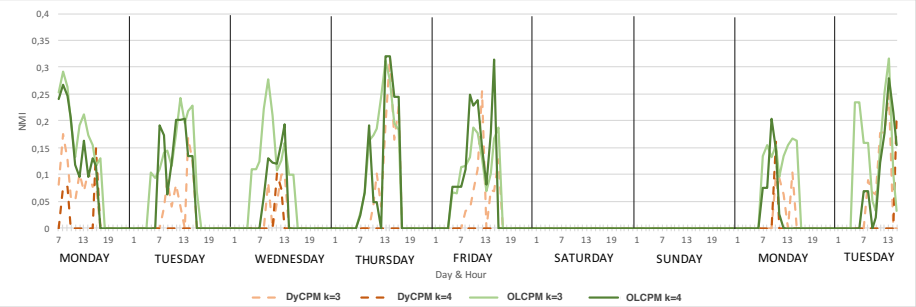


Figure 10: NMI values of OLCPM and CPM [13] for $k = 3$ and $k = 4$ in on SocioPatterns collaboration networks [31].

We can observe that the average NMI of OLCPM is higher than the original DyCPM, and that values of NMI are also higher for most snapshots.

The longitudinal visualization of Figure 10 illustrates the relevance of studying the evolution of a network with a fine granularity: only looking at this plot, we can see that the class structure is not always present in the data. For instance, we can observe that there is no community structure during evenings and weekends, or that the community structure is less observable during several days around lunchtime (Thursday, Friday, second Monday). One can then look in more details to the communities found and their evolution to interpret these observations. In this example, we were able to run DyCPM because of the small

size of the network, the restriction to one-hour interval, and the limitation to 9 days of data, but, as shown previously, it would not be possible to extend this analysis to a much larger number of steps due to the increase in complexity.

6. Conclusion

In this paper, we proposed OLCPM framework to discover overlapping and evolving communities in social networks. We proposed OCPM, an online version of CPM [13], working on a fully dynamic network model, i.e., described as flows of events, where nodes or edges can be added or removed over time. Instead of calculating all k-cliques for the whole network at each event occurring in the network, our method updates only the community structure alongside the node or the edge involved in the event. This local update of the community structure provides a significant improvement in computational time.

To cope with the covering problem of CPM, nodes belonging to OCPM communities are considered as core nodes and we proposed a post-process based on label propagation to discover peripheral nodes.

The experimental results of our framework in both artificial and real-world networks show good performance in both computing time and quality detection.

Our method has some drawbacks, some of which are related to CPM itself, like the dependency of the parameter k (clique size). We intend to propose a heuristic for finding appropriate values of k.

Currently, the post-process is run from scratch at each step, and although it is not as costly as a clique-finding problem, running it at each step for a large network can become very costly. For future research, it would be interesting to extend OLCPM by developing an online version of the post-process.

References

[1] J. Moreno, Who shall survive? A new approach to the problem of human interrelations., Nervous and Mental Disease Publishing Company, Washington, 1934.

[2] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks, Proceedings of the National Academy of Sciences 101 (9) (2004) 2658.
URL http://scholar.google.de/scholar.bib?q=info:Yu5P1ZhsmNUJ:scholar.google.com/&output=citation&hl=de&as_sdt=2000&ct=citation&cd=0

[3] L. Boratto, S. Carta, A. Chessa, M. Agelli, M. L. Clemente, Group recommendation with automatic identification of users communities, in: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03, WI-IAT '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 547–550. doi:10.1109/WI-IAT.2009.346.
URL http://dx.doi.org/10.1109/WI-IAT.2009.346

[4] W. Deng, R. Patil, L. Najjar, Y. Shi, Z. Chen, Incorporating community detection and clustering techniques into collaborative filtering model, in: Proceedings of the Second International Conference on Information Technology and Quantitative Management, ITQM 2014, National Research University Higher School of Economics (HSE), Moscow, Russia, June 3-5, 2014, 2014, pp. 66–74. doi:10.1016/j.procs.2014.05.246.
URL https://doi.org/10.1016/j.procs.2014.05.246

[5] F. Moradi, T. Olovsson, P. Tsigas, An evaluation of community detection algorithms on large-scale email traffic, in: Experimental Algorithms - 11th International Symposium, SEA 2012, Bordeaux, France, June 7-9, 2012. Proceedings, 2012, pp. 283–294. doi:10.1007/978-3-642-30850-5_25.
URL https://doi.org/10.1007/978-3-642-30850-5_25

[6] S. Kitchovitch, P. Lió, Community Structure in Social Networks: Applications for Epidemiological Modelling, PloS one 6 (7) (2011) e22220. doi:10.1371/journal.pone.0022220.

[7] E. Ferrara, P. D. Meo, S. Catanese, G. Fiumara, Detecting criminal organizations in mobile phone networks, CoRR abs/1404.1295.
URL http://arxiv.org/abs/1404.1295

[8] D. McKenzie-Mohr, W. Smith, Fostering Sustainable Behavior: An Introduction to Community-based Social Marketing, Education for sustainability, New Society Publishers, 1999.
URL https://books.google.dz/books?id=2ZnKy6BMpTQC

[9] D. Fenn, M. Porter, M. McDonald, S. Williams, N. Johnson, N. Jones, Dynamic communities in multichannel data: An application to the foreign exchange market during the 2007–2008 credit crisis, Chaos 19 (2009) 033119–8.
URL http://dx.doi.org/10.1063/1.3184538

[10] G. Palla, A.-L. Barabasi, T. Vicsek, Quantifying social group evolution, Nature 446 (7136) (2007) 664–667. doi:10.1038/nature05670.
URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=pubmed&dopt=Abstract&list_uids=17410175

[11] Q. Wang, E. Fleury, T. Aynaud, J.-L. Guillaume, Communities in evolving networks: definitions, detection and analysis techniques, in: Dynamics of Time Varying Networks, Ganguly, Mukherjee, Mitra, Peruani, Choudhury, 2013.

[12] R. Cazabet, F. Amblard, Dynamic community detection, in: Encyclopedia of Social Network Analysis and Mining, Springer New York, 2014, pp. 404–414.

[13] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (7043) (2005) 814–818.
URL http://dx.doi.org/10.1038/nature03607

[14] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of Statistical Mechanics: Theory and Experiment 2008 (10) (2008) P10008.
URL http://stacks.iop.org/1742-5468/2008/i=10/a=P10008

[15] M. Rosvall, C. T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proceedings of the National Academy of Sciences 105 (4) (2008) 1118.

[16] M. Girvan, M. E. J. Newman, Community structure in social and biological networks, PNAS 99 (12) (2002) 7821–7826.

[17] S. Fortunato, M. Barthelemy, Resolution limit in community detection, Proceedings of the National Academy of Sciences 104 (1) (2007) 36–41.

[18] T. Aynaud, J.-L. Guillaume, Static community detection algorithms for evolving networks, in: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2010 Proceedings of the 8th International Symposium on, IEEE, 2010, pp. 513–519.

[19] D. Greene, D. Doyle, P. Cunningham, Tracking the evolution of communities in dynamic social networks, in: Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining, ASONAM '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 176–183. doi:10.1109/ASONAM.2010.17.
URL http://dx.doi.org/10.1109/ASONAM.2010.17

[20] Y. Wang, B. Wu, X. Pei, Commtracker: A core-based algorithm of tracking community evolution, Advanced Data Mining and Applications (2008) 229–240.

[21] J. Xie, M. Chen, B. K. Szymanski, Labelrankt: Incremental community detection in dynamic networks via label propagation, CoRR abs/1305.2006.
URL http://dblp.uni-trier.de/db/journals/corr/corr1305.html#abs-1305-2006

[22] J. Xie, B. K. Szymanski, Labelrank: A stabilized label propagation algorithm for community detection in networks, CoRR abs/1303.0868.
URL http://arxiv.org/abs/1303.0868

[23] J. Xie, B. K. Szymanski, Community detection using A neighborhood strength driven label propagation algorithm, CoRR abs/1105.3264.
URL http://arxiv.org/abs/1105.3264

[24] N. P. Nguyen, T. N. Dinh, S. Tokala, M. T. Thai, Overlapping communities in dynamic networks: Their detection and mobile applications, in: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, MobiCom '11, ACM, New York, NY, USA, 2011, pp. 85–96. doi:10.1145/2030613.2030624.
URL http://doi.acm.org/10.1145/2030613.2030624

[25] R. Cazabet, F. Amblard, Simulate to detect: A multi-agent system for community detection, in: Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2011, Campus Scientifique de la Doua, Lyon, France, August 22-27, 2011, 2011, pp. 402–408. doi:10.1109/WI-IAT.2011.50.
URL https://doi.org/10.1109/WI-IAT.2011.50

[26] G. Rossetti, L. Pappalardo, D. Pedreschi, F. Giannotti, Tiles: an online algorithm for community discovery in dynamic social networks, Machine Learning (2016) 1–29doi:10.1007/s10994-016-5582-8.
URL http://dx.doi.org/10.1007/s10994-016-5582-8

[27] P. Holme, J. Saramäki, Temporal networks, Physics Reports 519 (3) (2012) 97–125.

[28] A. Lancichinetti, S. Fortunato, Community detection algorithms: a comparative analysis, arXiv e-print 0908.1062, physical Review E 80, 056117 (2009) (Aug. 2009).
URL http://arxiv.org/abs/0908.1062

[29] L. Danon, A. Díaz-Guilera, J. Duch, A. Arenas, Comparing community structure identification, Journal of Statistical Mechanics: Theory and Experiment 2005 (2005) P09008.

[30] A. Lancichinetti, S. Fortunato, J. Kertesz, Detecting the overlapping and hierarchical community structure of complex networks, ArXiv E-printsarXiv:0802.1218.

[31] J. Fournet, A. Barrat, Contact patterns among high school students, PLoS ONE 9 (9) (2014) e107878. doi:10.1371/journal.pone.0107878.
URL http://dx.doi.org/10.1371%2Fjournal.pone.0107878

Appendices

This Appendix contains the algorithms defining the OCPM and OLCPM methods.

input : K, G, AC, DC, SE

output: Update AC, DC, G

```
for ev ∈ SE do
    switch e do
        case Add Node do
            V ← V ∪ {n};
            break;
        end
        case Add Edge do
            E ← E ∪ {(i, j)};
            if (C_i ≠ ∅)or(C_j ≠ ∅) then
                AddNonExternalEdge(i, j, t, AC, DC, G);
            else
                AddExtrenalEdge(i, j, t, AC, G);
            end
            break;
        end
        case Remove Node do
            V ← V\{n};
            E ← E\{∀(i, j), i = n or j = n};
            if (C_n ≠ ∅) then
                RemoveInternalNode(n, t, AC, DC, G);
            end
            break;
        end
        case Remove Edge do
            E ← E\{(i, j)};
            if (C_i ∩ C_j ≠ ∅) then
                RemoveIntrnalEdge(i, j, t, AC, DC, G);
            end
            break;
        end
    end
end
```

Algorithm 1: Online Clique Percollation Method (OCPM)

input : i, j, t, AC, DC, G

output: Update AC, DC

$KC \leftarrow KCliques(\{i, j\}, G)$;
$KC\prime \leftarrow \{kcl \in KC, \forall cm \in AC, kcl \nsubseteq cm\}$;
$AKC \leftarrow AdjKCliques(KC\prime)$;
for $c \in AKC$ **do**
    $C_c \leftarrow \{\}$;
    for $e \in c$ **do**
        $C_c \leftarrow C_c \cup \{C_e\}$;
    end
    $AdjC \leftarrow \{\}$;
    for $cm \in C_c$ **do**
        if $|cm \cap c| \geqslant K - 1$ **then**
            $AdjC \leftarrow AdjC \cup \{cm\}$;
        end
    end
    if $AdjC \neq \varnothing$ **then**
        for $cm \in AdjC$ **do**
            $Growth(cm, c, AC)$;
        end
        if $|AdjC| > 1$ **then**
            $Merge(AdjC, t, AC, DC)$;
        end
    else
        $Birth(c, t, AC)$;
    end
end

Algorithm 2: Add Non-External Edge

input : i, j, t, AC, G

output: Update AC

$KC \leftarrow KCliques(\{i, j\}, G)$;
$AKC \leftarrow AdjKCliques(KC)$;
for $c \in AKC$ **do**
    $Birth(c, t, AC)$;
end

Algorithm 3: Add External Edge

input : n, t, AC, DC, G

output: Update AC, DC

for $c \in C_n$ **do**
    $KC \leftarrow KCliques(c, G)$;
    if $KC = \varnothing$ **then**
        $Death(c, t, AC, DC)$;
    else
        $Shrink(c, n, AC)$;
        $Split(c, t, AC, G)$;
    end
end

Algorithm 4: Remove Internal Node

input : i, j, t, AC, DC, G

output: Update AC, DC

for $c \in C_{ij} = \{\forall cm, cm \in (C_i \cap C_j)\}$ **do**
  $KC \leftarrow KCliques(c, G)$;
  if $KC = \varnothing$ **then**
    $Death(c, t, AC, DC)$;
  else
    $Split(c, t, AC, G)$;
  end
end

Algorithm 5: Remove Internal edge

input : SN:Set of nodes,G

output: SKC: Set of Set of nodes

for $n \in SN$ **do**
  $N \leftarrow Neighbors(n, G)$;
  $L \leftarrow L \cup \{N\}$;
end
$CN \leftarrow \cap_{l \in L} l$;
$SCL \leftarrow MaximalCliques(CN, K)$;

Algorithm 6: KCliques

input : $Adjc, t, AC, DC$

output: UpdateAC, DC

$mc \leftarrow c, |c| = max_{x \in Adjc}|x|$;
$Adjc \leftarrow Adjc \backslash \{mc\}$;
$mc \leftarrow \cup_{x \in Adjc}\ x$;
for $x \in Adjc$ **do**
    $Death(c, t, AC, DC)$;
end

Algorithm 7: Merge

input : c, t, AC, G

output: UpdateAC

$KCc \leftarrow KCliques(c, G)$;
$Adjc \leftarrow AKCliques(KCc)$;
$c \leftarrow mc, |mc| = max_{x \in Adjc}|x|$;
$Adjc \leftarrow Adjc \backslash \{c\}$;
for $cm \in Adjc$ **do**
    $Birth(cm, t, AC)$;
end

Algorithm 8: Split

input : AC, G

output: UpdateAC

$PN \leftarrow \{n, n \in c \forall c \in AC\}$;
//Label spreading
for $c \in AC$ **do**
    $d \leftarrow 1$;
    $S \leftarrow c$;
    x:$N \leftarrow Nieghbors(S)$;
    $N \leftarrow N \cap PN$;
    if $N \neq \varnothing$ **then**
        for $n \in N$ **do**
            $Label(n, c.id, d)$;
        end
        $S \leftarrow N$;
        $d \leftarrow d + 1$;
        goto x:
    end
end
//label Analyses
for $n \in PN$ **do**
    $idc \leftarrow LabelAnalysis(n.label)$;
    $Growth(idc, n)$;
end

Algorithm 9: OLCPM