Mach Learn
DOI 10.1007/s10994-016-5582-8

Tiles: an online algorithm for community discovery in dynamic social networks

Giulio Rossetti[1,2] · **Luca Pappalardo**[1,2] · **Dino Pedreschi**[1] · **Fosca Giannotti**[2]

Received: 12 March 2015 / Accepted: 5 July 2016
© The Author(s) 2016. This article is published with open access at Springerlink.com

Abstract Community discovery has emerged during the last decade as one of the most challenging problems in social network analysis. Many algorithms have been proposed to find communities on static networks, i.e. networks which do not change in time. However, social networks are dynamic realities (e.g. call graphs, online social networks): in such scenarios static community discovery fails to identify a partition of the graph that is semantically consistent with the temporal information expressed by the data. In this work we propose TILES, an algorithm that extracts overlapping communities and tracks their evolution in time following an online iterative procedure. Our algorithm operates following a domino effect strategy, dynamically recomputing nodes community memberships whenever a new interaction takes place. We compare TILES with state-of-the-art community detection algorithms on both synthetic and real world networks having annotated community structure: our experiments show that the proposed approach is able to guarantee lower execution times and better correspondence with the ground truth communities than its competitors. Moreover, we illustrate the specifics of the proposed approach by discussing the properties of identified communities it is able to identify.

Keywords Community discovery · Dynamic networks · Social network analysis

Guest Editors: Céline Rouveirol, Rushed Kanawati, and Ruggero G. Pensa.

✉ Giulio Rossetti
giulio.rossetti@di.unipi.it; giulio.rosetti@isti.cnr.it

Luca Pappalardo
lpappalardo@di.unipi.it

Dino Pedreschi
pedre@di.unipi.it

Fosca Giannotti
fosca.giannotti@isti.cnr.it

1 KDD Lab, University of Pisa, Pisa, Italy

2 KDD Lab, ISTI-CNR, Pisa, Italy

Published online: 08 September 2016

1 Introduction

Community Discovery has become during the last decade one of the most challenging and studied problems in social network analysis due to its relevance for a wide range of applications: from the study of information and disease spreading (Wu and Liu 2008; Bhat and Abulaish 2013) to the compression of networked data (Buehrer and Chellapilla 2008), the prediction of future interactions and activities of individuals (Rossetti et al. 2015a, b), and even the analysis of the patterns of human mobility (Rinzivillo et al. 2012; Bagrow and Lin 2012. Even though several definitions of network communities exist, the common sense depict such mesoscale substructures as sets of nodes that are closer or more similar to each other than to anybody else outside the community. Several community discovery algorithms have been proposed to deal with *static* networks, i.e. networks which do not change their topology in time (Fortunato 2010; Coscia et al. 2011). These algorithms work relying on the so-called quasi-steady state assumption (QSSA): they model networks as "frozen in time" by leveraging the observation that mutations in their topology happen only in the long run. This assumption simplifies the algorithmic design but often leads to biased results when analyzing peculiar network typology, such as social networks, which are by their nature highly *dynamic*: indeed, online interaction networks, call graphs, buyer-seller transactions are all examples of rapidly and continuously changing systems for which a QSSA does not apply. To clarify this point, consider the network scenario depicted in Fig. 1: a partition extracted by a static community discovery algorithm, that usually looks only at the final state of the network (e.g., $t = 5$ in this example), without taking into account the temporal ordering of interactions, it can group together nodes that have been in contact rarely and whose interactions can be very distant in time. Conversely, dynamic approaches can propose multiple time-aware partitions following different criteria: for instance a dynamic algorithm that search for Δ-communities (i.e., set of nodes connected by edges whose timestamp differ for at most Δ) can identify multiple, possibly overlapping, sub-structures such as, assuming $\Delta = 1$, {u,x,y}, {u,v,y,j} and {v,z,j}. Following an alternative view, an online approach can smooth the community evolution by identifying incrementally the boundaries each sub-structure has as the network evolves through time.

In dynamic networks, the rise of new nodes and edges produces deep topological mutations and creates new paths connecting once disconnected components. Therefore, an algorithm that considers social networks as static entities—frozen in time—necessarily introduces bias on its results. For these reasons, we advocate the need to weaken the QSSA and imagine social

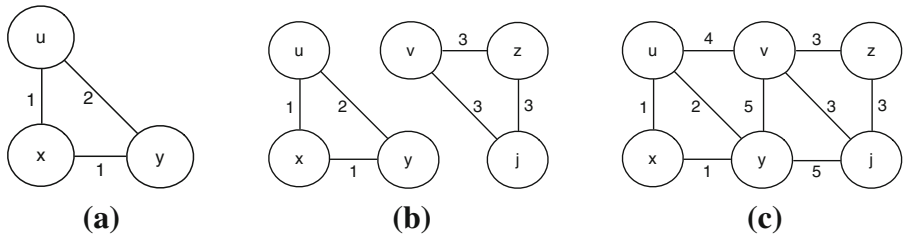

Fig. 1 Communities identified by a dynamic community discovery algorithm. *Numbers* on the edges represent interaction times. A static algorithm, working on the final graph (**c**), identifies just one community, since it does not take into account the network evolution. In contrast, looking at the same network through a dynamic perspective we are able to unveil different communities describing different stages of its evolution: at time $t = 2$ we have the community $C_0 = \{u, x, y\}$ (**a**); at time $t = 3$ a new community $C_1 = \{v, z, j\}$ appears (**b**) and at time $t = 5$ all nodes are part of a single community (**c**)

networks as complex, mutable, evolving objects which change in an *fluid* manner every time a new interaction appears (or disappears). In this scenario, pursuing an online community discovery approach enables valuable complementary benefits such as: (1) the reduction of computational complexity (both in space and time), (2) the tracking of community dynamics, (3) the possibility to feed predictive models with punctual and fine grained information regarding how the network topology changes over time.

Nonetheless, a dynamic approach to community discovery enables interesting practical applications. For instance, time-aware approaches can be used by mobile phone carriers that want to propose a flexible billing plan for their customers by lowering call prices to users in the same social circle: indeed, when imposing a fixed network structure such marketing strategy looses its effectiveness since static communities overestimate or underestimate the real connectivity (Fig. 1). In contrast, a dynamic community discovery algorithm provides up-to-date communities and helps the company in providing its users with a more customized service.

In this work, we propose to adapt the classical community discovery problem to the dynamic scenario and introduce an evolutionary formulation able to deal with evolving networks. This dynamic perspective on the community discovery problem allows us to investigate, describe and quantify relevant processes that take place on social networks, such as the evolution through time of the network community structure, the evolution through time of each single community both in terms of topology and events (birth, growth, death etc.) and even the evolution of single individuals connections within different communities. Moreover, we propose TILES,[1] an algorithm that tracks the evolution of communities through time. Our approach proceeds in a streaming fashion considering each topological perturbation as a fall of a domino tile: every time a new interaction appears in the network, TILES first updates the communities locally and then propagates the changes to the node surroundings adjusting the neighbors' community memberships. The online nature of TILES brings many advantages. First, the computation of network sub-structures is local and involves a limited number of nodes and communities, thus speeding up the updating process. Second, our approach allows to observe two types of evolutionary behaviors: (1) the stability of individuals' affiliations to communities, and (2) the evolution over time of interaction-based communities.

We validate the effectiveness of our algorithm by comparing it with state-of-the-art community discovery algorithms, using both synthetic and real networks enriched by ground truth communities. In our experimental analysis we underline that TILES is able to achieve a better match with the ground truth communities than the compared algorithms. Moreover, we show that TILES guarantees lower execution times than the competitors since it can be easily parallelized. We also provide a characterization of the communities extracted by our algorithm by analyzing three Big Data sources: a nation-wide call graph of one million users whose interactions are tracked for one month; a Facebook interaction network which covers a period of 52 weeks; an interaction network of 8 million users of the Chinese microblogging platform WEIBO observed for 1 year.

The paper is organized as follows. Section 2 summarizes the related works in community discovery, dynamic network analysis and evolutionary community discovery; Section 3 formalizes the problem of Evolutionary Community Discovery. Section 4 describes TILES providing algorithmic details and showing some characteristics of the algorithm. In Sect. 5 we compare TILES with other community detection algorithms and present a characterization of discovered communities. Finally, Sect. 6 concludes the paper, describing some scenarios for future works.

[1] Temporal Interactions a Local Edge Strategy.

2 Background and related works

The problem of finding and tracking communities in an evolutionary context is relatively novel. Here we discuss some relevant works regarding classical community discovery, dynamic networks analysis and, as their merging point, evolutionary community discovery.

2.1 Community discovery

The problem of finding communities in complex networks is a hot topic, as witnessed by the high number of works in this field. A survey by Fortunato (2010) explores all the most popular techniques to find communities in complex networks. The more recent survey by Coscia et al. (2011) tries to classify families of algorithms based on the typology of the extracted communities. The classic definition of community relates to a dense subgraph, in which the number of edges among its nodes is significantly higher than the number of outgoing edges. However, this definition does not cover many real world scenarios, and many different alternative definitions of communities have been proposed. One of the most famous is based on the modularity concept, a quality function of a graph partition proposed by Clauset et al. (2004), which scores high values for partitions whose internal cluster density is higher than the external density. An alternative approach is the application of information theoretic techniques, as for example in INFOMAP (Rosvall and Bergstrom 2008). An interesting property for community discovery is the ability to return overlapping sub-structures, i.e., to allow nodes to be part of more than one community. This property reflects the social intuition that each person is part of multiple different communities (e.g. work, family, hobby…). A wide set of algorithms were developed over this property, such as the one proposed in Palla et al. (2005). Other overlapping approaches are based on Label Propagation such as DEMON Coscia et al. (2012), a framework which allows a bottom–up formation of communities exploiting ego-networks. Given the rising interests on multiplex (multidimensional/multi-relational) networks, recently some community discovery algorithms able to partition labeled multigraph have been proposed (Boden et al. 2012).

2.2 Dynamic network analysis

Several graph problems are, by their nature, closely tied to network dynamics (Kostakos 2009). The flowing of time plays different roles over a complex network: it can determine the evolution of the graph topology (e.g. edges and node can fall and rise, communities born and die) or lead to the observation of diffusion processes. Among the problems related to network evolution, Link Prediction is one of the most studied: formulated by Nowell and Kleinberg (2003) its aim is to predict edges that will appear in the future given the actual state of the network. Models for network growth, as the ones proposed in Barabási and Albert (1999) and Leskovec et al. (2005), replicate network evolutions peculiarity in order to build synthetic graphs. Furthermore, diffusion processes have been studied in order to understand virus epidemics (Wang et al. 2009) and spreading of innovations (Burt 1987).

2.3 Evolutionary community discovery

Communities are certainly the mesoscale structures most affected by changes in network topology: as time goes by the rise and fall of nodes and edges determines the appearance and vanishing of social clusters that static community discovery algorithms are unable to detect. In order to understand how communities evolve, three main approaches have been

followed so far: Independent Community Detection and Matching, Global Informed iterative community detection, and Local Informed iterative community detection. In the following, we report a survey on these categories of approaches.

2.3.1 Independent community detection and matching

Strategies that fall in this category are prevalently aimed to track the evolution of communities by identifying key actions which regulate their life (birth, death, merge, split). Nguyen proposes an extended life-cycle model able to track, in an offline fashion, the evolution of communities (Nguyen 2012). Such methodology, as well as the one introduced in Goldberg et al. (2011), Dhouioui and Akaichi (2014), Takaffoli et al. (2014) and Asur et al. (2009), works on a two-step procedure: (1) the graph is divided in n temporal snapshots and, for each of them, a set of communities is extracted; (2) for each community an evolutionary chain is built by observing its evolution through temporal adjacent sets. In their work, Takaffoli et al. introduced MODEC (Takaffoli et al. 2011), a framework able to model and detect the evolution of communities obtained at different snapshots in a dynamic social network. The problem of detecting the transition of communities is solved by identifying events that characterize the changes of the communities across time. Unlike previous approaches (Palla et al. 2007) the MODEC framework is independent from the static community mining algorithm chosen to partition time-stamped networks.

2.3.2 Global informed iterative community detection

A different methodology to detect communities in a dynamic scenario, is to design a procedure where each community identified at time t is influenced by the ones detected at time $t - 1$ avoiding the need to match communities, thus introducing global *smoothness* in the community identification process. The approaches belonging to this category derive from the evolutionary clustering analysis (Chakrabarti et al. 2006). Folino and Pizzuti (2014) propose an evolutionary multi-objective approach to community discovery in dynamic networks which, moving from an evolutionary clustering perspective, searches for smooth community transitions among consecutive time steps. Rozenshtein et al. (2014) focus on identifying the optimal set of time intervals to discover dynamic communities in interaction networks. Although those approaches reduce the complexity of the matching phase, they are based on a static temporal partition of the complete temporal network. Other works belonging to this category are Sun et al. (2010), Shang et al. (2012) and Guo et al. (2014).

2.3.3 Local informed iterative community detection

The last category, also known as *online* approaches, is defined by algorithms that do not partition the full temporal annotated graph, but try to build and maintain communities in an online fashion following the rising and vanishing of new nodes and edges. Only a few works, at the best of our knowledge, have exploited this strategy so far. Qi et al. (2013) propose a probabilistic approach to determine dynamic community structure in a social sensing context. The main objective of the introduced IC- DRF model is to dynamically maintain a community partition of moving objects based on trajectory information up to the current timestamp. However, due to the information used to update the community membership, the approach is suitable only for a specific kind of networked data. Lin et al. (2008) propose an iterative algorithm that, avoiding the classical two-step analysis, extract communities taking care of the topology of the graph at the specific time frame t as well as the historical

evolutive patterns of previously computed communities. In Cazabet et al. (2010), Cazabet introduces iLCD an overlapping online approach to community detection which re-evaluates communities at each new interaction according to the path lengths between each node and its surrounding communities. Xu et al. (2013) propose an algorithm aimed at analyzing the evolution of community cores. The proposed approach tracks only stable links within face to face interaction graphs exploiting a rule based on-line approach. Other works belonging to this category are Zakreweska and Bader (2015), Lee et al. (2014) and Nguyen et al. (2011). TILES belongs to the latter family of approaches. However, unlike the previously mentioned algorithms, it uses only local topological information and a constrained label propagation in order to minimize the computation needed to maintain updated the community structure.

3 Evolutionary community discovery

The overwhelming number of papers proposed in recent years clearly expresses that researchers are not interested in formulating *"The Community Discovery algorithm"* but in finding the right algorithm for each specific declination of the problem. Moving from this observation we tackle a specific and not yet deeply studied problem: evolutionary community discovery in dynamic social networks.

Definition 1 (Evolutionary Community Discovery) Given an interaction streaming source S and a graph $G = (V, E)$, where $e \in E$ is a triple (u, v, t) with $u, v \in V$ and $t \in \mathbb{N}$ is the time of the interaction's generation by S, the Evolutionary Community Discovery (ECD) aims to identify and maintain updated the community structure of G as new interactions are generated by S.

The source S produces new interactions among pair of nodes which can be either already part of the graph or newcomers. It models scenarios in which interactions do not occur with a rigid temporal discretization but flow "in streams" as time goes by. After all, this is how social interactions actually take place: phone calls, SMS messages, tweets, Facebook posts are produced in a fluid streaming fashion and consequently the corresponding networks' social communities also change fluidly over time. In contrast with a static community detection algorithm, an ECD algorithm must produce a series of communities' observations in order to describe how time shapes network topologies in coherent substructures. Moreover, an ECD algorithm should address the following question: given a community C at timestamp t and a streaming source S, what its structure will be at an arbitrary time $t + \Delta$ given the interactions produced by S? In order to answer this question the discovery process must be able to *smooth* the evolution of a community from t to $t + \Delta$ by identifying its local mutations avoiding external matching as done by traditional two-step approaches. Our algorithm, TILES, is designed to solve the ECD problem since it tracks the evolution of communities following a domino tile strategy.

4 TILES algorithm

Social interactions determine how communities form and evolve: indeed, the rising and vanishing of interactions can change the communities' equilibrium. A common approach in literature to address topology dynamics is to: (1) split the network into temporal snapshots, (2) repeat a static community detection for each snapshot and (3) study the variation of

the results as time goes by. This approach introduces an evident issue: which temporal threshold has to be chosen to partition the network? This problem, which is obviously context dependent, also introduces another one: once the algorithm is performed on each snapshot how can we identify the same community in consecutive time slots? To overcome these issues we propose TILES, an ECD algorithm that does not impose fixed temporal thresholds for the partition of the network and the extraction of communities. It proceeds analyzing an interaction stream: every time a new interaction is produced by a given streaming source, TILES uses a label propagation procedure to diffuse the changes to the node surroundings and adjust the neighbors' community memberships. A node can belong to a community with two different levels of involvement: *peripheral* membership and *core* membership. If a node is involved in at least a triangle with other nodes in the same community it is a core node while if it is an one-hop neighbor of a core node it is a peripheral node. Only core nodes are allowed during the label propagation phase to spread community membership to their neighbors. TILES generates overlapping communities, i.e. each node can belong to different communities which can represent the different spheres of the social world of an individual (friendship, working relations, etc.).

The algorithm[2] takes as input four parameters: (1) the graph G, which is initially empty; (2) an edge streaming source S; (3) τ, a temporal observation threshold; (iv) a Time To Leave (ttl) value for the interactions. The temporal observation threshold τ specifies how often we want to observe the structure of the communities allowing us to customize the output of the algorithm. Furthermore, ttl models the expected lifespan of a new interaction: it acts as a temporal decreasing *countdown* that, when expired, leads to the removal of the edge it is attached to. Indeed the value of ttl impacts the overall stability of the observed phenomena.

Studying a dynamic network we can model its evolutionary behavior to comply with one of the following general scenarios: (a) *accumulative* growth or (b) *limited memory* growth. The former assumes that once an interaction among a pair of nodes takes place it is permanent; the latter states that interactions gradually lose their strength as time goes by till they disappear: ttl is introduced to interpolate these two behaviors and indicates the time after which an edge (u, v) decays if there were no interactions between nodes u and v. The output of the algorithm is a chronologically ordered sequence of community sets, each set representing the network partition as it appears at the end of an interval of size τ. Each community within a set is composed by two groups of nodes: the ones belonging to the community *periphery* and the ones belonging to the community *core*. Setting τ equals to the streaming source clock ensures the output of community status every new network update.

Algorithm 1 shows the behavior of TILES. First of all a new interaction $e = (u, v)$ generated by the source S is added to the graph (lines 3–8). Then the following scenarios are considered:

1. both nodes u and v appear for the first time in the graph. No other actions are performed until the next interaction is produced by the source S (Fig. 2a, Algorithm 1 lines 10–12);
2. one node appears for the first time and the other is already existing but peripheral or both nodes are existing but peripheral, in any case they do not belong to any community core. Since peripheral nodes are not allowed to propagate the community membership, no action is performed (none of the "if" clauses is satisfied) until the next interaction is produced by the source S (Fig. 2a, Algorithm 1 lines 14–18);
3. one node appears for the first time in G while the other is an already existing core node. The new node inherits a peripheral community membership from the existing core node (Fig. 2b, Algorithm 1 lines 20–23);

[2] TILES Python implementation available at: https://github.com/GiulioRossetti/TILES

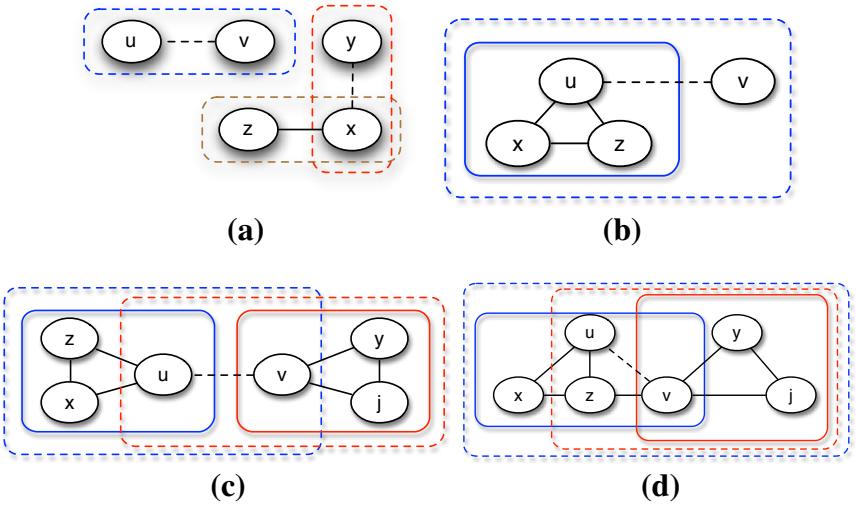

Fig. 2 Community updates at the appearance of a new edge. *Dashed black lines* identify new interactions; *dashed line rectangles* identify the periphery of communities; *continuous line rectangles* identify core communities. **a** Both nodes appears for the first time (*blue*) or one of them already exists but is not core (*red*): a peripheral community is created. **b** A new node v joins a core community and becomes part of its periphery; **c** a new interaction emerges between nodes u and v which are core nodes in different communities: the two nodes become peripheral nodes of the other's node core community. **d** A node, core for a community and peripheral in a different one, becomes core in the latter (Color figure online)

4. both nodes are core nodes already existing in G (Algorithm 1 lines 25–33). In this case we have two possible sub-scenarios:
   (a) Nodes u and v do not have common neighbors (we identify as $\Gamma(u)$ the set of neighbors of u): they propagate each other a peripheral community membership through the PERIPHERALPROPAGATION procedure (Fig. 2c, Algorithm 1 lines 27–29);
   (b) Nodes u and v do have common neighbors: their community memberships are re-evaluated and the changes propagated to their surroundings by the COREPROPAGATION function (Fig. 2c, d, Algorithm 1 lines 30–32).

TILES's communities grow gradually expanding their core and their peripheries through the PERIPHERALPROPAGATION (see Algorithm 2) and the COREPROPAGATION (see Algorithm 3) procedures.

The PERIPHERALPROPAGATION procedure regulates the events where a new node becomes part of an already established community. Since, initially, the newcomer is not involved in any triangle with other nodes of the community it becomes part of its periphery. The same function is performed when a new interaction connects existing nodes that do not share any neighbors.

The COREPROPAGATION procedure assumes that nodes u and v have at least one common neighbor z. For each triple (u, v, z) if at least two nodes are core for the same community the third one becomes core as well (Fig. 3c, d Algorithm 3 lines 8–16), otherwise a new community is created upon the new triangle (Figure 3a, b, Algorithm 3 lines 2–6). Once the core nodes are established they propagate a peripheral membership to their neighbors, if they are not already within the community. In Algorithm 3, the operator $C(\cdot)$ is used to

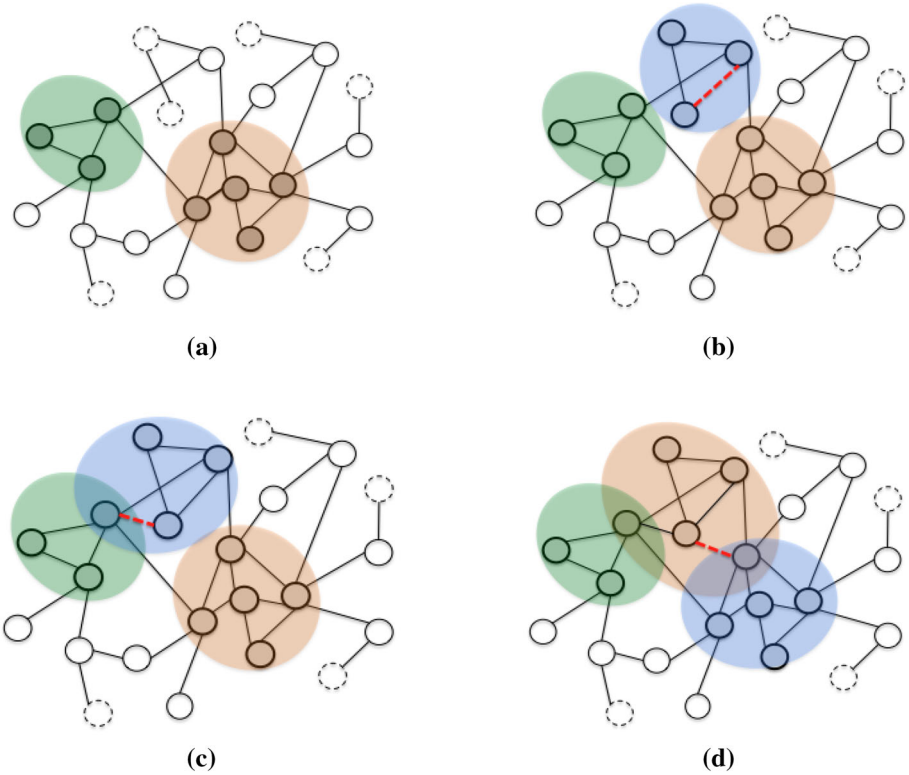

Fig. 3 Example of TILES community growth. We consider four consecutive updates extracted from a Facebook interaction network, each new interaction is depicted with a *red dashed line*. *Colored shapes* identify core communities. Nodes with *solid borders* outside the colored shapes are the peripheral nodes. Nodes with *dashed borders* outside the colored shapes are not involved in any community (Color figure online)

define the intersection of the communities of the nodes passed as parameters. TILES imposes a single condition for a node u to be in a *community core*: it must be involved in at least a triangle with others in the same *community core*. This choice guarantees an overall tightness of the observed topologies and avoids the presence of chain-like communities which, in our opinion, are not realistic structures in real social contexts.

4.1 Expired edges removal

We introduce a REMOVEEXPIREDEDGES procedure (Algorithm 4) to allow edges to decay. As we discussed, TILES execution is parametric on ttl (time to live) value for interactions: when $ttl = 0$ an edge disappears immediately after its rising producing an empty network at each new step; when $ttl = +\infty$ we fall in the accumulative growth scenario. Finally, if $ttl \in (0, +\infty)$ each edge decays after a ttl time from its generation by the streaming source S. In order to reduce the complexity of the removal phase the interactions are stored in an external priority queue (RQ in Algorithms 1 and 4) and retrieved chronologically from the oldest to the newest. Moreover if an existing interaction is refreshed before it expires we only update its priority in the queue restarting its ttl (Algorithm 1 line 4). In REMOVEEXPIREDEDGES (Algorithm 4) we analyze each interaction (u, v, t) present in QR and handle two scenarios:

Algorithm 1 TILES($\mathcal{G}$, S, τ, ttl)

Require: $\mathcal{G}$: undirected graph, S: streaming source, τ: temporal observation threshold,ttl: edges time to leave

```
1: actual_t = 0, RQ = { }
2: while S.ISACTIVE( ) do
3:    (u,v,t) ← S.GETNEWINTERACTION( )
4:    RQ.UPDATE(u,v,t)
5:    G.REMOVEEXPIREDEDGES(RQ, ttl, actual_t)
6:    if (u,v) ∉ G then
7:       G.ADDEDGE(e)
8:    end if
9:  ---------------------------------------------------------------
10:   if |Γ(u)|==1 and |Γ(v)|==1 then                          ▷ (1)
11:      Continue
12:   end if
13: ---------------------------------------------------------------
14:   core_u = G.GETCOMMUNITYCORE(u)                           ▷ (2)
15:   core_v = G.GETCOMMUNITYCORE(v)
16:   if core_u == ∅ and core_v == ∅ then
17:      Continue
18:   end if
19: ---------------------------------------------------------------
20:   if |Γ(u)|==1 and |Γ(v)| > 1 then                         ▷ (3)
21:      G.PERIPHERALPROPAGATION(u, {v})
22:   else if |Γ(u)| > 1 and |Γ(v)|==1 then
23:      G.PERIPHERALPROPAGATION(v, {u})
24: ---------------------------------------------------------------
25:   else                                                     ▷ (4)
26:      CN = Γ(u) ∩ Γ(v)
27:      if |CN| == 0 then                                     ▷ (4a)
28:         G.PERIPHERALPROPAGATION(u, {v})
29:         G.PERIPHERALPROPAGATION(v, {u})
30:      else                                                  ▷ (4b)
31:         G.COREPROPAGATION(u, v, CN)
32:      end if
33:   end if
34: ---------------------------------------------------------------
35:   if t - actual_t == τ then
36:      OUTPUTCOMMUNITIES(G)
37:      actual_t = t
38:   end if
39: end while
```

Algorithm 2 PERIPHERALPROPAGATION(u,nodes)

Require: u: node of G, nodes: a set of nodes

```
1: for v ∈ nodes do
2:    for c ∈ G.GETCOMMUNITYCORE(u) do
3:       G.ADDTOCOMMUNITYPERIPHERY(v, c)
4:    end for
5: end for
```

1. (u, v, t) is expired: the edge is removed from the graph and from RQ (Algorithm 4, lines 2-17);
2. (u, v, t) is still valid: since the interactions in RQ are ordered the execution of REMOVEEXPIREDEDGES is terminated (Algorithm 4, line 19).

Algorithm 3 CorePropagation(u,v,CN)

Require: u,v: nodes of G, CN: u & v common neighbors in G.

```
1: for z ∈ CN do
2:    if C(u,v,z) ≠ ∅ then
3:       G.CreateNewCommunity(u, v, z)                    ▷ New Community
4:       PeripheralPropagation(u, Γ(u))                   ▷ Periphery expansion
5:       PeripheralPropagation(v, Γ(v))
6:       PeripheralPropagation(z, Γ(z))
7:
8:    else if C(u,v) ≠ ∅ then
9:       G.AddToCommunityCore(z, C(u, v))                 ▷ Core expansion
10:      PeripheralPropagation(z, Γ(z))                   ▷ Periphery expansion
11:   else if C(u,z) ≠ ∅ then
12:      G.AddToCommunityCore(v, C(u, z))
13:      PeripheralPropagation(v, Γ(v))
14:   else if C(z,v) ≠ ∅ then
15:      G.AddToCommunityCore(u, C(z, v))
16:      PeripheralPropagation(u, Γ(u))
17:   end if
18: end for
```

Algorithm 4 removeExpiredEdges(RQ, ttl,$actual_t$)

Require: RQ: a priority queue containing the edge candidate to be removed, ttl: edges time to live, $actual_t$: actual timestamp.

```
1: for (u, v, t) in RQ do
2:    if (actual_t - t) ≤ ttl then                        ▷ (u,v) ttl is expired
3:       G.removeEdge(u,v)
4:       RQ.remove(u,v)
5:       to_update = {Γ(u) ∩ Γ(v)}∪{u, v}
6:       for community ∈ C(u,v) do
7:          components = G.getComponents(community)
8:          if |components| == 1 then
9:             G.UpdateNodeRoles(community, to_update)
10:         else
11:            for c ∈ components do                     ▷ Handling splits
12:               sc = G.NewCommunity(c)
13:               G.RemoveNodes(community, V_c)
14:               G.UpdateNodeRoles(sc, c)
15:            end for
16:         end if
17:      end for
18:   else
19:      return                                           ▷ No more expired edges
20:   end if
21: end for
```

The removal of an interaction (case 1) affects the communities shared by its endpoints: in particular, the removal of (u, v, t) produces the re-evaluation of community memberships for nodes u, v and their first level neighbors. Two scenarios can occur after such events:

(a) the original community is not "broken", i.e. it is still composed by a single component: we need only to re-evaluate the "roles" of nodes u, v and their first level neighbors (Algorithm 4, lines 8–9);
(b) the original community is splitted into two or more separate entities: each component is then considered as a new community and the "roles" of nodes are recomputed (Algorithm 4, lines 10–16).

Algorithm 5 UpdateNodeRoles(c, to_update)

Require: c: community, to_update: set of nodes.
1: **for** node $\in$ to_update **do**
2: **if** ClusteringCoefficient(node, c)>0 **then**
3: $c_{core} = c_{core} \cup \{node\}$ ▷ Add node to the core
4: **else**
5: **if** node $\in$ c_{core} **then**
6: $c_{core} = c_{core} - \{node\}$ ▷ Remove node from the core
7: $c_{periphery} = c_{periphery} \cup \{node\}$ ▷ Add node to the periphery
8: G.RemovePeriphery(node, c) ▷ Clean the first level neighborhood
9: **end if**
10: **end if**
11: **end for**

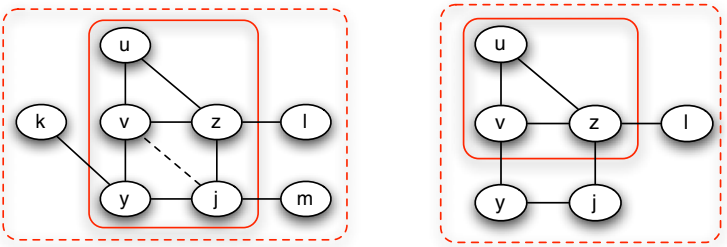

Fig. 4 Example of expired edge removal. **a** A community where the edge (v, j) is candidate for removal (the *dashed line*). **b** The updated community: node j and y leave the community core because they are not involved in any triangle with core nodes (clustering coefficient equals to 0); node k and m leave the community periphery due to the propagation phase

In order to assign each node to the *periphery* or the *core* of a community we define the UpdateNodeRoles procedure (Algorithm 5).
UpdateNodeRoles analyzes the local clustering coefficient (CC) of each node within the specific community and retains as core nodes the ones with $CC > 0$ (Algorithm 5, lines 2–3) and as peripheral nodes the ones with $CC = 0$ (Algorithm 5, lines 4–9). Once ensured that the node roles within the modified community are consistent, a propagation is performed on the neighborhood of nodes which moved from the *core* to the *periphery* (Algorithm 5, line 8): peripheral nodes attached only to the demoted core node are removed from the community. Figure 4 shows an example of edge removal scenario.

Even if in the proposed formulation the interaction removal is performed through a fixed size sliding window controlled by the ttl parameter, Tiles can be easily parametrized to allow custom removal strategies. For instance, we can substitute the interaction validity check (Algorithm 4 line 2) with a decay function[3] or to handle directly interaction removal as done for the insertion, i.e. leveraging if available in the data explicit information on edge vanishing. Certainly this latter scenario represents the optimum since the analyst does not need to define arbitrary thresholds and/or make additional assumptions. We choose to adopt a sliding window in order to provide a simple and tunable way to simulate asynchronous updates, an approach often used when dealing with temporal annotated data streams.

[3] The analysis of the distribution of interaction inter-arrival times is a very challenging topic, see Passarella et al. (2011), Boldrini et al. (2011), Burt (2000) and Goh and Barabási (2008).

4.2 Computational complexity of TILES

Since TILES operates on streaming data its complexity analysis depends on two main operations: interaction insertion and removal.

4.2.1 Interaction insertion phase

Interaction insertion causes perturbations on the network topology and it can induce updates on the community structure. As shown in Algorithm 1 there are five mutually exclusive rules that apply when a new interaction (u, v, t) arises:

- both u and v appear for the first time: no action taken, so complexity $O(1)$;
- at least one node was already present in the network but both nodes are not core: no action taken, so complexity $O(1)$;
- node u is core in one or more communities and node v is new: PERIPHERALPROPAGATION is called on the new node: the function cycles on the communities for which u is core to perform the propagation on v, so complexity $O(core(v)) < O(|V|)$;
- u and v are both core nodes for one or more communities that do not share neighbors: PERIPHERALPROPAGATION is called on both nodes, so complexity $O(core(v) + core(u)) < O(|V|)$;
- u and v are core nodes for one or more communities that share neighbors: COREPROPAGATION is called. This function cycles over the common neighbors of u and v, which in the worst case scenario are all the nodes of the network ($O(|V|)$), updates the community cores and performs PERIPHERALPROPAGATION $O(core(v) + core(u))$. Thus the final complexity is $O(|V| * (core(u) + core(v)))$.

In the worst case scenario this step has complexity $O(|V| * (core(u) + core(v)))$ since the most costly rule is applied when the edge endpoints u and v are core nodes that share $|V|$ neighbors. However, reaching this upper bound is unusual due to the power law degree distribution which characterizes real world interaction networks (it applies only for few hubs in the network).

4.2.2 Interaction removal phase

The interaction removal phase complexity can be decomposed in two steps:

1. Main loop on the removal queue RQ (Algorithm 4). The cycle is executed until a valid interaction is found:
   - if $ttl = 0$ (zero-memory scenario) it consumes all the edges in RQ: therefore, we have $O(|E|)$ cycles;
   - if $0 < ttl < \infty$, $|RQ_{ttl}|$ is the expected average size of the interactions processed when REMOVEEXPIREDEDGES is called: we have $O(|RQ_{ttl}|) < O(|E|)$ cycles;
   - if $ttl = \infty$ (full memory scenario), the removal is not executed at all, thus the complexity is $O(1)$.
2. Node role update (Algorithm 5). The main computational cost here is due to the clustering coefficient computation for each of the selected nodes on the community-induced graph. A naive implementation, assuming a complete clique, has cubic cost on the cardinality of nodes to be updated $O(|to_update|^3)$. Interaction networks are sparse so, in order to provide a more realistic complexity, we can assume that that every node in the set

has $\sqrt{|to_update|}$ neighbors within the community: thus, we can estimate the overall complexity with $O(\sqrt{|to_update|}^2) * |to_update| = O(|to_update|^2)$.

Considering the common scenario where $0 < ttl < \infty$, the final complexity is $O(|RQ_{ttl}| * |to_update|^2)$. Moreover, it is worth noting that as $ttl \to 0$, $|RQ_{ttl}|$ becomes small because each interaction is removed right after its appearance, while when $ttl \to \infty$ it increases its size. In the latter case the removal phase is executed rarely or not executed at all when ttl exceeds the observation period available for the data.

4.3 TILES properties

Given its streaming nature TILES shows two main properties: (1) it can be used incrementally on a precomputed community set; (2) it can be parallelized if specific conditions are satisfied. Moreover, in presence of a deterministic interaction source S (i.e., a generator that always produces the same ordered sequence of interactions), the output of TILES is uniquely determined. In this section we discuss and formalize such characteristics.

4.3.1 Incrementality

As specified above, TILES is called on an initially empty graph. However, it also works on non-empty graphs whose nodes are assigned to valid TILES communities. Given a deterministic streaming source S_t at time t and a non-empty graph G_t, whose nodes are assigned to a (valid) TILES community set $C_t = (c_1, c_2, \ldots, c_n)$, the following property holds:

$$TILES(G_t, S_t) = TILES(G, S_0) \tag{1}$$

where G is an empty graph, S_0 is the streaming source S at the initial time $t = 0$. In Equation 1 the $TILES$ function takes as inputs a graph and a streaming source and returns the partition computed once the stream ends. Since TILES updates the partition incrementally the final communities produced starting with the source S on a given graph G at time 0 or at time t are identical – assuming S deterministic.

Incrementality is a property that an online algorithm operating on streaming data must satisfy: it assures that every new network perturbation produces updates in the community status and that the computation proceeds smoothly one interaction after the other. Moreover it ensures that the approach does not require external community matching across time as done by two-step approaches. Incrementality, if the stream source is deterministic, imposes that the final partition is univocally determined. In TILES this is ensured by construction by the mutual exclusivity of the update rules: given a new edge and a given network status only a single pattern among the ones described can be executed (both for insertion and deletion). On the other hand in case of a non-deterministic streaming source, incrementality guarantees the smoothness of the updates but not that the final partition will be always the same (i.e., given a set of interactions generated with a different ordering w.r.t. a previous TILES execution it is not assured to reach the same final partition).

4.3.2 Compositionality

TILES is parallelizable by identifying disjoint streams of edges produced by the deterministic source S. Given a graph G, and two disjoint streams of edges S^i, S^{ii} iff $\forall (u_1, v_1) \in S^i$ $\forall (u_2, v_2) \in S^{ii}$: $(C(u_1) \cup C(v_1)) \cap (C(u_2) \cup C(v_2)) = \emptyset$, where $C(\cdot)$ returns the set of

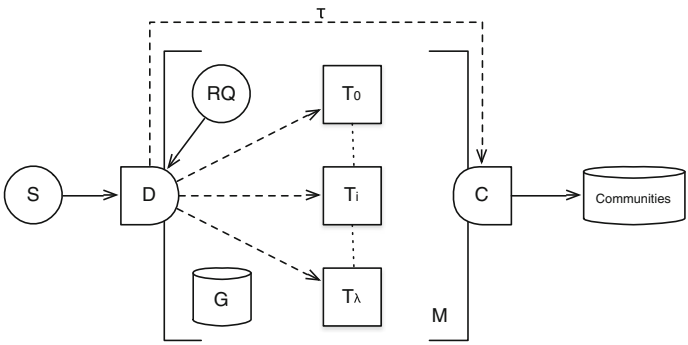

Fig. 5 TILES parallelization schema. The interactions produced by S as well as the ones in the removal queue RQ are handled by a dispatcher D, which checks the parallelization constraints and, if satisfied, assigns the interactions to the λ TILES workers. A collector C outputs the communities every time τ. The dispatcher, the workers and the collector access the graph G via a shared memory M

communities the node is part of, then:

$$TILES(G, S) = TILES(G, S^{i}) \cup TILES(G, S^{ii}) \tag{2}$$

The underlying idea is to operate updates on network subgraphs that are disjoint w.r.t. the communities assigned to their nodes: this parallel decomposition of the original problem is made possible by the constrained label propagation used to spread community membership. By definition, each local perturbation affect only the communities shared by the nodes involved in it thus allowing for parallel updates of the remaining communities. This property also holds for the edge removal phase considering the prioritized removal queue RQ as a streaming source. Isolating interactions among nodes of different communities allow us to parallelize the algorithm speeding up of the computation of community evolution.

Since identifying disjoint source streams is a challenging problem, TILES parallelization is achieved by adopting a map-reduce approach with shared memory (see Fig. 5): to this extent we interpose a dispatcher D between the stream S and the λ TILES workers which share the same memory space. The dispatcher collects λ consecutive updates, evaluates if they satisfy the introduced constraints and, if so, it distributes them to the workers. If the constraint are violated the updates are assigned to a single worker maintaining their original order. At the end of each observation period a collector accesses the shared memory and outputs the observed communities.

5 Experimental results

Evaluating the results provided by a community detection algorithm is a hard task, since there is not a shared and universally accepted definition of what a community is. In literature each approach provides its own community definition, often maximizing a specific quality function (e.g. modularity, density, conductance, …). Even though the communities identified by a given algorithm on a network are consistent with its community definition, it is not guaranteed that they are able to capture the real sub-topology of the network. For this reason, a common methodology used to assess the quality of a community detection algorithm is to evaluate the similarity between the partition it produces with the ground truth communities of the analyzed network.

In this section, we compare TILES to other state-of-the-art algorithms on both synthetic and real networks with ground truth communities (Sect. 5.1). Moreover, we characterize the communities our algorithm produces on three large-scale real-world datasets of social interactions (Sect. 5.2), discuss the event-based community lifecycle of communities (Sect. 5.3) and, finally, analyze the impact of the ttl parameter on the node/community stability.

5.1 Evaluation on networks with ground truth communities

We compare TILES with other static (DEMON and CFINDER) and dynamic (ILCD) overlapping community discovery algorithms. In order to cope with the absence of edge removal in the other algorithms we instantiate TILES for an accumulative growth scenario ($ttl = \infty$). DEMON (Coscia et al. 2012) is a bottom-up approach which exploits label-propagation to identify communities from ego-networks.[4] CFINDER (Palla et al. 2005) is an algorithm based on clique percolation that searches for clique-based network structures.[5] ILCD (Cazabet et al. 2010) is an algorithm for dynamic networks which re-evaluates communities at each new interaction produced by a streaming source.[6] In particular, every time a new interaction appears ILCD recomputes communities according to the path lengths between each node and its surrounding communities.

The slightly different community definitions introduced by the chosen algorithms make questionable a direct comparison of the outputs obtained on the same network when a ground truth is not provided. To overcome such issue and perform the analysis in a controlled environment we use both synthetic and real networks with ground truth communities.

5.1.1 Synthetic networks

LFR (Lancichinetti and Fortunato 2009) is the synthetic graph generators mostly used to evaluate community discovery algorithms since it provides, along with real-world like network topologies, annotated ground truth partitions. We performed a controlled experiment by generating multiple networks varying the following LFR parameters:[7]

- N, the network size (from 1k to 500k nodes);
- C, the network density (from 0 to 0.9, steps of 0.1);
- μ, the average per-node ratio between the number of edges to its communities and the number of edges with the rest of the network (from 0 to 0.9, steps of 0.1).

Varying the values of N, C and μ, we produced a total of 2500 different synthetic networks. Since the LFR benchmark does not generate a timestamped stream of edges, we imposed a random temporal ordering on the network edges in order to simulate the streaming source S for TILES and iLCD. We compared LFR ground truth communities with the communities obtained by TILES and the other algorithms using the Normalized Mutual Information (NMI) score, a measure of similarity borrowed from information theory (Lancichinetti et al. 2009):

$$NMI(X, Y) = \frac{H(X) + H(Y) - H(X, Y)}{(H(X) + H(Y))/2} \tag{3}$$

[4] DEMON Python implementation available at: https://github.com/GiulioRossetti/DEMON.

[5] CFINDER C implementation available at: http://www.cfinder.org/.

[6] ILCD Java implementation available at: http://cazabetremy.fr/iLCD.html.

[7] All the algorithms were executed on a Linux 3.12.0 machine with an Intel Core i7-2600 CPU @3.4GHzx8 at 3.2GHz and 8GB of RAM.

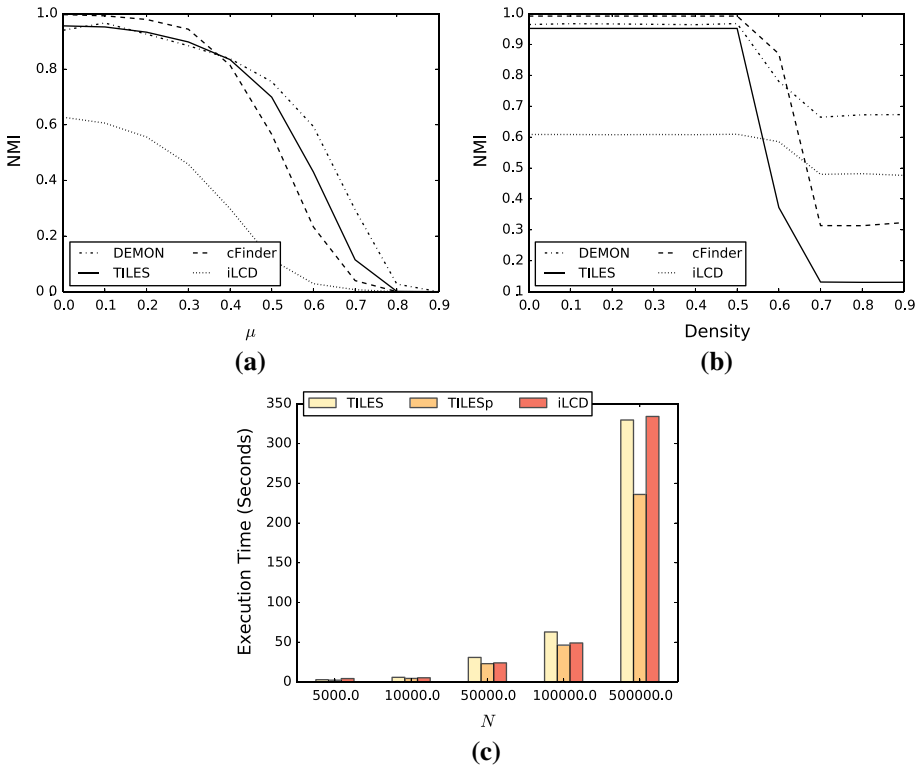

Fig. 6 Comparison of the algorithms on synthetic networks with ground truth communities. **a** NMI versus μ; **b** NMI versus network density; **c** runtime execution versus network size for TILES, iLCD and TILES$_p$ that uses two parallel processes

where $H(X)$ is the entropy of the random variable X associated to the community produced by the algorithm, $H(Y)$ is the entropy of the random variable Y associated to the ground truth community, whereas $H(X, Y)$ is the joint entropy. NMI ranges in the interval [0,1] and is maximized when the compared communities are identical. Figure 6 compares the NMI of the algorithms as the values of the LFR parameters vary. Varying μ we can observe that TILES produces communities whose NMI w.r.t. the ground truth is comparable to DEMON and CFINDER, but significantly outperforms iLCD (Fig. 6a). In Fig. 6b we observe that the NMI of the algorithms is stable till the density $C < 0.5$, i.e. half of all the possible edges are present in the network. As the network becomes dense ($C \geq 0.5$) we observe a drop in the NMI of the algorithms. Such a high density, however, is unusual for real interaction networks where density usually falls in the range [0.1, 0.2]. Fig. 6c compares the execution time of TILES, iLCD and TILES$_p$, an instantiation of the algorithm which exploits the compositionality property in order to parallelize the computation. While TILES has an execution time comparable to iLCD, TILES$_p$ significantly reduces the runtime by handling parallel updates.

5.1.2 Real networks

Moving from the results obtained on the LFR benchmark, we compared TILES and its competitor on four large-scale real-world networks. In order to do so, we adopted a novel community

evaluation technique able to cope with the computational issues that arise when calculating NMI on large community sets. Indeed, following Equation (3), given two sets of communities X and Y the former identifying the community extracted by an algorithm (having size m) and the latter representing the ground truth community set (having size n), in order to compute NMI it is necessary to identify the best community matches with cost $O(mn)$. Assuming $m \simeq n$ the NMI computation requires $O(n^2)$ comparisons, making it not suitable for large-scale networks. If for the synthetic networks we analyzed before the number of communities still allowed us to compute NMI in reasonable time, this is not the case for the four real world graphs we selected. In order to reduce the computational complexity, and thus speed up the evaluation process, we adopted the approach proposed in Rossetti et al. (2016):[8] given an algorithm community $x \in X$, (1) we label its nodes with their corresponding ground truth community $y \in Y$, then (2) we match community x with the ground truth community with the highest number of labels in community x. We define two measures:

– *Community Precision*: the percentage of nodes in algorithm community x labeled with ground truth community y, computed as

$$P = \frac{|x \cap y|}{|x|} \tag{4}$$

– *Community Recall*: the percentage of nodes in ground truth community y covered by algorithm community x, computed as

$$R = \frac{|x \cap y|}{|y|}. \tag{5}$$

The two measures describe, for each pair (x, y), the overlap between algorithm community x and ground truth community y: a perfect match is obtained when both *precision* and *recall* are 1. We also define a quality score for the algorithm community set by computing *precision* and *recall* on all the communities in the set and then computing their average *F1*-measure, the harmonic mean of $precision$ and $recall$:

$$F1 = 2\frac{precision * recall}{precision + recall}. \tag{6}$$

We applied this evaluation procedure to four large-scale real-world static networks with ground truth communities: dblp, Youtube, Amazon and LiveJournal.[9] Figure 7 compares the precision and recall of communities extracted by the four algorithms on the dblp dataset. In the density scatter plots, the upper-right corner (maximum precision and recall) identifies the optimal community match. We observe that TILES produces high quality communities (high overlap with the ground truth communities) while the other algorithms overestimate the ground truth: they generate communities that are bigger than the corresponding ground truth communities, resulting in high recall and low precision. In Table 1 we report the average $F1$-measure of the algorithms on the networks. Even if TILES is designed for a dynamic scenario, it produces the highest $F1$ scores guaranteeing at the same time low standard deviation.

5.2 Characterization on large-scale real networks

Once compared TILES with state-of-the-art approaches both in synthetic and real world data, in this subsection we analyze the communities it produces on three large-scale real-world

[8] NF1 Python code available at: https://github.com/GiulioRossetti/f1-communities

[9] The networks are available at: https://snap.stanford.edu/data/.

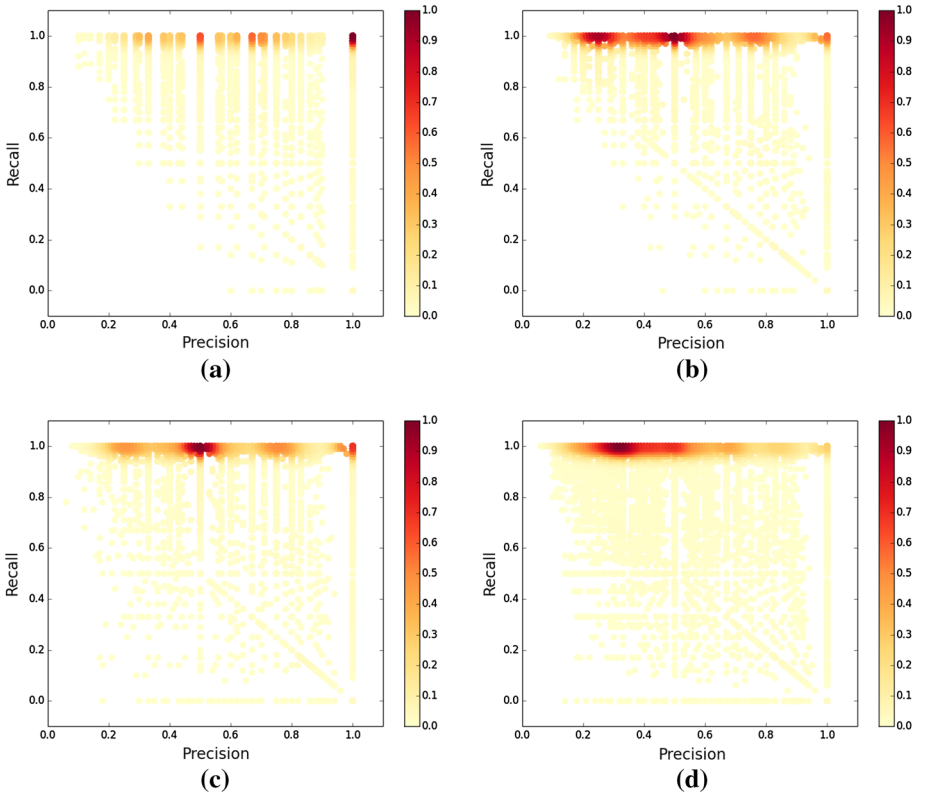

Fig. 7 Community precision and community recall produced by the four algorithms on the dblp network. The density scatter plots, compared the precision on the *x*-axis and the recall on the *y*-axis. A perfect match between the communities identified by a given algorithm and the ground truth is observed when both precision and recall are 1 (*top-right corner of the plot*). Color depth, in a gradient from *yellow* to *red*, indicates the density of (precision, recall) points: the deeper the color of a point, the higher the density of points around it (Color figure online)

Table 1 Real world network datasets

Network	Nodes	Edges	Coms.	CC	d	TILES	iLCD	cFINDER	DEMON
Amazon	334,863	925,872	75,149	.396	44	**.78**(.05)	.78(.23)	.77(.27)	.75(.24)
Dblp	317,080	1,049,866	13,477	.632	21	**.80**(.09)	.70(.23)	.74(.24)	.65(.24)
Youtube	1,134,890	2,987,624	8385	.080	20	**.64**(.11)	.42(.20)	.60(.20)	.42(.10)
LiveJournal	3,997,962	34,681,189	287,512	.284	17	**.73**(.17)	.71(.04)	.32(.30)	.64(.29)

For each network we indicate the number of nodes, edges and ground truth communities as well as the average clustering coefficient, CC, and the diameter, d: moreover, the average F1 scores produced by the compared algorithms is reported (standard deviation within brackets)
In bold the best F1 score for each network

dynamic interaction networks: a wall post network extracted from Facebook, a Chinese micro-blogging mention network, and a nation-wide call graph extracted from mobile phone data.

Table 2 General features of the networks

Network	Nodes	Edges	CC	#Observations (τ)
CG	1,007,567	16,276,618	0.067	10 (3 days)
FB07	19,561	304,392	0.104	52 (1 week)
WEIBO	8,335,605	49,595,797	0.014	52 (1 week)

CC identifies the network clustering coefficient, τ the observation window we use in the experiments

These dynamic networks allow us to characterize the communities produced by the algorithm in three slightly different scenarios: two "virtual" contexts where people share thoughts and opinions via social media platforms, and a "real" one where people directly keep in touch through a mobile phone device. In Table 2 are reported the main statistics of the selected networks.

5.2.1 Call graph

The call graph is extracted from a nation-wide mobile phone dataset collected by a European carrier for billing and operational purposes. It contains date, time and coordinates of the phone tower routing the communication for each call and text message sent by 1, 007, 567 anonymized users during one month. We discarded all the calls to external operators. In the experiments we adopt as τ a window of 3 days.

5.2.2 Facebook wallpost

The FB07 network is extracted from the WOSN2009 (Viswanath et al. 2009) dataset[10] and regards online interactions between users via the wall feature in the New Orleans regional network during 2007. We adopted an observation period τ of 1 week.

5.2.3 WEIBO interactions

This dataset is obtained from the 2012 WISE Challenge[11]: built upon the logs of the popular Chinese micro-blog service WEIBO,[12] its interactions represent mentions of users in short messages. We selected a single year, 2012, and used an observation window of one week.

It is worth noting that any arbitrary chosen value of τ does not affect the execution of TILES but only the number and frequency of community status observation. The τ threshold is introduced with the purpose of simplifying the analysis of results reducing the number of community observations. Note that in order to get as output the complete history of community updates (an observation for each local perturbation) it is sufficient to set τ equal to the clock of the streaming source.

We analyzed four aspects of the communities produced by TILES: (1) the distribution of community size; (2) the distribution of community overlap; (3) the distribution of communities' average clustering coefficient; (4) the transition time of nodes from the periphery to the core of communities. The size of TILES communities follows a heavy tail distribution for all

[10] http://socialnetworks.mpi-sws.org/data-wosn2009.html

[11] http://www.wise2012.cs.ucy.ac.cy/challenge.html.

[12] http://weibo.com.

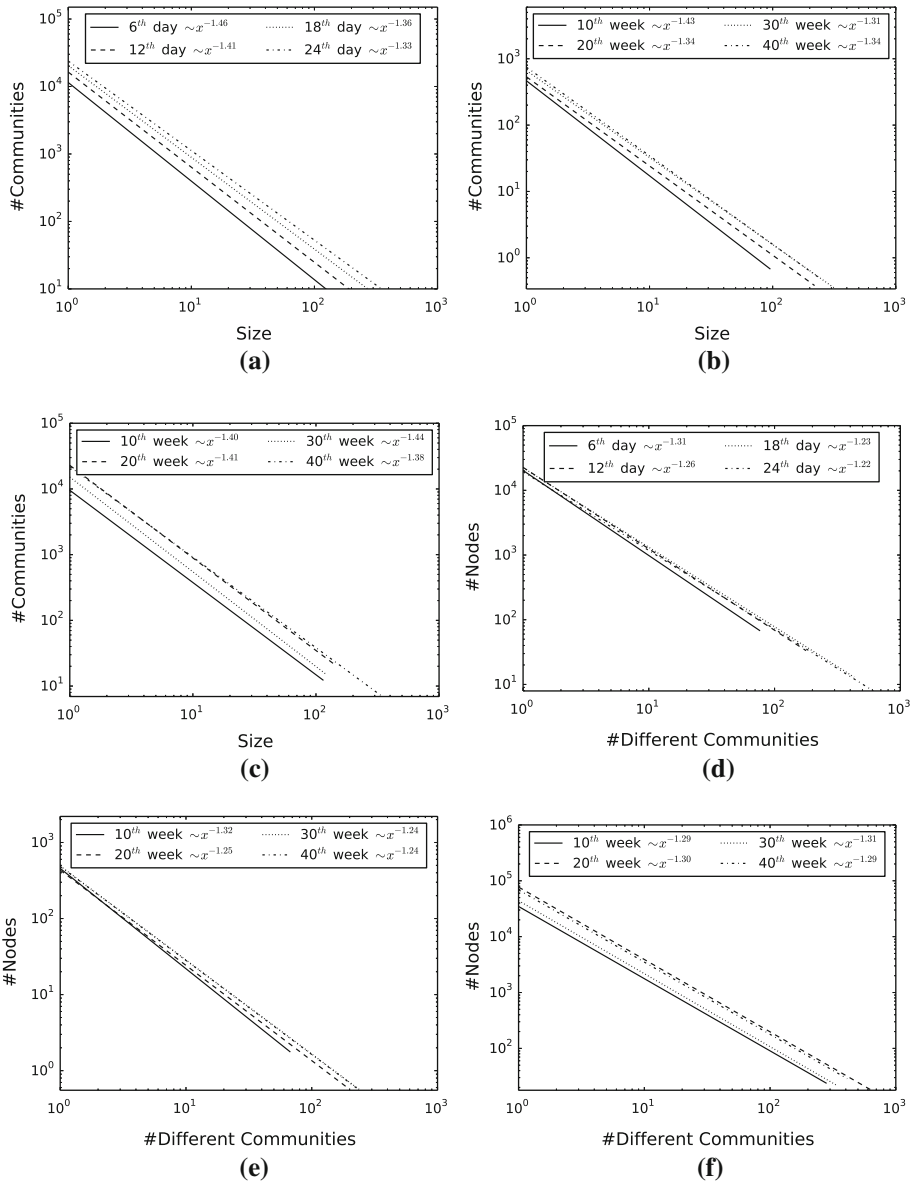

Fig. 8 *(First row)* Community size distribution fitted powerlaw for CG (**a**), FB07 (**b**) and WEIBO (**c**). *(Second row)* Node overlap distribution fitted powerlaw for CG (**d**), FB07 (**e**) and WEIBO (**f**)

networks (Fig. 8a, b, c). This means that the vast majority of communities have few nodes while a small but significant portion of them have several thousands nodes. Such a great heterogeneity also characterizes the community overlap, i.e. how many different communities a node belongs to (Fig. 8d, e, f). The majority of nodes belong to just one or two communities, while some nodes belong to thousands different communities. Figure 9 shows the average clustering coefficient of communities computed over the core nodes. Communities maintain

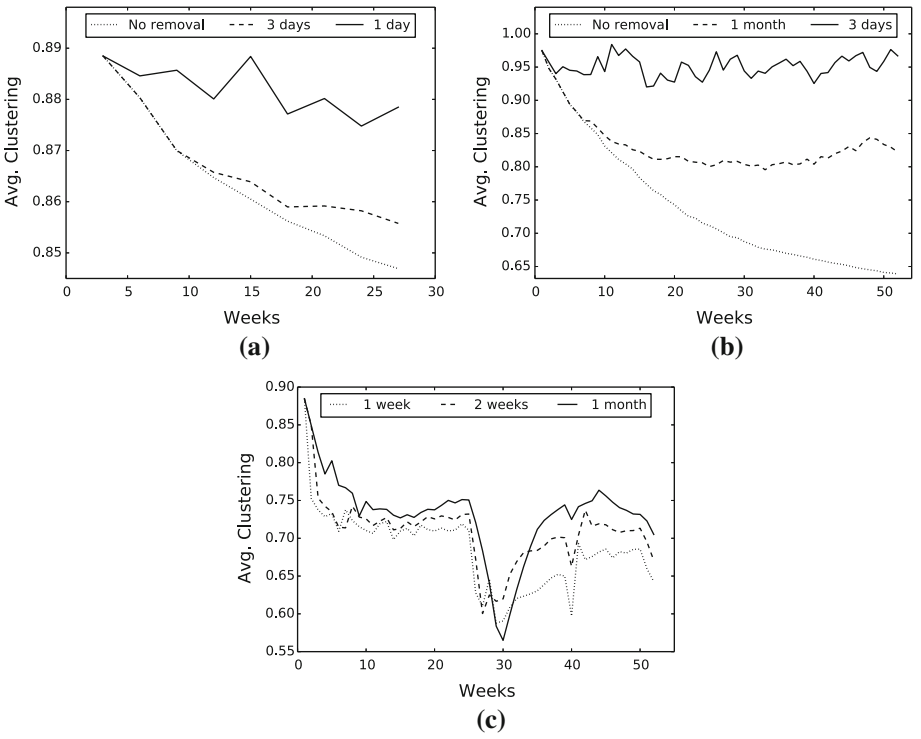

Fig. 9 Distribution of average clustering coefficient per community for CG (**a**), FB07 (**b**) and WEIBO (**c**). Each *color* identify a ttl value

high average clustering coefficients as time goes by, with minimum values of 0.6 for FB07 and WEIBO networks and 0.8 for CG. These values are significantly higher than the overall clustering coefficients of the networks (see Table 2) highlighting that TILES is capable of identifying dense network sub-structures. Moreover, increasing the ttl value causes a decrease in the average CC, with the exception of WEIBO for which we can observe a low drift (Fig. 9d, e, f). An explanation for this trend is that, when the interactions are persistent in time, communities grow at the expense of their internal cohesion causing a decrease of internal clustering coefficient.

TILES' definition also affects the overall node *coverage* of the identified communities, i.e. how many nodes are included into communities. This property is hence strictly related to the clustering coefficient of the analyzed network: the greater the clustering coefficient the higher is the nodes coverage. This tendency is depicted in Fig. 10a, b, c where we report, for CG, FB07 and WEIBO, how the ratio of "core" nodes, "periphery" nodes and the sum of the two (total nodes coverage) change in time during the period of observation. FB07 has a clustering coefficient of 0.104 showing a high coverage: the 80% of nodes are included in some communities. In contrast, CG and WEIBO reach a coverage ranging between 40 and 50%, due to their low overall clustering coefficient (0.067 and 0.014, respectively).

A peculiarity of TILES is the concept of community *periphery*. As discussed in Section 4 peripheral nodes are not involved in triangles with other nodes of the community. Every node first joins a community as peripheral node then it becomes a core one once

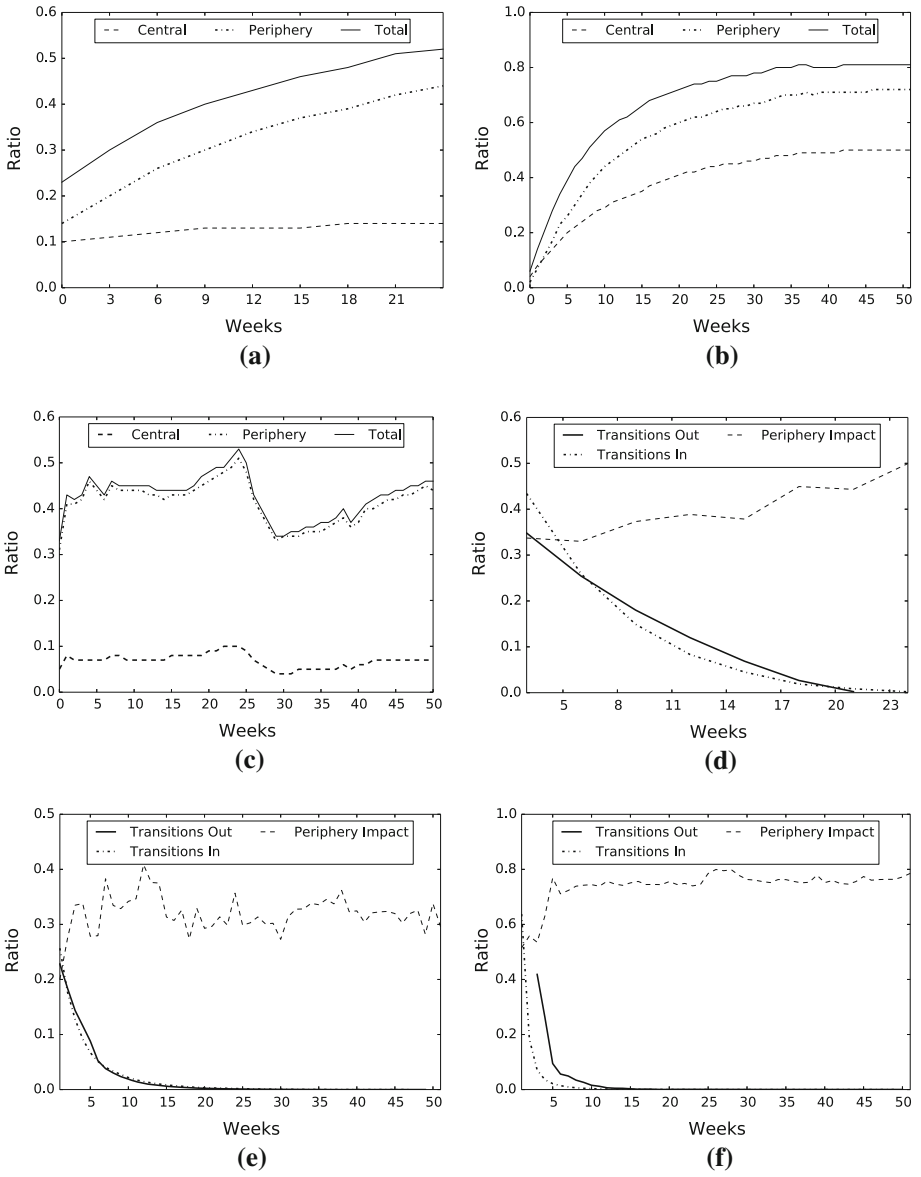

Fig. 10 *(First row)* TILES communities nodes coverage; *(Second row)* Distribution of transition time from periphery to core (transitions in) and viceversa (transition out) as well as ratio of nodes that move from the periphery to the core across consecutive observations (periphery impact). **a** CG-coverage, **b** FB07-coverage, **c** WEIBO-coverage, **d** CG, **e** FB07, **f** WEIBO

it is involved within a triangle with other core nodes. We found that the expected time of transition from the periphery to the core of a community is generally short (Fig. 10d, e, f, Transitions In): in CG 40 % of nodes become core nodes in just 3 days; in FB07 15 % of nodes perform the transition during the first week; in WEIBO almost 60 % of transitions occur within a single week. Moreover, the transitions of nodes from the core to the periphery

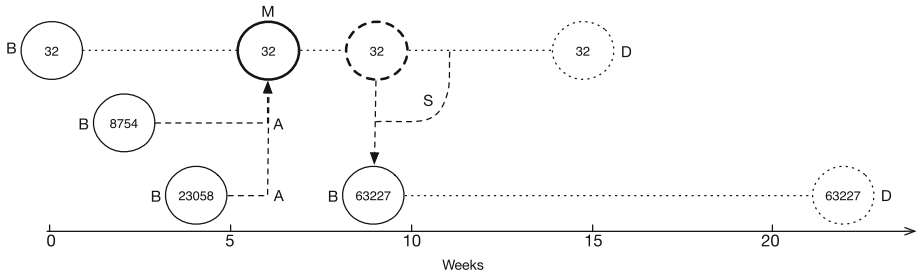

Fig. 11 Example of community lifecycle extracted from WEIBO. Each community is represented by a *circle* and identified by a *number*. Events are identified by the relative letter: (B) birth, (M) merge, (*A*) absorption, (*S*) split, (*D*) death. Merged communities and residual of splits are highlighted with *thicker lines*

(Fig. 10d, e, f, Transitions Out) follow distributions similar to the ones observed for the reverse path. However, if we do not consider the distributions shapes but the total number of both events an interesting pattern emerges: the number of nodes that are *"attracted"* by the core of communities are between 2 and 900 times more of than the nodes that follow the opposite route (263,483 to 8415 in FB07, 680,932 to 420,530 in CG and 6,373,316 to 7070 in WEIBO). This peculiarity highlights that the community cores are able to provide meaningful—and stable—boundaries around nodes that frequently interact with each other.

We also investigated how many nodes perform the transition from periphery to core across consecutive community observations: we asked ourself, given two observation of a community C, what is the ratio of core nodes in C at $t + \Delta$ that where in the peripheral nodes at time t? In Fig. 10d, e, f dashed line, we observe that this ratio has values between the 30 % and 50 % of the nodes for CG and FB07, and around 70 % for WEIBO. This means that in all the networks almost the half of the peripheral nodes become core nodes in the subsequent time window.

5.3 Event-based community lifecycle and time to leave analysis

Several works on evolutionary community discovery focus their attention on the identification and analysis of the events that regulates community life-cycles—birth, merge, split and death of communities. In this section we show how TILES allows us to easily capture these events by observing step by step the network evolution and tracking the perturbations of the communities. In order to perform an analysis of community life-cycles we identify five main events:

- **Birth** (B): the community first appearance, i.e. the rising of the first set of core nodes of the community;
- **Merge/Absorption**: two or more communities merge when their core nodes completely overlap: we define as Absorbed (A) the communities which collide with an existing one, we define as Merged (M) the already existing community;
- **Split** (S): a community splits in one or more sub-communities as consequence of the edge removal phase;
- **Death** (D): a community dies when its core node set becomes empty.

In Fig. 11 is reported an example of community life-cycle extracted from the WEIBO network. Figure 12 shows the event trends, i.e. the number of events of a given type as time goes by, for

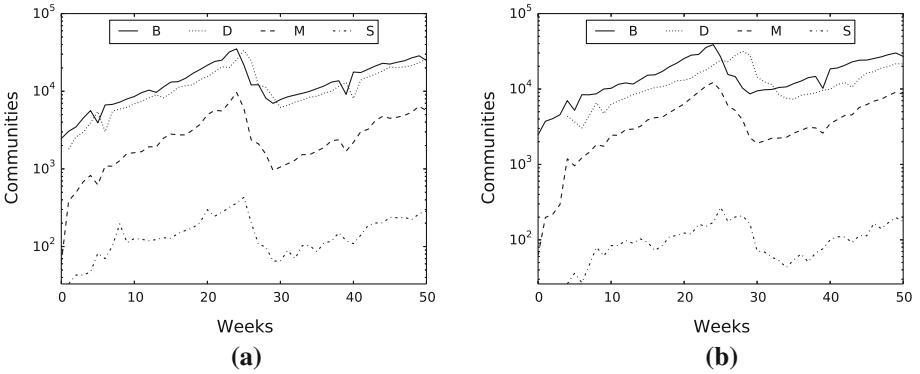

Fig. 12 The number of community events as time goes by in the WEIBO network for **a** one week removal scenario and **b** one month removal scenario. Each *line* represents a community event (birth B, merge M, absorption A, split S, death D)

the WEIBO network when the ttl is set to one week (a) and one month (b).[13] We can observe that all the compared trends follow similar patterns regardless the value of ttl, expressing only a slight increase of merge events and decrease of split events when ttl is set to one month.

We analyzed how the choice of ttl affects the communities characteristics by executing TILES with different ttl values on CG (1 day, 3 days, $+\infty$), FB07 (1 week, 2 weeks, 3weeks) and WEIBO (1 week, 2 weeks and 1 month). We have shown that ttl affects the trends of community life-cycle events: now our aim is to understand the degree of stability it induces on TILES communities at a micro level. To do so we analyze the impact of ttl on the rate nodes join and leave communities. Figure 13 shows two series of plots: (1) on the top, the trend values for rate of joins and leaves; (2) on the bottom, how this trend relates on average to the stability of communities (how many communities per observation are affected by at least a join/leave action). We observe that, as ttl increases, nodes and communities stabilize quickly and *leave* actions emerge less frequently. The reduction of *leave* actions depends on the ttl: by extending the time to live an interaction is more likely to renew than to expire. On the other hand, Fig. 14 shows the community average life (the number of weeks/days from its rising to its disappearance) w.r.t. its average size. A correlation between the two measures clearly emerges: in FB07 (average Pearson correlation $\rho = .88$, standard deviation $\sigma = .03$) and CG ($\rho = .4$, $\sigma = .11$) the bigger a community is the longer it lives regardless the value of ttl. In contrast, in WEIBO we observe a negative correlation ($\rho = -.52$, $\sigma = .13$) between size and average community life. Moreover, increasing ttl the expected life and size of communities tends to grow reaching their maximum when the removal phase is avoided ($ttl \rightarrow +\infty$). This observation is reinforced by the average clustering coefficient trend shown in Fig. 9: lower ttl values produce more compact community structures.

Our experiments show that interactions' ttls deeply affect the outcome of the algorithm. In particular:

- higher ttl values produce bigger communities and foster the stabilization of the node memberships;
- lower ttl values produce smaller, denser, and often more unstable communities.

[13] FB07 and CG show similar behaviors.

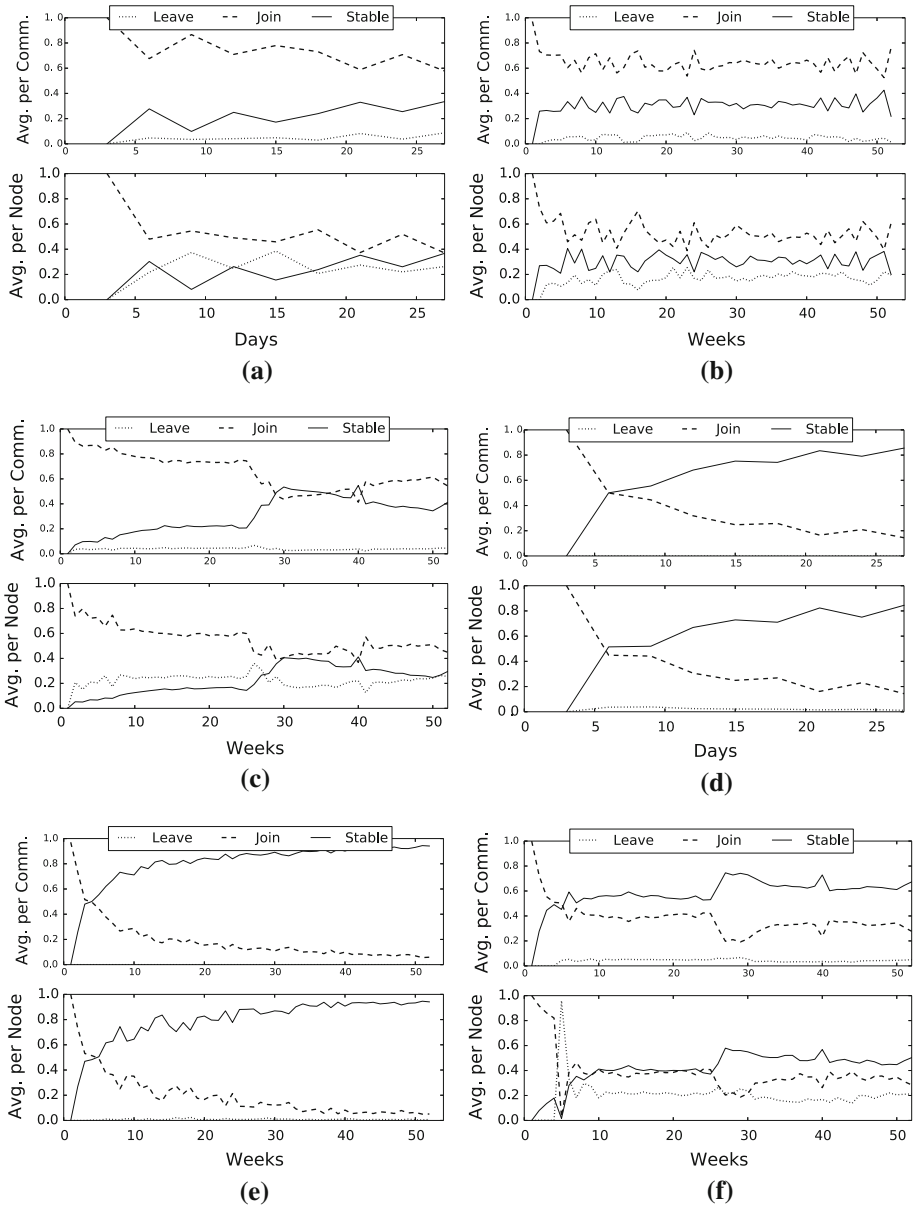

Fig. 13 Evolution of community memberships as time goes by. **a–c** scenarios for low ttl: CG one day, FB07 one week, WEIBO one week; **d–f** scenario for high ttl: CG and FB07 no removal, WEIBO one month

We can argue that reasonable values for this threshold are the ones which lead to a stability rate trend (for both nodes and communities) that overcome the join/leave ones. When this condition is satisfied, TILES is able to extract communities having stable life-cycles (e.g., they do not appear and fall apart quickly). However, the choice of ttl is obviously context dependent and it reasonable to assume that different phenomena can be characterized by different interactions persistence.

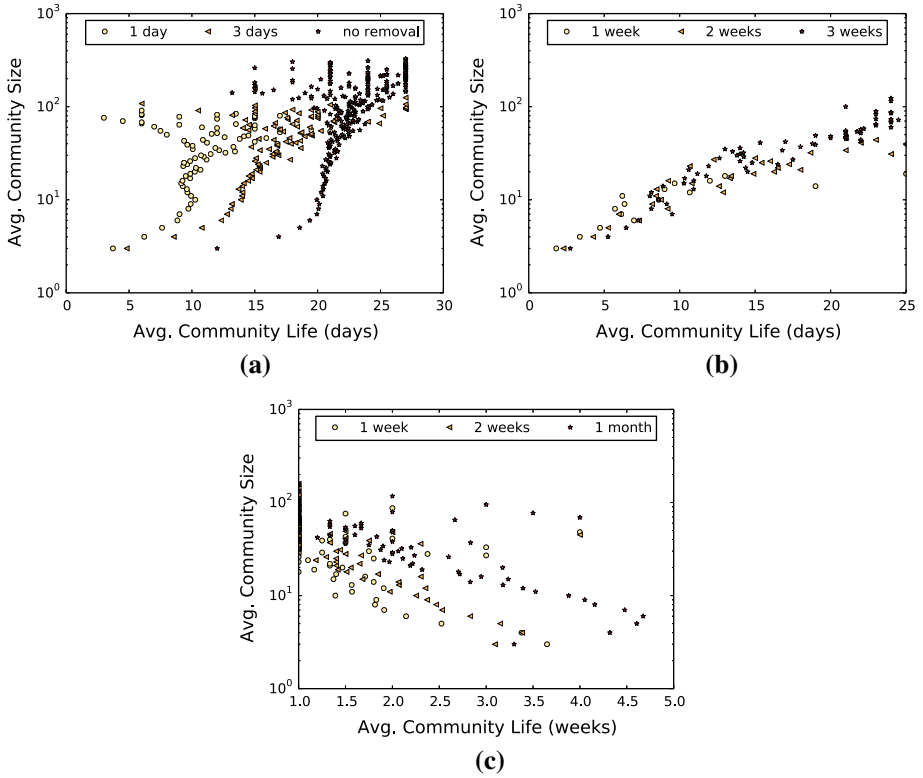

Fig. 14 Community size versus average community life for **a** CG, **b** FB07 and **c** WEIBO networks. Each *marker* identifies a different *ttl* value

6 Conclusion and future works

In this paper we proposed TILES, a community discovery algorithm which tracks the evolution of overlapping communities in dynamic social networks. It follows a "domino" approach: each new interaction determines the re-evaluation of community memberships for the endpoints and their neighborhoods. TILES defines two types of community memberships: peripheral membership and core membership, the latter indicating nodes involved in at least a triangle within the community. An interesting property of TILES is compositionality, which allows for algorithm parallelization, thus speeding up the computation of the communities. Other interesting characteristics emerged by the application of the algorithm on large-scale real-world dynamic networks, such as the skewed distribution of TILES community size and their high average clustering coefficient. Compared with other community detection algorithms both on synthetic and real networks, TILES shows better execution times and a higher correspondence with the ground truth communities. Moreover we shown how our approach enables the identification of the main events regulating the community life-cycle (i.e., birth, merge, split and death).

Many lines of research remains open for future works, such as identifying a more complex and precise way to manage the removal phase: indeed, one limit of the current approach is the needs of defining explicitly a time to leave threshold that is the same for all the interactions

among the nodes within the social network. To overcome this issue we plan to define a data driven approach able to dynamically provide an estimate of the expected persistence for each single interaction. Moreover, the mechanisms which regulate the node transitions from the *periphery* to the *core* of a community is another interesting aspect we propose to investigate: once fully understood it can be exploited as predictive information in a link prediction scenario, or used to explore how the transition of nodes from periphery to core can affect the spreading of information over the network.

Acknowledgments This work is partially Funded by the European Community's H2020 Program under the funding scheme "FETPROACT-1-2014: Global Systems Science (GSS)", Grant agreement #641191 CIMPLEX *"Bringing CItizens, Models and Data together in Participatory, Interactive SocialL EXploratories"*, https://www.cimplex-project.eu. This work is supported by the European Community's H2020 Program under the scheme "INFRAIA-1-2014-2015: Research Infrastructures", Grant agreement #654024 *"SoBigData: Social Mining & Big Data Ecosystem"*, http://www.sobigdata.eu.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

Asur, S., Parthasarathy, S., & Ucar, D. (2009). An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD), 3*(4), 16.

Bagrow, J. P., & Lin, Y.-R. (2012). Mesoscopic structure and social aspects of human mobility. *PLoS ONE*, *7*(5), e37676.

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286.5439, 509–512.

Bhat, S., & Abulaish, M. (Aug 2013). Overlapping social network communities and viral marketing. In *International Symposium on Computational and Business Intelligence*, pp. (243–246).

Boden, B., Günnemann, S., Hoffmann, H., & Seidl, T. (2012). Mining coherent subgraphs in multi-layer graphs with edge labels. In *ACM SIGKDD*.

Boldrini, C., Conti, M., & Passarella, A. (2011). From pareto inter-contact times to residuals. *Communications Letters IEEE*, *15*(11), 1256–1258.

Buehrer, G., & Chellapilla, K. (2008). A scalable pattern mining approach to web graph compression with communities. *Proceedings of the 2008 International Conference on Web Search and Data Mining, WSDM '08* (pp. 95–106). New York.

Burt, R. S. (1987). Social contagion and innovation: Cohesion versus structural equivalence. *American Journal of Sociology*.

Burt, R. S. (2000). Decay functions. *Social Networks*, *22*(1), 1–28.

Cazabet, R., Amblard, F., & Hanachi, C. (2010). Detection of overlapping communities in dynamical social networks. In *SocialCom*, (pp. 309–314).

Chakrabarti, D., Kumar, R., & Tomkins, A. (2006). Evolutionary clustering. *ACM SIGKDD*.

Clauset, A., Newman, M. E. J., & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E, 70*(6), 066111.

Coscia, M., Giannotti, F., & Pedreschi, D. (2011). A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining, 4*(5), 512–546.

Coscia, M., Rossetti, G., Pedreschi, D., & Giannotti, F. (2012). Demon: a local-first discovery method for overlapping communities. In *ACM SIGKDD*.

Dhouioui, Z., & Akaichi, J. (2014). Tracking dynamic community evolution in social networks. In *ASONAM*.

Folino, F., & Pizzuti, C. (2014). An evolutionary multiobjective approach for community discovery in dynamic networks. *IEEE Transactions on Knowledge and Data Engineering, 26*(8), 1838–1852.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports, 486*(3), 75–174.

Goh, K.-I., & Barabási, A.-L. (2008). Burstiness and memory in complex systems. *EPL (Europhysics Letters)*, *81*(4), 48002.

Goldberg, M., Magdon-Ismail, M., Nambirajan, S., & Thompson, J. (2011). Tracking and predicting evolution of social communities. *PASSAT*.

Guo, C., Wang, J., & Zhang, Z. (2014). Evolutionary community structure discovery in dynamic weighted networks. *Physica A: Statistical Mechanics and its Applications, 413*, 565–576.

Kostakos, V. (2009). Temporal graphs. In *Physica A: Statistical Mechanics and its Applications*.

Lancichinetti, A., & Fortunato, S. (2009). Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E, 80*(1), 016118.

Lancichinetti, A., Fortunato, S., & Kertész, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, *11*(3), 033015.

Lee, P., Lakshmanan, L., & Milios, E. (2014). Incremental cluster evolution tracking from highly dynamic network data. In *ICDE*.

Leskovec, J., Kleinberg, J. M., & Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *ACM SIGKDD*.

Lin, Y., Chi, Y., & Zhu, S. (2008). Facetnet: A framework for analyzing communities and their evolutions in dynamic networks. In *WWW*.

Nguyen, M. V. (2012). Community evolution in a scientific collaboration network. *CEC IEEE*.

Nguyen, N. P., Dinh, T. N., Xuan, Y., & Thai, M. T. (2011). Adaptive algorithms for detecting community structure in dynamic social networks. In *IEEE INFOCOM*, (pp. 2282–2290).

Nowell, L., & Kleinberg, J. (2003). The link prediction problem for social networks. In *CIKM*.

Palla, G., Barabási, A. L., & Vicsek, T. (2007). Quantifying social group evolution. *Nature, 446*(7136), 664–667.

Palla, G., Derényi, I., Farkas, I., & Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature, 435*(7043), 814–818.

Passarella, A., Conti, M., Boldrini, C., & Dunbar, R.I. (2011). Modelling inter-contact times in social pervasive networks. In *Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, (pp. 333–340). ACM.

Qi, G., Aggarwal, C. C., & Huang, T. S. (2013). Online community detection in social sensing. *WSDM*.

Rinzivillo, S., Mainardi, S., Pezzoni, F., Coscia, M., Giannotti, F., & Pedreschi, D. (2012). Discovering the geographical borders of human mobility. *KI - Künstliche Intelligenz*, *26*(3), 253–260.

Rossetti, G., Guidotti, R., Pennacchioli, D., Pedreschi, D., & Giannotti, F. (2015). Interaction prediction in dynamic networks exploiting community discovery. In *Proceedings of the 2015 ACM/IEEE International Conference on Advances in Social Network Analysis and Mining*.

Rossetti, G., Pappalardo, L., & Rinzivillo, S. (2016). A novel approach to evaluate community detection algorithms on ground truth. In *7th Workshop on Complex Networks*, Studies in Computational Intelligence. Springer-Verlag.

Rossetti, G., Pappalardo, L., Kikas, R., Pedreschi, D., Giannotti, F., & Dumas, M. (2015). Community-centric analysis of user engagement in skype social network. In *Proceedings of the 2015 ACM/IEEE International Conference on Advances in Social Network Analysis and Mining*.

Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences, 105*(4), 1118–1123.

Rozenshtein, P., Tatti, N., & Gionis, A. (2014). Discovering dynamic communities in interaction networks. *ECML PKDD*.

Shang, J., Liu, L., & Xie, F. (2012). A real-time detecting algorithm for tracking community structure of dynamic networks. *6th SNA-KDD*.

Sun, Y., Tang, J., Han, J., Gupta, M., & Zhao, B. (2010). Community evolution detection in dynamic heterogeneous information networks. *MLG*.

Takaffoli, M., Rabbany, R., & Zaiane, O. R. (2014). Community evolution prediction in dynamic social networks. In *ASONAM*.

Takaffoli, M., Sangi, F., Fagnan, J., & Zaïane O. (2011). Modec-modeling and detecting evolutions of communities. *ICWSM*.

Viswanath, B., Mislove, A., Cha, M., & Gummadi, P. K. (2009). On the evolution of user interaction in facebook. *WOSN*.

Wang, P., Gonzàlez, M. C., Hidalgo, C.A., & Barabási, A. L. (2009). Understanding the spreading patterns of mobile phone viruses. *Science, 324*(5930), 1071–1076.

Wu, X., & Liu, Z. (2008). How community structure influences epidemic spread in social networks. *Physica A Statistical Mechanics and its Applications*, *387*, 623–630.

Xu, H., Wang, Z., & Xiao, W. (2013). Analyzing community core evolution in mobile social networks. In *SocialCom*.

Zakreweska, A., & Bader, D. (2015). A dynamic algorithm for local community detection in graphs. In *ASONAM*.