Soc. Netw. Anal. Min. (2016) 6:15
DOI 10.1007/s13278-016-0325-1

ORIGINAL ARTICLE

Dynamic community detection in evolving networks using locality modularity optimization

Mário Cordeiro[1] · **Rui Portocarrero Sarmento**[2] · **João Gama**[2]

Received: 8 September 2015 / Revised: 3 March 2016 / Accepted: 5 March 2016 / Published online: 24 March 2016

Abstract The amount and the variety of data generated by today's online social and telecommunication network services are changing the way researchers analyze social networks. Facing fast evolving networks with millions of nodes and edges are, among other factors, its main challenge. Community detection algorithms in these conditions have also to be updated or improved. Previous state-of-the-art algorithms based on the modularity optimization (i.e. Louvain algorithm), provide fast, efficient and robust community detection on large static networks. Nonetheless, due to the high computing complexity of these algorithms, the use of batch techniques in dynamic networks requires to perform network community detection for the whole network in each one of the evolution steps. This fact reveals to be computationally expensive and unstable in terms of tracking of communities. Our contribution is a novel technique that maintains the community structure always up-to-date following the addition or removal of nodes and edges. The proposed algorithm performs a local modularity optimization that maximizes the modularity gain function only for those communities where the editing of nodes and edges was performed, keeping the rest of the network unchanged. The effectiveness of our algorithm is demonstrated with the comparison to other state-of-the-art community detection algorithms with respect to Newman's Modularity, Modularity with Split Penalty, Modularity Density, number of detected communities and running time.

Keywords Dynamic community detection · Large-scale networks analysis · Modularity · Evolving networks

✉ Mário Cordeiro
mario.miguel.cordeiro@gmail.com; pro11001@fe.up.pt

Rui Portocarrero Sarmento
jgama@fep.up.pt

João Gama
rui.p.sarmento@inescporto.pt

[1] FEUP, Porto University, R. Dr. Roberto Frias s/n, 4200-465 Porto, Portugal

[2] LIAAD - INESC TEC, FEP, Porto University, R. Dr. Roberto Frias 378, 4200-465 Porto, Portugal

1 Introduction

Community Detection is one of the most important tasks in social network analysis. Previously developed algorithms have been used mainly for static network data [4]. Very fast heuristic-based algorithms to optimize the graph communities modularity function are nowadays able to handle and discover communities in larger static networks [1]. With one of these algorithms, the Louvain Method, it was possible to find communities in a very efficient way in several large-scale static networks: Twitter social network dataset with 2.4M nodes and 38M links [14]; LinkedIn social network dataset with 21M nodes [8]; Mobile phone network with 4M nodes and 100M links [6]. In this work, we modified the original Louvain algorithm [1] to fulfill the requirements of a community detection task in evolving large-scale networks. When compared with original Louvain, the proposed modifications revealed to be conclusive about the gains in terms of modularity and speed. The supremacy of the algorithm was also confirmed by comparing results with three other algorithms (LabelRank [18], LabelRankT [16], GANXiSw [20] [19] and AFOCS [11]). In our work, we managed to provide the following contributions to dynamic community detection problem:

Optimization: Our algorithm only performs community detection in the full network at the first snapshot. In the following snapshots, the previous community structure is taken into account by applying the Louvain in a much smaller aggregated network;

Efficiency: Empirical results have demonstrated that the proposed dynamic version keeps most of the communities of the previous snapshot unchanged. The modified Louvain steps will only compute communities which are affected by addition or removal of nodes/edges. Small network's changes resulted in better elapsed time executions. Just two or three iterations are enough to reduce the full network to an equivalent smallest one by several orders of magnitude;

Stability: Unlike the static Louvain, that may change significantly the way communities are obtained between snapshots (two runs in the snapshot lead to different solutions due its non-determinism). In the proposed algorithm, unaffected communities keep unchanged. Therefore, they preserve the same nodes and even the same community id between snapshots. Community stability is important because it will simplify the process of tracking of communities over time.

2 Related work

Frequently, the Louvain algorithm [1] is used in dynamic network community detection by performing individual runs of the algorithm in snapshots of the network. This approach apart from being computationally inefficient makes it almost impossible to track communities in a fine-grained way. The algorithm is non-deterministic and produces a solution for the communities that is very unstable between separate runs on the snapshots. The community detection work referenced in a 2010 survey [4] was later complemented by an incremental community detection algorithm based on modularity and proposed in [15]. The algorithm based on the principles of events in the life of communities (growth, contraction, merging, splitting, birth and death) defined by [13] calculates the modularity gain in each one of the iterations to detect and track communities over time in incremental networks. This algorithm only considers the addition of new edges, describing a two-step approach in detecting static communities. In the first step, it uses the Louvain method, then applies an incremental number of up-to-date strategies to track the dynamic communities. The algorithm is very dependent on the initial community structure of the network. Another algorithm, the QCA [12], was presented as being a fast and adaptive algorithm that provides efficient identification of the community structure of dynamic social networks. This algorithm allows the addition and removal of nodes and edges. Starting with the initial communities calculated with the Louvain method, the algorithm applies adaptive node community changes by considering each node as an autonomous agent demonstrating flocking behavior toward their preferable neighboring groups [21]. In a following work, the same authors proposed AFOCS [11], a new community detection algorithm for dynamic networks. This algorithm shares the same principles of QCA being only modified to allow the possibility of detection of overlapping communities. A detailed comparison between QCA and AFOCS was presented in [11]. Due to the unavailability of the source code of QCA, in the present work only AFOCS is being considered for evaluation purposes. Label propagation techniques and specifically speaker–listener label propagation (SLPA) was used in community detection over large networks. LabelRank [20] and GANXiSw [20] [19] used the SLPA technique to perform static network community detection while LabelRankT [16] was designed to handle dynamic networks. The main focus of those algorithms is overlapping community detection. Nonetheless, all of them have a non-overlapping mode. In our work, the non-overlapping mode was considered the reference. The algorithms presented better performance for low overlapping density networks [17].

3 Preliminaries

3.1 Notation

The undirected unweighted graph that represents a network with N nodes and M links is given by $G = (V, E)$. The components of the graph are: the node set V, which is just a list of indices; the edge set E where each edge consists of two vertices. The undirected graph has $n = |V|$ nodes, $V = \{u_1, u_2, \ldots, u_n\}$, and $e = |E|$ edges, $E = \{(i_1, j_1), (i_2, j_2), \ldots, (i_e, j_e)\}$. For each node u, d_u is its respective degree. The disjoint set of communities of the graph is given by $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, where each $C_i \in \mathcal{C}$ is a community of G. The community to which the node u is assigned is given by $c = N2C(u)$, $TOT(c)$ is the sum of all the degrees of the nodes that belong to the community c, $IN(c)$ the number of inner loops of the community c and $w = NEIG(c)$ the set of adjacent communities. During the course of the paper we will use the following nomenclature: Steps, refer to individual steps of the Louvain Algorithm. Iterations, refer to the

iterations of the Louvain algorithm (repeated till no increase in modularity is possible). Operations reflect the kind of changes that could be applied to the communities of the network (e.g. merging or splitting communities). Procedures are main tasks of the proposed algorithm.

3.2 Modularity function

We used a modularity measure $\mathcal{Q}$ to evaluate the quality of the community structure of a graph. Modularity serves as the objective function during the process of calculating the communities [10]. This measure, apart from being the most widely used [2, 3], was considered as the quality measure used in the evaluation of the algorithms. Higher values for the modularity $\mathcal{Q}$ mean better community structures. Therefore, the objective is to find a community assignment for each node in the network such that $\mathcal{Q}$ is maximized using the modularity function defined by

$$\mathcal{Q} = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \quad (1)$$

A_{ij} represents the weight of the edge between i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ-function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise and $m = \frac{1}{2}\sum_{ij} A_{ij}$. To calculate the modularity of a specific community, the number of inner edges (*in* [n]) and the total number of edges (*tot*[n]) of a specific node n is used. The modularity of the full network can be calculated using the previous $\mathcal{Q}$ function, by considering all the entries of *in* and *tot* for all the nodes.

3.3 Representing communities

Using Fig. 1c as an example, the network is defined using a list of the degrees for each one of the seven nodes (pairs *node* $\rightarrow$ *degree*): $degrees = \{(1,2),(2,2),\ldots,(6,2)\}$, with size $|n2c| = 7$, the list of edges that form the network (pairs *src* $\rightarrow$ *dest*): $links = \{(1,2),(1,3),\ldots,(6,7)\}$ and the weights of each one of the edges (pairs *edge* $\rightarrow$ *weight*): $weights = \{(1,1),(1,1),\ldots,(6,1)\}$, with size $|links| = |weights| = 16$. The association between node community is given by: $n2c = \{(1,3),(2,3),\ldots,(6,7)\}$, the number of inner edges for each one of the communities being $in = \{(1,0),(2,0),(3,6),(5,2),(4,0),(7,2),(6,0)\}$, and the total number of edges *in* and *out* of the community: $tot = \{(1,0),(2,0),(3,7),(5,5),(4,0),(7,4),(6,0)\}$, where $|n2c| = |in| = |tot| = 7$. The same can be extended to the Fig. 1d network: $n2c = \{(3,3),(5,5),(7,7)\}$, $in = \{(3,6),(5,2),(7,2)\}$, $tot = \{(3,7),(5,5),(7,4)\}$, $degrees = \{(3,2),(5,3),(7,2)\}$, $links = \{(3,3),(3,5),\ldots,(7,7)\}$, $weights = \{(3,6),(3,1),\ldots,(7,2)\}$. Communities are represented by dark gray nodes and identified by the higher order of all the nodes that belong to the community. Nodes belonging to the same community are being placed inside the same area.

3.4 Introduction to the Louvain algorithm

Figures 1 and 2 describe the sequences of the Louvain algorithm to perform community detection in two example networks. The algorithm uses a greedy optimization method that attempts to optimize the modularity of a partition of the network in successive iterations. It is non-deterministic, this means that multiple runs on the same network may lead to distinct final community structures. Communities are calculated by maximizing the objective function in a two-step optimization in each one of the iterations. In the first step (step 1), small communities are formed by optimizing the modularity locally. Only local changes of communities are allowed in this step. In the following step (step 2), nodes belonging to the same community are aggregated in a single node that represents a community in a new aggregated network of communities. Iteratively

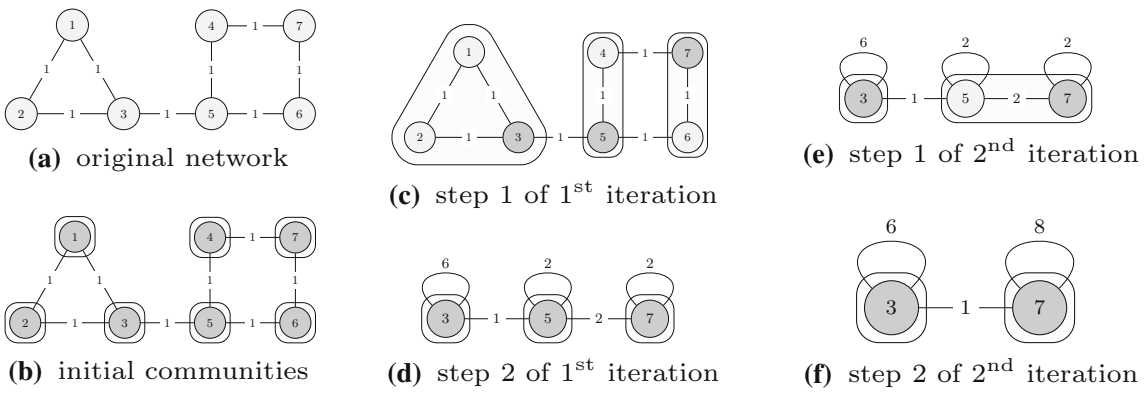

Fig. 1 The original Louvain algorithm steps

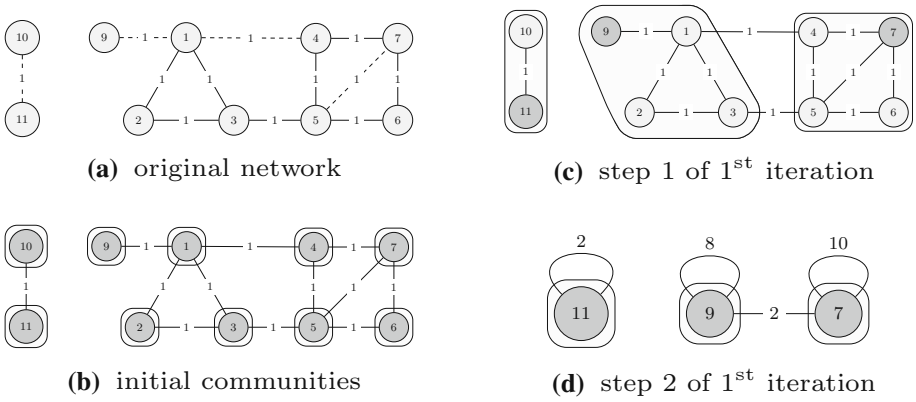

Fig. 2 Steps for Fig. 1a with edges {(1, 4), (5, 7), (1, 9), (10, 11)} added

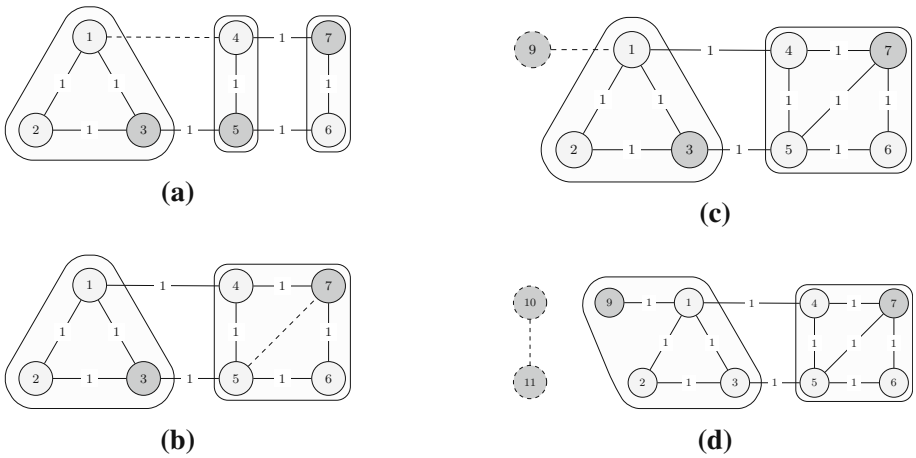

Fig. 3 Types of new edges that can be added to or removed from a graph

these steps are repeated until no increase in modularity is possible with a hierarchy of communities being produced: Fig. 1a shows the initial network; Fig. 1b represents initial individual node communities; Fig. 1c exemplifies local modularity optimization after first step; Fig. 1d symbolizes community aggregation results and the new initial communities; Figs. 1e and 1f are the two Louvain steps where the local modularity optimization and community aggregation for the second iteration are presented; the algorithm stops at the 2nd iteration since no increase in modularity is possible. Figure 2 shows the network in Fig. 1 with a set of edges being added ($A = \{(1,4),(5,7),(1,9),(10,11)\}$). The new edges as well new nodes 9, 10 and 11 are marked using dashed lines in the initial network (Fig. 2a). Once the original algorithm does not allow the addition or removal of new nodes and edges after obtaining the community structure, a re-computation of the communities starting from an initial network is necessary. Figure 2c presents the initial communities. In the first step (step 1) modularity is optimised by allowing only local changes of communities (Fig. 2d). In the following step (step 2), found communities are aggregated to build a new network of communities (Fig. 2d). The optimal solution is obtained right after the first iteration, i.e. no increase in modularity was possible with additional community structure changes.

3.5 From static to dynamic networks

To enable true dynamic graph community detection, the Louvain Method described in Sect. 3.4 was modified to support incremental community structure changes when nodes are added or removed from the network. Regarding incremental networks, networks that only support the addition of new nodes and edges, four types of operations to communities were already defined [15]:

Operation 1 (`Op1`): keeps the community structure unchanged;
Operation 2 (`Op2`): merges two communities into one;
Operation 3 (`Op3`): assigns nodes to an existing community;
Operation 4 (`Op4`): creates a new community with new nodes.

Figure 3 shows the four types of new edges that can be added to a graph. Full dynamic community detection is achieved by extending the algorithm to support the scenarios

Table 1 Effects at community structure level that result from addition of edges/nodes

Edges	Structure	Description
Add	Cross-community edge `Op1` or `Op2`	The two nodes incident to the edge already exist and belong to different communities (Fig. 3a). Because the increased edge is a cross-community edge, which means source and destination nodes belong to two different communities; therefore, operation `Op1` or `Op2` can take place. The former keeps the community structure unchanged, while the latter combines both communities into one
	Inner community edge `Op1`	The two nodes incident to the edge already exist and belong to the same community (Fig. 3b). Adding a new edge between two nodes belonging to the same community increases the inner connections of the community and maintains the inter-community connections unchanged, which coincides with the basic principle of the modularity increase, so `Op1` will be taken resulting in an unchanged community structure
	Half-new edge `Op3` or `Op4`	In this case, the increased edge is a half-new edge, which means that one of the nodes already exists in the network while the other is new (Fig. 3c). Operation `Op3` or `Op4` can be taken, `Op3` assigns the new node to the existent community, `Op4` creates a new community with the new node depending on the gain of modularity of both settings
	New edge `Op4`	Both of the nodes incident to the edge are new (Fig. 3d). In this case two of `Op4`'s results could take place, assigning both nodes to the same new community, or creating two individual communities for each one of the nodes

of the problem regarding the removal of edges as defined in Fig. 3. The four operations need to be considered:

Operation 5 (`Op5`): keeps the community structure unchanged;

Operation 6 (`Op6`): splits a community in two or more communities.

Operation 7 (`Op7`): removes a terminal node from an existing community;

Operation 8 (`Op8`): removes an existing community formed by two nodes.

In terms of community structure, `Op2` and `Op8` reduce the number of communities while `Op4` and `Op6` increase it. `Op1`, `Op3`, `Op5` and `Op7` actually do nothing on the community structure. Table 1 exhibits the effects at community structure level and respective possible operations when edges are being added to the graph. Table 2 presents equivalent information for the cases where edges are being removed.

4 Local modularity optimization

Algorithm 1 presents the pseudo-code of the proposed dynamic community detection algorithm. The algorithm input parameters are the initial network ($G = (V, E)$) and the list of edges to be added and removed from the graph during the iterations (A and R respectively). For storing the community information the algorithms use two internal community networks: the lower level network ($\mathcal{C}_{\text{ll}}$) where the original network is maintained, and the upper level network ($\mathcal{C}_{\text{ul}}$) where the aggregated community network is stored. Section 4.1 describes the roles of each one of the networks. The algorithm main procedure (line 4) handles the main flow and the several subprocedures for specific algorithm tasks. These tasks' subprocedures are separated into two types. The subprocedures type that does not change the network and are used to get data from both the lower level and upper level network (e.g. `CommunityChangedNodes()`, line 11), and subprocedures that update the networks in terms of edges or nodes and/or community assignment (eg. `DisbandCommunities()`, line19 or line 26). The task procedures of the algorithm are conceptual and aggregate subprocedures to complete specific tasks (i.e. adding edge to the $\mathcal{C}_{\text{ll}}$). They are repeated until a modularity increase is possible or edges to be removed or added.

Procedure P1a: Adding edge to $\mathcal{C}_{\text{ll}}$ consists in the retrieval of a list of affected nodes and respective communities by the addition of an edge (line 17) and by the addition of the edge itself to $\mathcal{C}_{\text{ll}}$ (line 18);

Procedure P1b: Removal of edge in the $\mathcal{C}_{\text{ll}}$. This procedure consists in the retrieval of a list of affected nodes and respective communities when the removal of an edge is performed (line 24), and by the removal of the edge itself to $\mathcal{C}_{\text{ll}}$ (line 25);

Procedure P2: Disband Affected Communities in $\mathcal{C}_{\text{ll}}$. Based on the list of affected nodes and respective communities retrieved by `AffectedByAddition()` or `AffectedByRemoval()` the affected communities will be disbanded in $\mathcal{C}_{\text{ll}}$ (line 19 or line 26);

Procedure P3: Update the $\mathcal{C}_{\text{ul}}$ with changes of $\mathcal{C}_{\text{ll}}$. The list of affected nodes and respective communities retrieved by `AffectedByAddition()` or `AffectedByRemoval()` will be also used to replicate the changes in community structure to the $\mathcal{C}_{\text{ul}}$ (line 20 or line 27). Notice that in this procedure, the added

Table 2 Effects at community structure level that result from removal of edges/nodes

Edges	Structure	Description
Remove	Cross-community edge `Op5`	The two nodes incident to the removed edge belong to different communities (Fig. 3a). Removing this type of edges maintains the inner connections of the community and decreases the inter-community connections, modularity of the two communities will increase. This operation will never result in the merging of existent communities neither will it disband any of the communities where the nodes linked by the removed edge belong. In this case `Op5` will occur resulting in an unchanged community structure
	Inner community edge `Op5` or `Op6`	The two nodes incident to the edge belong to the same community (Fig. 3b). Removing this type of edges decreases the inner connections of the community but preserves the inter-community connections unchanged. The decreasing of the inner modularity may lead to two distinct operations: `Op5` by maintaining the community structure unchanged or `Op6` where the disbanding of the community results in smaller communities or the joining of parts of the disbanded community to other existent communities with better individual modularity
	Edge to isolated node `Op7`	One of the nodes incident to the edge is an isolated node, removing this edge implies removing the isolated node (Fig. 3c). This operation will not affect the community structure because the removed node is a terminal node, and therefore it will not change the inner connections of the community. Meaning that `Op7` should take place
	Edge between isolated nodes `Op8`	The edge to be removed belongs to two isolated nodes. Removing this edge implies removing both nodes (Fig. 3d) extinguishing the community or communities that those nodes belong to by applying `Op8`. The other communities will remain unaffected

or removed edges will also be updated in the $\mathcal{C}_{\text{ul}}$;

Procedure P4: $\mathcal{C}_{\text{ul}}$ will be used to perform the Louvain Algorithm Step 1 and calculate the changes in the community structure that may lead to a locally optimised modularity (line 10);

Procedure P5: Update $\mathcal{C}_{\text{ll}}$ with the communities that changed by applying the Louvain Algorithm Step 1 to $\mathcal{C}_{\text{ul}}$ (line 12);

Procedure P6: Use the $\mathcal{C}_{\text{ul}}$ to perform the Louvain Algorithm Step 2 and aggregate communities (line 14).

Algorithm 1 Dynamic Community Detection Algorithm

```
1: V ← {u_1, u_2, .., u_v} , E ← {(i_1, j_1), (i_2, j_2), .., (i_e, j_e)}
2: A ← array{(i_1, j_1), .., (i_m, j_m)}
3: R ← array{(i_1, j_1), .., (i_n, j_n)}
4: procedure MAIN(G ← (V, E), A, R)
5:     C_ll ← {C_1, C_2, .., C_n}, C_ul ← {}, C_aux ← C_ll
6:     INITPARTITION(C_aux)
7:     mod ← MODULARITY(C_aux), old_mod ← 0
8:     m ← 1, n ← 1
9:     while (mod ≥ old_mod ∨ m ≤ |A| ∨ v ≤ |R|) do
10:        C_aux ← ONELEVEL(C_aux)
11:        ⟨n, c⟩ ← COMMUNITYCHANGEDNODES(C_ll, C_aux)
12:        C_ll ← UPDATECOMMUNITIES(C_ll, n, c)
13:        old_mod ← mod, mod ← MODULARITY(C_ll)
14:        C_ul ← PARTITIONTOGRAPH(C_ll)
15:        if m ≤ |A| then
16:            ⟨src, dest⟩ ← A[m]
17:            a_nodes ← AFFECTEDBYADDITION(src, dest, C_ll)
18:            C_ll ← ADDEDGE(src, dest, C_ll)
19:            C_ll ← DISBANDCOMMUNITIES(C_ll, a_nodes)
20:            C_ul ← SYNCCOMMUNITIES(C_ll, C_ul, a_nodes)
21:        end if
22:        if n ≤ |R| then
23:            ⟨src, dest⟩ ← R[n]
24:            a_nodes ← AFFECTEDBYREMOVAL(src, dest, C_ll)
25:            C_ll ← REMOVEEDGE(src, dest, C_ll)
26:            C_ll ← DISBANDCOMMUNITIES(C_ll, a_nodes)
27:            C_ul ← SYNCCOMMUNITIES(C_ll, C_ul, a_nodes)
28:        end if
29:        C_aux ← C_ul, m ← m + 1, n ← n + 1
30:    end while
31: end procedure
```

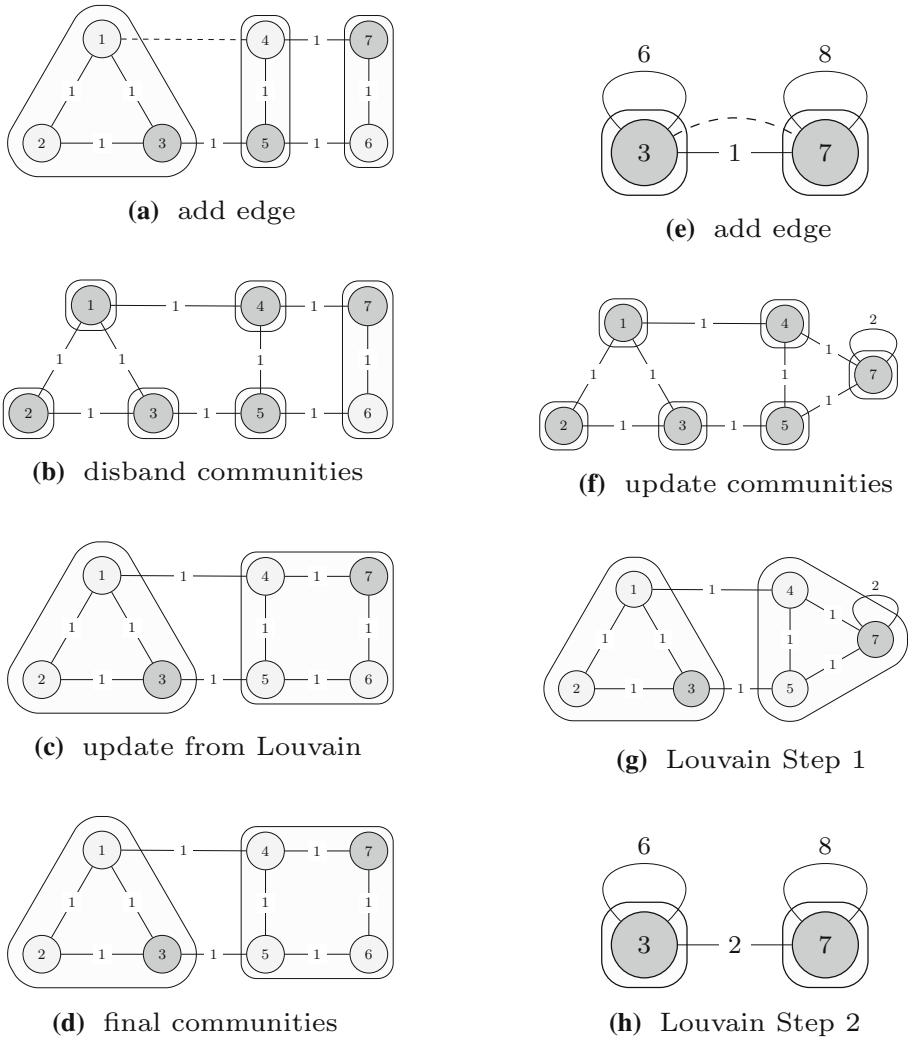

Fig. 4 Add cross-community edges (1–4)

4.1 Process

The addition and removal of edges start after a first iteration of the Louvain algorithm, the sequence of the algorithm when adding or removing edges is given by the following figure list: begins with Fig. 4a where the addition or removal of nodes and edges is done using `P1a`. Figure 4b follows with the affected communities being disbanded to individual node communities using `P2`. To maintain $\mathcal{C}_{\text{ul}}$ up-to-date, all the changes in the $\mathcal{C}_{\text{ll}}$ should be replicated in the upper level network as exemplified in Fig. 4f. Figure 4g follows, where the Louvain algorithm step 1 is applied in `P4`, Fig. 4c exhibits the $\mathcal{C}_{\text{ll}}$ being updated with the new community assignment using `P5`, the last Fig. 4h represents the final $\mathcal{C}_{\text{ul}}$ after the Louvain step 2 is executed using `P6`. This network will be used in the next algorithm iteration (Louvain iteration or addition/removal of edges). Graphs of Figs. 4d, e are only represented as auxiliary figures once they are not being used in any of the algorithm steps. The process described may be seen as a real example of how the dynamic community detection algorithm works. The process starts with the initial network of Fig. 1a after the first Louvain iteration being performed. The network and community assignments of Fig. 1c will be used as the $\mathcal{C}_{\text{ll}}$ and Fig. 1d as the $\mathcal{C}_{\text{ul}}$. The following eight steps will explain the addition of four new edges ($A = (1,4),(5,7),(1,9),(10,11)$) and the removal of other four ones ($R = (1,4),(5,7),(1,9),(10,11)$). The outcome of each one of the steps will be the starting point for the next one. After adding and removing the same edges, the original network and respective community structure are recovered. Summing up, the algorithm maintains two networks, the lower level network ($\mathcal{C}_{\text{ll}}$) responsible for maintaining all the original nodes and final communities. Additionally, the upper level network ($\mathcal{C}_{\text{ul}}$) used to perform the iterations of the Louvain algorithm using smaller representation of $\mathcal{C}_{\text{ll}}$. Synchronization tasks between $\mathcal{C}_{\text{ul}}$ and $\mathcal{C}_{\text{ll}}$ are mandatory in each increment. In both networks, the

aggregated communities are represented by the node with the highest order number (i.e. a community of nodes 1, 3, 4, 7 and 9 is represented by community 9). $\mathcal{C}_{ul}$ follows the standard steps and iterations of the Louvain algorithm but requires to be updated when the addition and removal of nodes occur (performed in $\mathcal{C}_{ll}$). On the other hand, $\mathcal{C}_{ll}$ should be updated every time node community assignments change. Both $\mathcal{C}_{ll}$ and $\mathcal{C}_{ul}$ should be kept synced regarding the assignment of nodes to communities (community structure) and regarding network structure (nodes and edges). In these network figures, left stack of graphs represents the evolution of the lower level network ($\mathcal{C}_{ll}$) and final communities, right stack represents the upper level network ($\mathcal{C}_{ul}$) containing the aggregated network used by each iteration of the Louvain algorithm. Figures 4, 5, 6, 7, 8, 9, 10 and 11 exhibit the process of updating dynamically a network showing one example per type of edge described in Sect. 3 for both addition and removal of edges. Each one of the steps will identify the operations defined in Sect. 3.5 and the corresponding task procedures used in each one of the operations. Lower level network is displayed in the left side, upper level on the right:

Add cross-community edge: steps involved in the addition of an edge between nodes belonging to different communities (Fig. 4). The community detection could lead to two different operations as defined in Sect. 3.5. The figure exhibits the two communities affected by the addition of the edges maintained unchanged (`Op1`), but cases resulting in the merge of the two communities (`Op2`) could also occur. `P1a` adds a new edge between nodes 1 and 4 to the $\mathcal{C}_{ll}$ (Fig. 4a). Node 1 belongs to community 3 while node 4 belongs to community 5. The addition of this edge affects all nodes of communities 3 and 5 as presented in the $\mathcal{C}_{ul}$ (Fig. 4e); `P2` disbands all affected communities at the $\mathcal{C}_{ll}$ with nodes forming individual node communities (Fig. 4b). `P3` updates the $\mathcal{C}_{ul}$ with all the original individual node communities, this network will be used as basis for a new Louvain iteration (Fig. 4f); `P4` outcome of the local modularity optimization (Louvain step 1) on the $\mathcal{C}_{ul}$ (Fig. 4g); `P5` updates the $\mathcal{C}_{ll}$ with resulting new initial communities obtained by local modularity optimization (step 1) done at the $\mathcal{C}_{ul}$ in the previous operation (Fig. 4c); `P6` aggregates communities (Louvain step 2) at the $\mathcal{C}_{ul}$ (Fig. 4h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges; `Final Communities` displayed at the $\mathcal{C}_{ll}$ (Fig. 4d and $\mathcal{C}_{ul}$ (Fig. 4h);

Add inner community edge: steps involved in the addition of an edge between nodes belonging to the same community (Fig. 5). This example will always result in an unchanged community structure as described in the `Op1` in Sect. 3.5. `P1a` adds a new edge between node 5 and node 7 to the $\mathcal{C}_{ll}$ (Fig. 5a). Nodes 5 and 7 belong to the same community 7. $\mathcal{C}_{ul}$ displays that the addition of this edge affects only nodes of community 7 (Fig. 5e); `P2` adds edges to nodes of the same community, increases the modularity

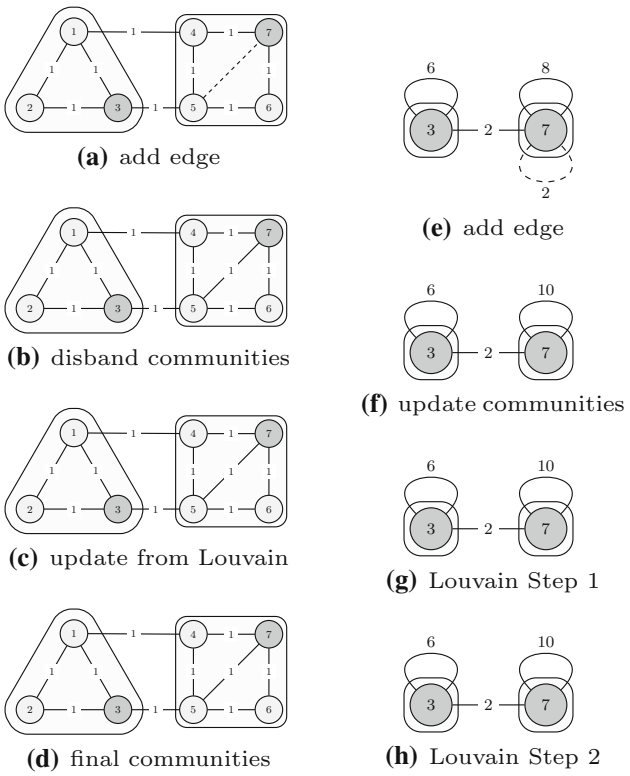

Fig. 5 Add inner community edges (5–7)

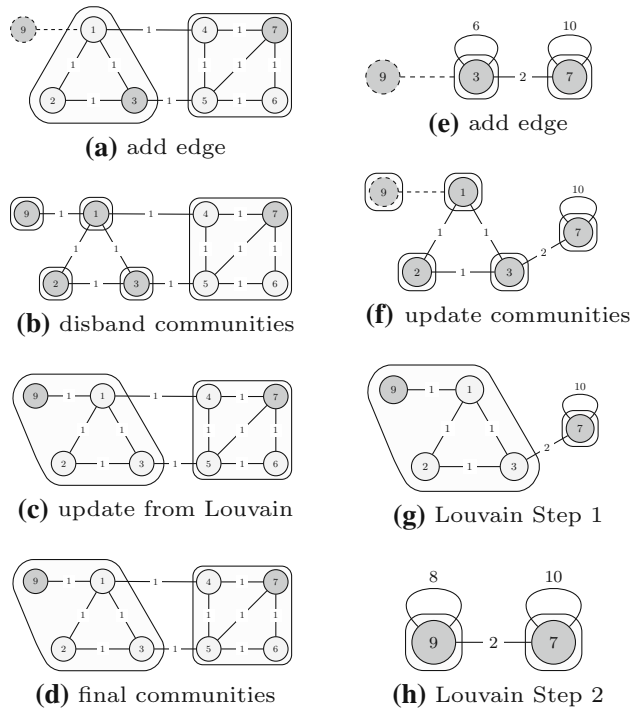

Fig. 6 Add half-new edges (1–9)

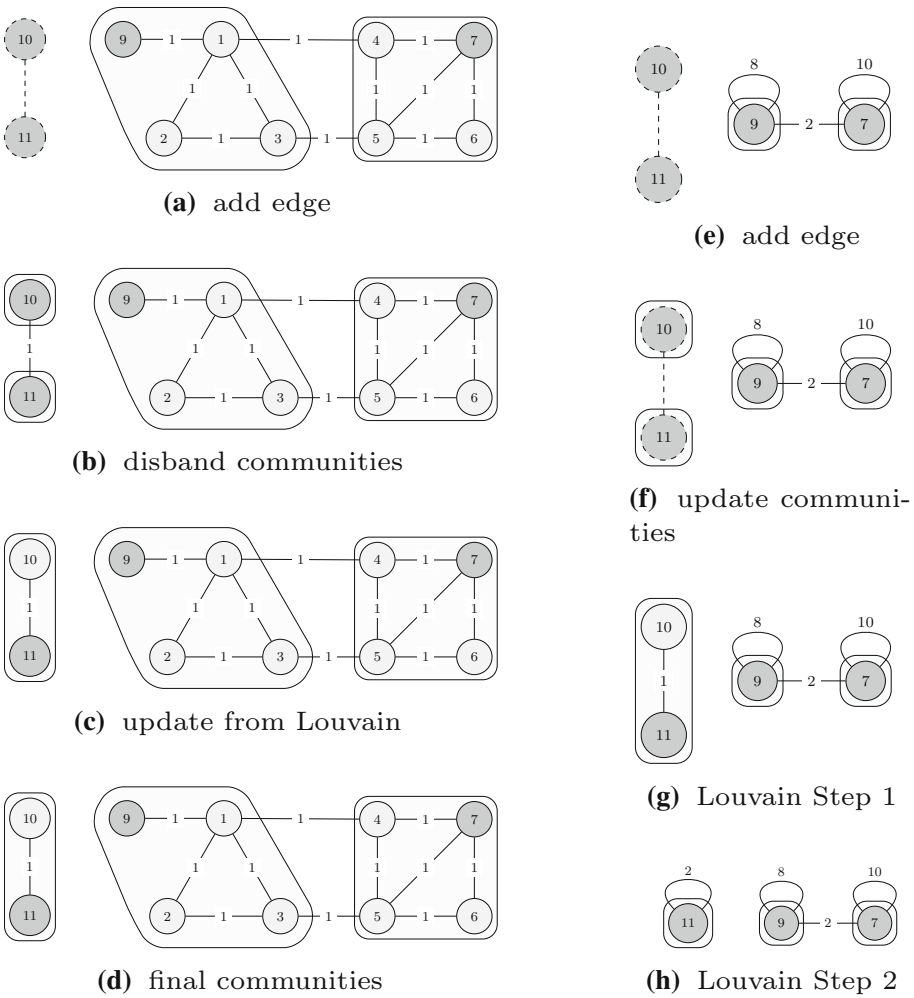

Fig. 7 Add new nodes edges (10–11)

of the community, so it is expected that the community structure of the community 7 will be kept unchanged (Fig. 5b). `P3` only updates the $\mathcal{C}_{\text{ul}}$ with the new weight for the community 7, the weight increase reflects the addition of the inner edge (Fig. 5f). The resulting $\mathcal{C}_{\text{ul}}$ will be used in the next iteration of the Louvain algorithm; in `P4` no community structure changes after the local modularity optimization (Louvain step 1) (Fig. 5g); `P5` does not need to update the $\mathcal{C}_{\text{ll}}$ with the resulting new initial communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ (Fig. 5c); In `P6`, the aggregation of communities with the Louvain algorithm (step 2), performed in the $\mathcal{C}_{\text{ul}}$ did not change the community structure (Fig. 5h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges;

Add half-new edge: steps involved in the addition of an edge between a new node and an existing node belonging to an existent community (Fig. 6). This example is an `Op3` as described in Sect. 3.5. `P1a` adds a new edge between node 1 and node 9 to the $\mathcal{C}_{\text{ll}}$ (Fig. 6a). Node 1 belongs to community 3 and node 9 is a new node. $\mathcal{C}_{\text{ul}}$ shows that the addition of this edge affects only nodes of community 3 (Fig. 6e); `P2`, taking into account that the addition of this edge could change the community structure of community 3, will disband the communities at the $\mathcal{C}_{\text{ll}}$, with nodes of the affected communities forming individual node communities (Fig. 6b). `P4` updates the $\mathcal{C}_{\text{ul}}$ with the community structure changes introduced in the $\mathcal{C}_{\text{ll}}$ (Fig. 6f) being the input network for the next iteration of the Louvain algorithm; `P3` outcome of the local modularity optimization (Louvain step 1) (Fig. 6g). All the four nodes form a single community; `P5` updates the $\mathcal{C}_{\text{ll}}$ with resulting new initial communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ (Fig. 6c); `P6` aggregates communities (Louvain step 2) in the $\mathcal{C}_{\text{ul}}$ (Fig. 6h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges;

Add new nodes edge: steps involved in the addition of an edge between two new nodes (Fig. 7). This example is an `Op4` as described in Sect. 3.5. `P1a` adds a new edge between nodes 10 and 11 to the $\mathcal{C}_{\text{ll}}$ (Fig. 7a). Nodes 10 and 11 are new nodes and do not belong to any current

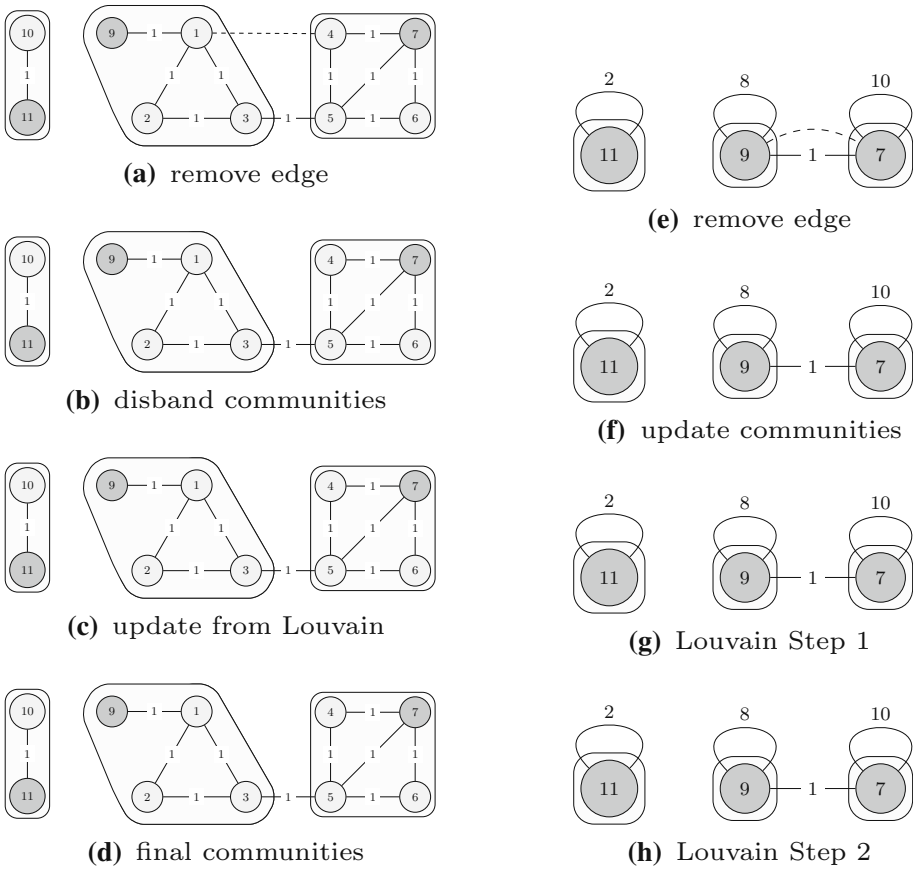

Fig. 8 Remove inter-community edges (1–4)

community. $\mathcal{C}_{\text{ul}}$ exhibits that the addition of this edge does not affect any of the current communities (Fig. 7e); `P2` maintains the community structure unchanged for all the other nodes and communities (Fig. 7a). In `P3`, $\mathcal{C}_{\text{ul}}$ is only updated to add the two new nodes and respective edge (Fig. 7f) serving as input network for the next iteration of the Louvain algorithm; only nodes 10 and 11 had their community structure changed after the local modularity optimization (Louvain step 1) done in `P4`, forming a new community 11 (Fig. 7g); `P5` updates nodes 10 and 11 of the $\mathcal{C}_{\text{ll}}$ with the resulting new initial communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ (Fig. 7c); `P6` aggregates communities (Louvain step 2) in the $\mathcal{C}_{\text{ul}}$ (Fig. 7h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges;

Remove inter-community edge: steps involved in the removal of an edge between two nodes belonging to different communities (Fig. 8). This is an example of `Op5` as described in Sect. 3.5, the community structure will be maintained unchanged though the local modularity of both communities will increase. `P1b` removes an edge between node 1 and node 4 to the $\mathcal{C}_{\text{ll}}$ (Fig. 8a). Node 1 belongs to community 9, node 4 belongs to community 7. The $\mathcal{C}_{\text{ul}}$ displays that the removal of this edge only affects communities 9 and 7 (Fig. 8e); In `P2`, there is no need to disband communities at the $\mathcal{C}_{\text{ll}}$. Removing edges between inter-communities increases the modularity of each one of the communities 9 and 7 (Fig. 8b). `P3` does not need to update the $\mathcal{C}_{\text{ul}}$ with the community structure changes of the $\mathcal{C}_{\text{ll}}$, only the weights of the edge between communities 9 and 7 need to be decreased to reflect the removal of the edge (Fig. 8f). The next iteration of the Louvain algorithm uses $\mathcal{C}_{\text{ul}}$; in `P4`, the outcome of the local modularity optimization (Louvain step 1) presents that communities 9 and 7 remained separated (Fig. 8g), there was no gain in modularity by joining any of the communities; `P5` updates the $\mathcal{C}_{\text{ll}}$ with the resulting new initial communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ (Fig. 8c); `P6` aggregates communities (Louvain step 2) in $\mathcal{C}_{\text{ul}}$ (Fig. 8h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges;

Remove inner community edge: steps involved in the removal of an edge between nodes belonging to the same community (Fig. 9). The removal of the edge could lead to two distinct operations as described in Sect. 3.5. `Op5` in case the removal of the edge decreases significantly the modularity of the community, or `Op6` if the edge removal results in the splitting of the community into two or more communities. `P1b` removes an edge between node 5 and 7 to $\mathcal{C}_{\text{ll}}$ (Fig. 9a). Nodes 5 and 7 belong to community 7. $\mathcal{C}_{\text{ul}}$

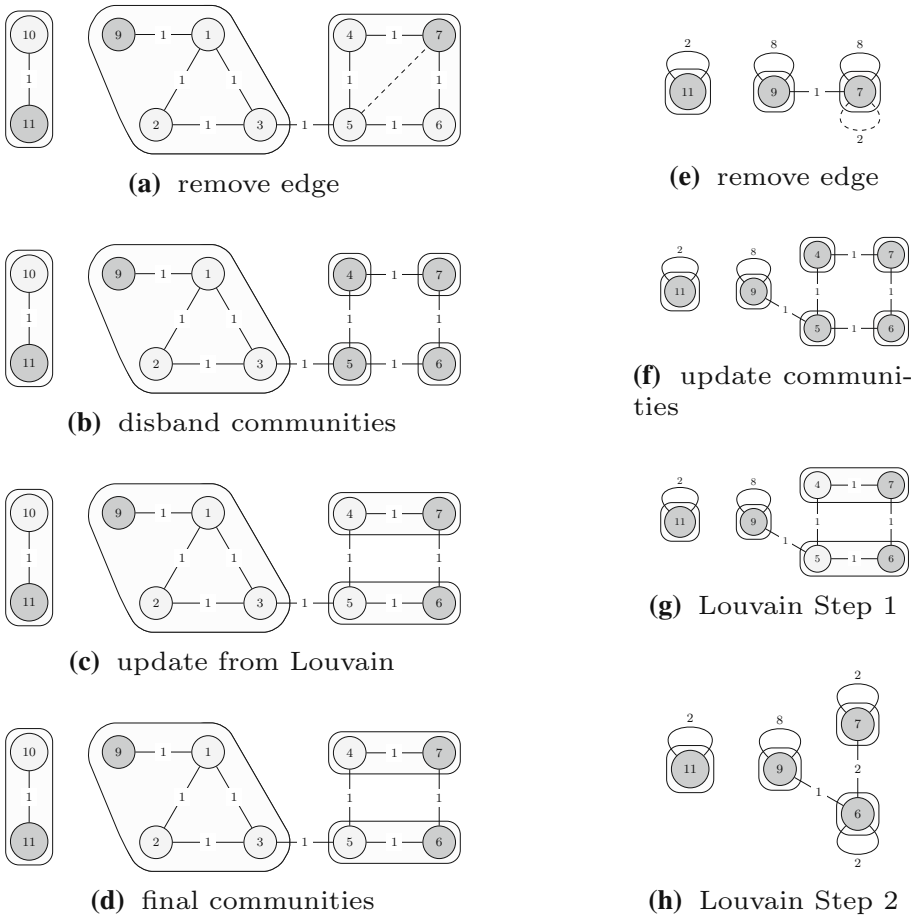

Fig. 9 Remove inner community edges (5–7)

displays that the removal of this edge affects only nodes of community 7 (Fig. 9e); now that the edge is between nodes belonging to the same community, the community structure can change to an unpredictable community structure. In `P2`, all nodes belonging to the affected communities will form individual node communities (Fig. 9b). In `P3`, the $\mathcal{C}_{\text{ul}}$ is updated with the community structure changes of the $\mathcal{C}_{\text{ll}}$ (Fig. 9f) and will be used as input for the next Louvain iteration; `P4` outcome of the local modularity optimization (Louvain step 1). The four nodes formed two separated communities, community 6 and 7 (Fig. 9g); `P5` updates the $\mathcal{C}_{\text{ll}}$ with resulting new initial communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ (Fig. 9c); `P6` aggregates communities (Louvain step 2) in the $\mathcal{C}_{\text{ul}}$ (Fig. 9h). This network will serve as a basis for a new Louvain iteration or a new addition/removal of edges;

Remove edge to terminal node: steps involved in the removal of an edge between a terminal node inside an existing community (Fig. 10). This is an example of `Op7` in Sect. 3.5. `P1b` removes edge between node 1 and 9 to the $\mathcal{C}_{\text{ll}}$ (Fig. 10a). Node 1 and node 9 belong to the same community. $\mathcal{C}_{\text{ul}}$ displays that the removal of this edge affects only nodes of community 9 (Fig. 10e); note that the removal of this edge may change the community structure of community 9 and needs to reflect the removal of node 9, the disbanding of the community 9 will be performed at the $\mathcal{C}_{\text{ll}}$ by `P2`. All nodes belonging to the affected community will form individual node communities (Fig. 10b). `P3` updates the $\mathcal{C}_{\text{ul}}$ with the community structure changes of the $\mathcal{C}_{\text{ll}}$ (Fig. 10f). $\mathcal{C}_{\text{ul}}$ will be used for the next iteration of the Louvain algorithm; `P4` outcome of the local modularity optimization (Louvain step 1) (Fig. 10g). It must be remarked that in this step a community 3 was formed with nodes 1, 2 and 3 and community 7 with aggregated nodes 6 and 7; `P5` updates the $\mathcal{C}_{\text{ll}}$ with resulting new initial communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ (Fig. 10c); `P6` aggregates communities (Louvain step 2) in the $\mathcal{C}_{\text{ul}}$ (Fig. 10h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges;

Remove edge between isolated nodes: steps involved in the removal of an edge between two nodes of an isolated community (Fig. 11). This is an example of `Op8` in Sect. 3.5. `P1b` removes an edge between node 10 and node 11 to the $\mathcal{C}_{\text{ll}}$. Nodes 10 and 11 are two isolated nodes that, themselves, form a community with only two nodes (Fig. 11a). $\mathcal{C}_{\text{ul}}$ indicates that the removal of this edge and respective nodes affects only the community 11 (Fig. 11e); in `P2`, once the edge is between two isolated nodes, the community structure of all the other communities will

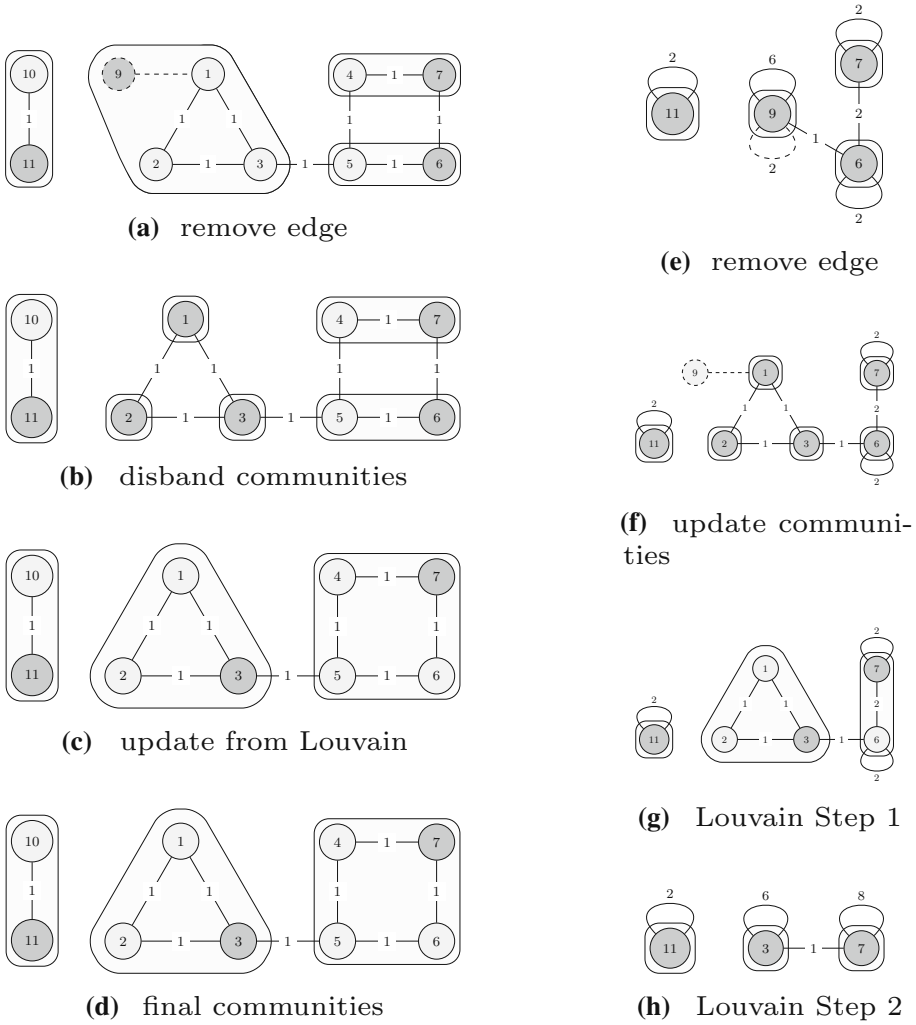

Fig. 10 Remove edge to terminal nodes (1–9)

remain unchanged (Fig. 11a). In P3 the $\mathcal{C}_{\text{ul}}$ will only be updated to remove the two nodes and respective edge (Fig. 11f) and will be the input network for the next Louvain iteration; because there was no increase of modularity by joining communities 3 and 7, the community structure has not changed after the local modularity optimization (Louvain step 1) done in P4 (Fig. 11g; In P5, there was no need to update the communities obtained by local modularity optimization (step 1) in the $\mathcal{C}_{\text{ul}}$ since they remain unchanged (Fig. 11c); in P6, no community aggregation of the Louvain algorithm (step 2) was performed in the $\mathcal{C}_{\text{ul}}$ (Fig. 11h). This network will serve as basis for a new Louvain iteration or a new addition/removal of edges.

5 Experimental analysis

The Dynamic Louvain algorithm was evaluated in incremental and dynamic network setups, Sects. 5.2.1 and 5.2.2, respectively. Results for incremental networks were obtained for the high-energy physics theory citation network [9], using the original Louvain, Dynamic Louvain, LabelRank, LabelRankT, GANXiSw and AFOCS. The original Louvain algorithm served as a baseline. As the original Louvain and LabelRank only support static networks, their results reflect the full network runs with added edges at each increment (snapshots). In this setup , snapshot size varies from 1, 2, 5, 10 and 20 edges randomly added in each increment (i.e. no timestamp information was used). In a second incremental network setup, the same high-energy physics theory citation network dataset was used, but now with snapshots built by aggregating timestamps of citations in a monthly basis. Evaluation was performed only for algorithms Dynamic Louvain, LabelRankT, GANXiSw and AFOCS.

In the dynamic network setup, the Autonomous systems AS-733 dataset [9] was used to obtain results for the three algorithms that support dynamic community detection (Dynamic Louvain, LabelRankT and AFOCS) and GANXiSw that does not support dynamic community detection.

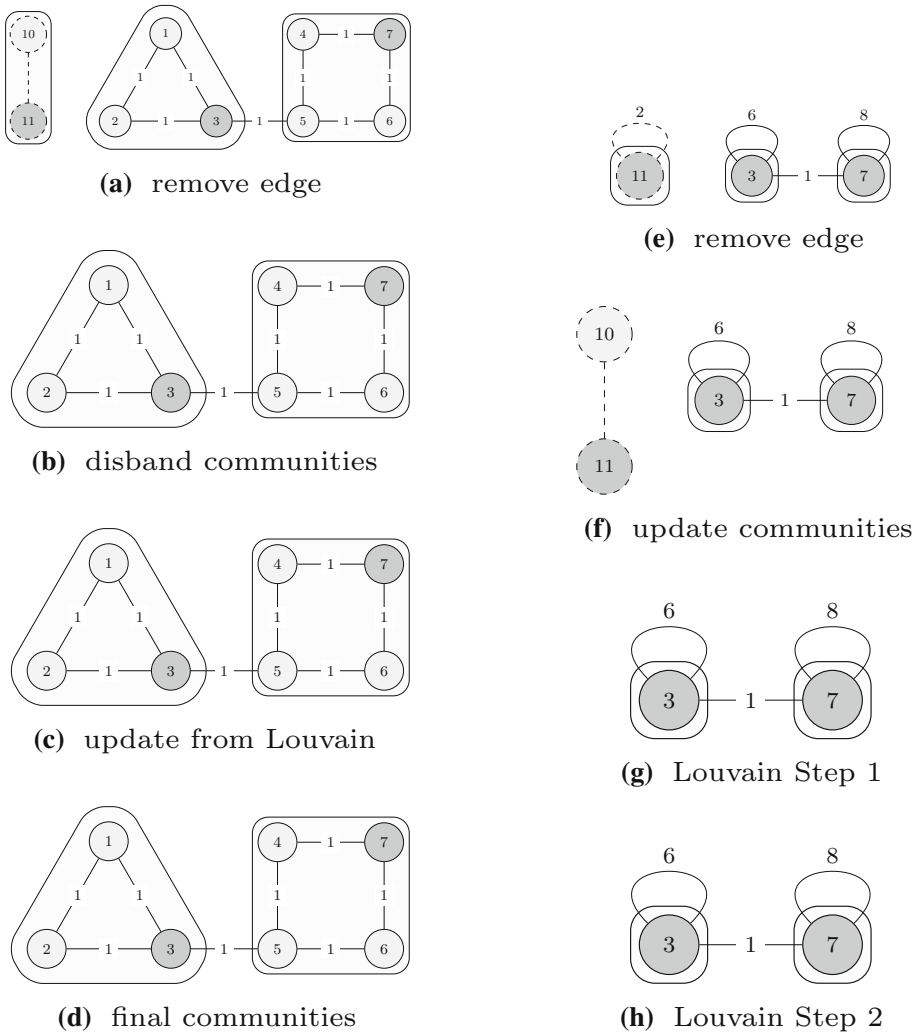

Fig. 11 Remove edge between isolated nodes (10–11)

The empirical evaluation was done by comparing run times of each increment (duration of each increment and cumulative execution time), the size of the network (number of nodes and edges) and three measures of the quality of the community structure: modularity ($\mathcal{Q}$), modularity with split penalty ($\mathcal{Q}_s$) and modularity density ($\mathcal{Q}_{ds}$) detailed in Sect. 5.1. The number of detected communities and the number of nodes that are not assigned to any community are also included in the evaluation.

The hardware used was a Intel (R) Core (TM) i7-4702MQ processor computer with 8 GBytes and SSD HDD. Three runs per algorithm/dataset were performed with values presented in graphs as the average values of each one of those 3 runs.

5.1 Measuring quality of network community structure

Modularity is one the most used metrics to measure the strength of the community structure found by community detection algorithms [2, 3]. Algorithms, like the Louvain, use the modularity as the objective function during the process of calculating the communities. Two opposite yet coexisting problems of modularity maximization are known: in some cases, it tends to favor small communities over large ones while in others favor large communities over small ones showing an resolution limit problem [5]. A new measure called Modularity Density [2, 3] was proposed as an improved measurement of the community quality compared to modularity. This new measure takes into account the two problems described before. In addition because not all of the evaluated algorithms maximize the modularity, it was decided to include an analysis covering the following measures of Quality of Network Community Structure:

Newman's modularity: For unweighted and undirected networks, modularity is defined as the ratio of difference between the actual and expected number of edges within the community. Already presented in Sect. 3.2, for the given community partition of a network $G = (V, E)$, with $|E|$ edges, modularity ($\mathcal{Q}$) is given by:

$$\mathcal{Q} = \sum_{c_i \in C} \left[\frac{\left|E_{c_i}^{\text{in}}\right|}{|E|} - \left(\frac{2\left|E_{c_i}^{\text{in}}\right| + \left|E_{c_i}^{\text{out}}\right|}{2|E|} \right)^2 \right] \tag{2}$$

with C being the set of all communities, c_i is a specific community i of C, $\left|E_{c_i}^{\text{in}}\right|$ the number of edges within community c_i, and $\left|E_{c_i}^{\text{out}}\right|$ the number of edges from the nodes of community c_i to nodes that belong to nodes of other communities.

Modularity with split penalty: To address the drawback of favoring small communities, the quality of the community structure should take into account the edges between different communities [2, 3]. The Modularity with Split Penalty ($\mathcal{Q}_s$) is calculated by subtracting from modularity the split penalty ($\mathcal{SP}$) which is the fraction of edges that connects nodes of different communities:

$$\mathcal{SP} = \sum_{c_i \in C} \left[\sum_{\substack{c_i \in C \\ c_j \neq c_i}} \frac{\left|E_{c_i,c_j}\right|}{2|E|} \right] \tag{3}$$

where $\left|E_{c_i,c_j}\right|$ is the number of edges from community c_i to community c_j for unweighted networks. Therefore, by subtracting the split penalty given by Eq. 3 to the Newman's modularity equation 2, $\mathcal{Q}_s$ is given by:

$$\mathcal{Q}_s = \mathcal{Q} - \mathcal{SP} = \sum_{c_i \in C} \left[\frac{\left|E_{c_i}^{\text{in}}\right|}{|E|} - \left(\frac{2\left|E_{c_i}^{\text{in}}\right| + \left|E_{c_i}^{\text{out}}\right|}{2|E|} \right)^2 - \sum_{\substack{c_i \in C \\ c_j \neq c_i}} \frac{\left|E_{c_i,c_j}\right|}{2|E|} \right] \tag{4}$$

Modularity density: Both Newman's modularity ($\mathcal{Q}$) and modularity with split penalty ($\mathcal{Q}_s$) are still independent of the number of nodes in the communities as long as the number of edges is preserved. They still reveal the resolution limit problem, moreover, modularity with split penalty ($\mathcal{Q}_s$) makes this problem even worse [2, 3]. To address the above two shortcomings, [2, 3] introduced community density into modularity by incorporating both the number of edges and the number of nodes in the communities and also the Split Penalty. For undirected networks, modularity density ($\mathcal{Q}_s$) is defined by:

$$\mathcal{Q}_{ds} = \sum_{c_i \in C} \left[\frac{\left|E_{c_i}^{\text{in}}\right|}{|E|} d_{c_i} - \left(\frac{2\left|E_{c_i}^{\text{in}}\right| + \left|E_{c_i}^{\text{out}}\right|}{2|E|} d_{c_i} \right)^2 - \sum_{\substack{c_i \in C \\ c_j \neq c_i}} \frac{\left|E_{c_i,c_j}\right|}{2|E|} d_{c_i,c_j} \right] \tag{5}$$

with, d_{c_i} being the internal density of community c_i,

$$d_{c_i} = \frac{2\left|E_{c_i}^{\text{in}}\right|}{|c_i|(|c_i| - 1)} \tag{6}$$

and $d_{c_i c_j}$ the pair-wise density between community c_i and community c_j,

$$d_{c_i,c_j} = \frac{\left|E_{c_i,c_j}\right|}{|c_i||c_j|}. \tag{7}$$

5.2 Test scenario

Three test scenarios were considered in the evaluation of the algorithm. In Sect. 5.2.1 two incremental network setups where edges and nodes are added to an initial network, no removal of edges and nodes has been performed. In the first incremental scenario, no timestamped data were considered. In the second incremental scenario, the edges were added to the network according to their paper citation date. In the second scenario, Sect. 5.2.2, a fully dynamic setup was used, with edges and nodes being added and removed from the initial network.

5.2.1 *Incremental network setup*

The high-energy physics theory citation network dataset [9] is a medium-size undirected network with 27770 nodes and 352807 edges[1]. The results were obtained by performing runs with 99 increments for configurations of 1, 2, 5, 10 and 20 edges per increment. The initial network used for each configuration was the original network with an amount of TotalIncrements × Number Of Edges removed randomly from the initial network, i.e. for configuration with one edge per increment, 99 edges were randomly removed to be added during the process. Table 3 presents the initial network size in terms of initial edges and the number of edges added per configuration. Table 4 and Fig. 12 show the results comparison, for the original Louvain, LabelRank and GANXiSw, with full batch runs with initial full network plus added edges in the current increment and the results for the Dynamic Louvain, LabelRankT and AFOCS, used incremental runs, with the initial network being the one obtained from the previous increment with the new edges added.

Now an incremental network with timestamped data was evaluated. The timestamps of the nodes (paper submission time to Arxiv) of the high-energy physics theory citation network dataset [9] were used. Instead of adding an randomly chosen edge, the nodes and edges were added according to the submission date. Nodes were aggregated monthly (136 months) between January 1992 and April 2003. Each month represents a step. The initial network has 4 nodes (papers) and 2 edges (citations). Table 5 and Fig. 13 show the results obtained in this setup.

[1] http://snap.stanford.edu/data/cit-HepTh.html.

Table 3 Initial network size with total added edges

Initial edges	Increments	Edges per increments	Total of new edges
352708	99	1	99
352609	99	2	198
352312	99	5	495
351817	99	10	990
350827	99	20	1980

Table 4 Incremental network results (average): network size, duration (ms), modularity ($\mathcal{Q}$) and cumulative execution time (ms) for the high-energy physics theory citation network dataset

	Increment size				
	1	2	5	10	20
Louvain					
# Nodes	27,770	27,769	27,768	27,767	27,765
# Edges	704,511	704,411	704,111	703,615	702,620
Duration	286	288	281	286	284
Modularity ($\mathcal{Q}$)	0.5923	0.5922	0.5925	0.5928	0.5916
Dynamic Louvain					
# Nodes	1609	3914	9081	13,465	15,508
# Edges	38,819	100,938	242,207	370,043	442,364
Duration	230	1370	1855	2714	3553
Modularity ($\mathcal{Q}$)	0.6235	0.6279	0.5984	0.5819	0.5881
LabelRank					
# Nodes	27,770	27,768	27,767	27,766	27,764
# Edges	704,470	704,368	704,062	703,556	702,541
Duration	70,556	70,448	74,643	75,071	75,865
Modularity ($\mathcal{Q}$)	0.5976	0.5978	0.5729	0.5494	0.5312
LabelRankT					
# Nodes	27,770	27,767	27,766	27,766	27,762
# Edges	704,455	704,290	703,838	703,029	701,446
Duration	8775	9871	11,407	12,955	10,930
Modularity ($\mathcal{Q}$)	0.5980	0.5980	0.5280	0.5265	0.5183
GANXiSw					
# Nodes	27,770	27,768	27,767	27,766	27,764
# Edges	704,470	704,368	704,062	703,556	702,541
Duration	60,847	62,477	60,905	60,604	60,436
Modularity ($\mathcal{Q}$)	0.5666	0.5666	0.5656	0.5664	0.5646

5.2.2 *Dynamic network setup*

The Autonomous Systems AS-733 dataset [9] is a medium-size dynamic undirected network with a maximum of 6474 nodes and 13895 edges[2]. The dataset contains 733 daily instances (from November 8 1997 to January 2 2000), only the first 400 instances were used in the present evaluation. In contrast to the citation network, where nodes and edges only get added over time (not deleted), the AS-733 dataset exhibits the addition and deletion of the nodes and edges over time. Table 6 and Fig. 14 show the results for the Dynamic Louvain, LabelRankT, GANXiSw and AFOCS with network being initialised with the snapshot of November 8, 1997 and in each of the subsequent 399 steps, nodes and edges were added or removed accordingly to the respective day.

5.3 Comparison and evaluation of network size and execution time

Figure 12 presents execution times and network size for the different edge increments configurations ($k = \{1, 2, 5, 10, 20\}$). All algorithms, except the Dynamic Louvain, maintain the network size in their original size. The duration time for each increment is also constant. The variable network size of the Dynamic Louvain, always smaller than their counterparts, leads to an improved efficiency when adding new nodes and edges. Results indicated that the lower network size for each one of the steps was correlated with lower values for the duration. This effect is more noticeable for $k = 1$ and $k = 2$ where network size falls very sharply after a few increments. Apart from the communities affected by the additions of edges, the rest of the network is converging very fast to the modularity optimized communities. Incremental execution times are low because step 1 and step 2 of the Louvain algorithm operate in smaller networks (approx. 164 nodes with 249 edges). A slight degradation in the duration of some incremental steps was noticed for $k > 2$. Those peaks in duration are associated with increases in the size of the network due the disbanding of communities to the original nodes when adding cross-community edges (Fig. 4) or removing inner community edges (Fig. 9). Worse values occur when community disbanding affect the large communities with the duration maintained around static Louvain baseline. Consecutive merges and disbands of communities require a higher effort in the synchronization of the lower and upper level networks resulting in values of size closer to the full network and duration values above the ones obtained by the original Louvain. In the results for 10 or more edges, figures exhibit an approximation to the original Louvain. When adding a high number of edges, the probability of community disbanding in increments increases, leading the Dynamic Louvain algorithm to work upon the same

[2] http://snap.stanford.edu/data/as.html.

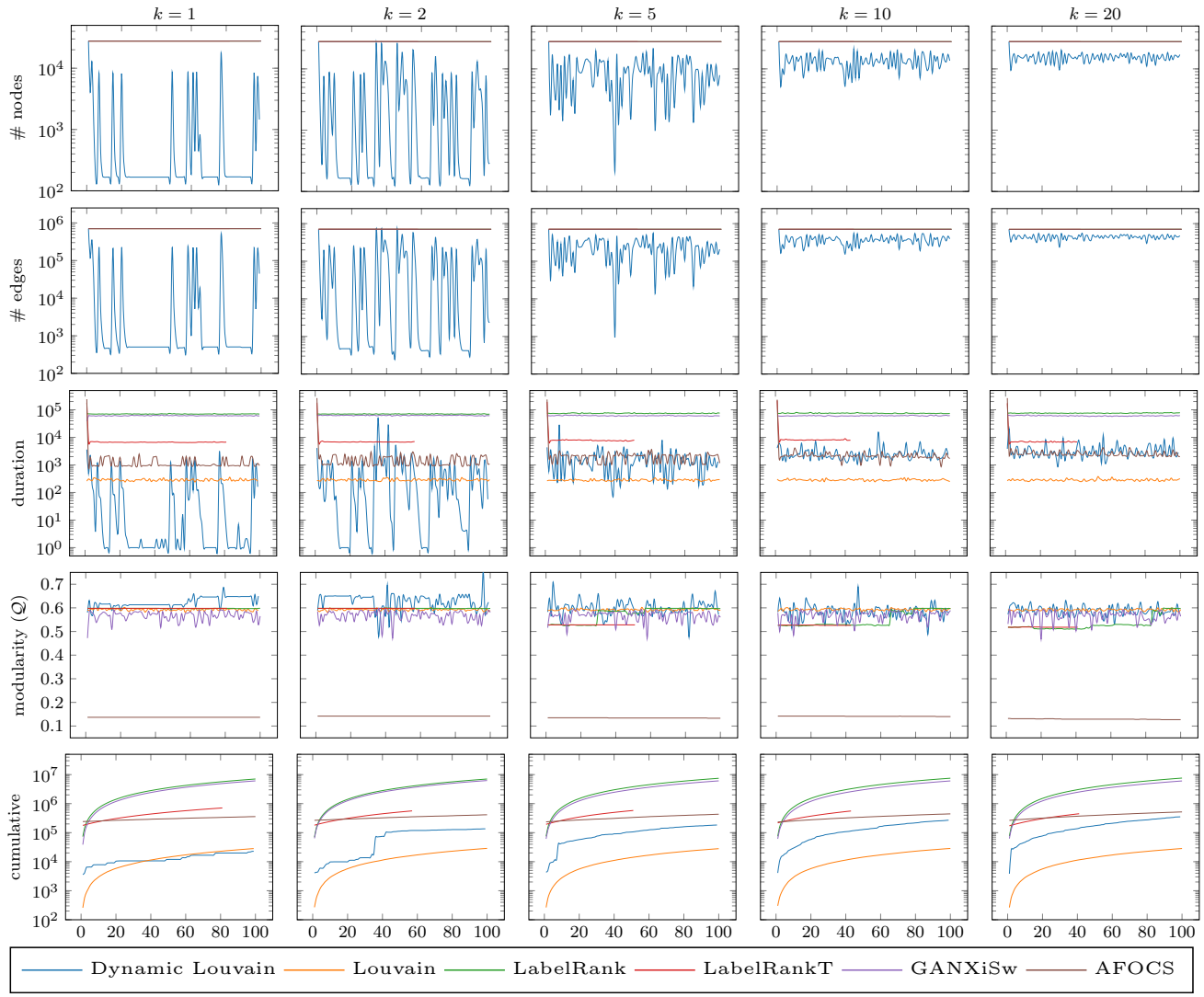

Fig. 12 Incremental network results: evolution of network size, increment duration (ms), average modularity and cumulative execution time (ms) for different edge increment configurations ($k = \{1, 2, 5, 10, 20\}$) for algorithms Louvain, Dynamic Louvain, LabelRank, LabelRankT, GANXiSw and AFOCS. *X*-axis show each one of the 100 increments. *Y* in log scale except for modularity

Table 5 Timestamped incremental network results (average): network size, duration (ms), cumulative execution time (ms), total communities, modularity ($\mathcal{Q}$), split penalty ($\mathcal{SP}$), modularity with split penalty ($\mathcal{Q}_s$), modularity density ($\mathcal{Q}_{ds}$) and unassigned nodes for the high-energy physics theory citation network dataset

	Dynamic Louvain	LabelRankT	GANXiSw	AFOCS
# Nodes	9541	11,871	11,784	11,784
# Edges	207,713	223,303	221,661	221,661
Duration	1072	7282	16,919	10,419
Cumulative	145,784	983,013	2,301,004	1,416,920
Total communities	431	507	498	309
Modularity ($\mathcal{Q}$)	0.6016	0.6441	0.6362	0.1952
Split penalty ($\mathcal{SP}$)	0.3562	0.1898	0.2187	0.5133
Modularity with split penalty ($\mathcal{Q}_s$)	0.2513	0.4543	0.4175	−0.3473
Modularity density ($\mathcal{Q}_{ds}$)	0.1109	0.1047	0.1141	0.0283
Unassigned nodes	118	31	0	3467

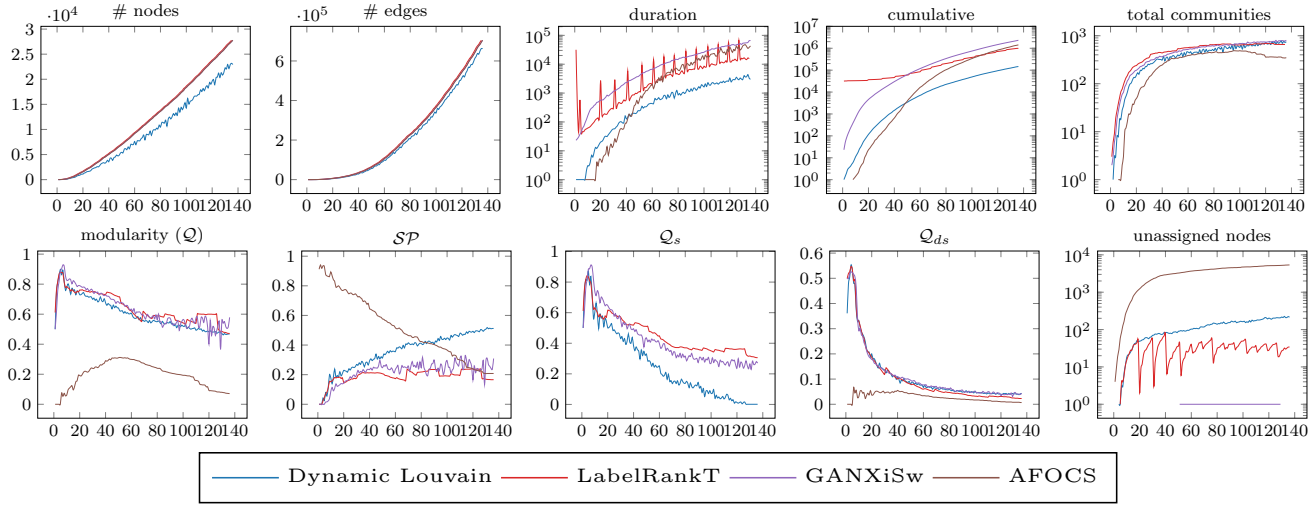

Fig. 13 Timestamped incremental network results: evolution of network size, increment duration (ms), average modularity and cumulative execution time (ms) for increment configurations of 1 month for algorithms Dynamic Louvain, LabelRankT, GANXiSw and AFOCS. *X*-axis show each one of the 136 increments. *Y* in log scale except for modularity

Table 6 Dynamic network results (average): network size, duration (ms), cumulative execution time (ms), total communities, modularity ($\mathcal{Q}$), split penalty ($\mathcal{SP}$), modularity with split penalty ($\mathcal{Q}_s$), modularity density ($\mathcal{Q}_{ds}$) and unassigned nodes for the Autonomous systems AS-733 dataset

	Dynamic Louvain	LabelRankT	GANXiSw	AFOCS
# Nodes	2811	3722	3718	3718
# Edges	11,756	13,423	13,407	13,407
Duration	55	549	1109	544
Cumulative	22,070	219,786	443,488	217,714
Total communities	224	156	150	84
Modularity ($\mathcal{Q}$)	0.5590	0.4337	0.4955	0.1210
Split penalty ($\mathcal{SP}$)	0.4010	0.2211	0.2281	0.7603
Modularity with split penalty ($\mathcal{Q}_s$)	0.1581	0.2127	0.2674	−0.6436
Modularity density ($\mathcal{Q}_{ds}$)	0.0821	0.0478	0.0565	0.0069
Unassigned nodes	37	8	0	6154

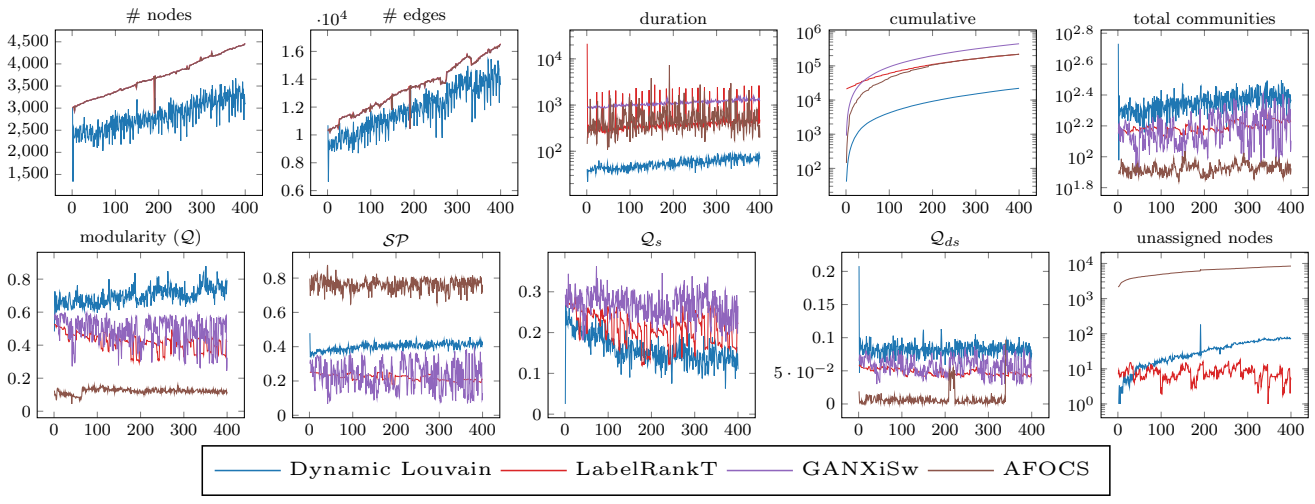

Fig. 14 Dynamic network results: evolution of network size, step duration (ms), average modularity and cumulative execution time (ms) for algorithms Dynamic Louvain, LabelRankT, GANXiSw and AFOCS

number of nodes and edges as the other algorithms. Nevertheless, the Dynamic Louvain exhibits lower execution times than LabelRank, LabelRankT, GANXiSw and AFOCS for all k values. Table 4 presents the average results in terms of network size and duration in each one of the runs for each one of the algorithms. In the timestamped incremental network results presented in Fig. 13 and Table 5, due to an increase of the number of edges added in each one of the increments, the Dynamic Louvain is dealing with a network almost of the same size of the original network. This effect reveals a penalty in the duration of each one the steps. Nevertheless Dynamic Louvain presents better always duration and cumulative execution times than LabelRankT and GANXiSw. When compared with AFOCS, the first 50 increments show worse duration and cumulative time performance, the algorithms quickly recovers after 50 increments and achieve 145.784 ms while AFOCS took 1.416.920 ms (aprox. 10× more). Figure 14 and Table 6 display the results for the incremental network setup. In the presented 400 steps, each one containing the addition and removal of nodes and edges, confirm the good performance of the Dynamic Louvain. In the first steps, the Dynamic Louvain drops the size of the network significantly when compared to the networks maintained by LabelRankT, GANXiSw and AFOCS. This allows the Dynamic Louvain to have lower increment duration times and final cumulative execution time than their counterparts (aprox. 10x lower than LabelRankT and AFOCS, 20x lower than GANXiSw). Cumulative execution times presented in Tables 4, 5, 6, Figs. 12, 13 and 14 allow to conclude that the original Louvain algorithm performs almost equally independently of the size of the increments. When using increments of 1 edge ($k = 1$) results indicate that it is possible to have better run times with the Dynamic Louvain than with the other algorithms. It is also notorious the effect of increasing the increment size in the overall duration of the community detection. Increments of $k > 1$ edges per step present worse performance than the baseline Louvain algorithm, but still indicate better execution time than LabelRank, LabelRankT, GANXiSw and AFOCS.

5.4 Results for quality of the communities

Figure 12 indicates that the quality of communities obtained by the Dynamic Louvain in the incremental network scenario had a slightly higher variance in modularity compared to the original Louvain. While the dynamic algorithm uses local modularity optimization in the affected communities, the original makes a global optimization by running the algorithm in each increment from scratch. Table 4 exhibits, for $k = \{1, 2, 5\}$, that the best modularity values were obtained using Dynamic Louvain. No significant penalty was noticed when moving from the global modularity optimization of the original Louvain to the local version of the Dynamic Louvain. In the Timestamped incremental network and Dynamic network results, apart from the modularity ($\mathcal{Q}$) are presented values for modularity with split penalty ($\mathcal{Q}_s$) and modularity density ($\mathcal{Q}_{ds}$). Using the high-energy physics theory citation network, the Dynamic Louvain algorithm results presented in Fig. 13 and Table 5 show similar values for modularity and modularity density when compared with LabelRankT and GANXiSw while superior values where obtained when compared with AFOCS. Results for modularity and modularity density for the Autonomous Systems AS-733 dataset, Fig. 14 and Table 6, show always better values when compared with all the others algorithms. In both incremental network setup (Fig. 12, Table 4) and dynamic network setup (Fig. 14, Table 6), the Dynamic Louvain results shown better values for the total number of communities and lower unassigned nodes when comparing with all the other four algorithms. This fact in conjunction with values obtained for modularity density leads us to conclude that Dynamic Louvain have a good compromise between number of communities their quality while keeping the number of unassigned nodes to communities very low.

5.5 Results for stability of the communities

In this section are presented the results for the stability of communities for both incremental and dynamic network setups. The stability measure used in the present work was based on the similarity between different communities of two consecutive snapshots. Widely adopted Jaccard coefficient for binary sets were calculated between pairs of communities of consecutive snapshots. This measure was used to track the evolution of communities in dynamic social networks [7]. The Jaccard coefficient between two snapshot communities equal to 1 mean that both communities kept the same nodes and can be marked as being the same dynamic community. In our stability measure, instead of identifying and track the dynamic communities, we only want to define a measure that describes how detected communities share the same node ids between consecutive snapshots (i.e. stability). Therefore, for each one of the detected communities in a snapshot (S_i), we searched for the community with maximum Jaccard coefficient in the following snapshot (S_{i+1}). A Jaccard coefficient close to 1 means that the two communities are very similar. The sum of all maximum Jaccard coefficients (one per detected community in the snapshot) gives the measure of stability for the current snapshot. Higher values for this measure mean that communities between two consecutive snapshots share the same node ids, on the other hand, lower values mean that most ids of nodes belonging to the detected

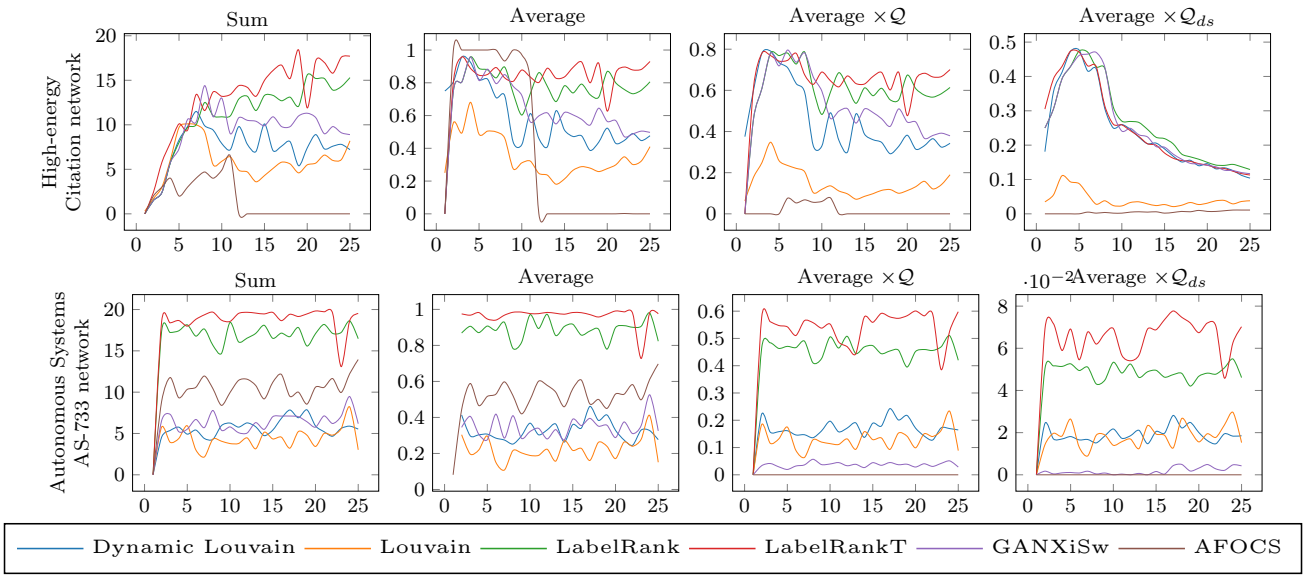

Fig. 15 Results for stability of communities for the high-energy physics theory citation network dataset (*top*) and Autonomous Systems AS-733 dataset (*bottom*). The *curves* present the sum and average of the Jaccard similarity coefficients when calculating the similarity from communities of successive snapshots. Comparison of Average of the Jaccard similarity coefficients against modularity ($\mathcal{Q}$) and modularity density ($\mathcal{Q}_{ds}$) are also included. *X*-axis show values for each one of the 25 increments

communities changed between snapshots. Based on this, other three measures were introduced: the Average Jaccard coefficient, that is the ratio between the sum of maximum Jaccard coefficients and the number of identified communities in the next snapshot (communities with Jaccard coefficient higher than 0); the Average $\times\mathcal{Q}$ and the Average $\times\mathcal{Q}_{ds}$, that plots the Average Jaccard coefficients against respective modularity ($\mathcal{Q}$) and modularity density ($\mathcal{Q}_{ds}$) values for each pair of snapshots. Calculating the Jaccard coefficients for all communities of two snapshots requires $|C_{S_i}| \times |C_{S_{i+1}}|$ calculations, where $|C_{S_i}|$ and $|C_{S_{i+1}}|$ are the total number of detected communities for snapshot S_i and S_{i+1} respectively. To simplify calculation of the stability measure only the top-20 communities were considered (i.e. the 20 biggest communities). The number of snapshots pairs under analysis was also reduced to the first 25.

In Fig. 15 are presented the results of stability for the high-energy physics theory citation network dataset [9] and for the Autonomous Systems AS-733 dataset [9]. LabelRank and LabelRankT present the best values regarding stability. This is somehow expected once that both algorithms are deterministic. Nevertheless for the high-energy physics theory citation network dataset Dynamic Louvain presents equivalent stability behavior. When compared to GANXiSw, the Dynamic Louvain algorithm shows equivalent Sum and Average stability performance and superior performance regarding stability versus modularity and stability versus modularity density (Average $\times\mathcal{Q}$ and Average $\times\mathcal{Q}_{ds}$ respectively). Finally it can be observed that Dynamic Louvain outperforms static Louvain in all stability measures. This means that communities detected by Dynamic Louvain are more stable between snapshots than the ones calculated with static Louvain. This might be justified by the fact that community ids change between snapshots and also due the fact that Louvain is non-deterministic (i.e. provides distinct solutions for different runs).

6 Conclusions

This paper presents a modularity-based dynamic community detection algorithm. The algorithm is a modification of the original Louvain method where dynamically added and removed nodes and edges only affect their related communities. In each iteration, the algorithm maintains unchanged all the communities that were not affected by modifications to the network. By reusing community structure obtained by previous iterations, the local modularity optimization step operates in smaller networks where only affected communities are disbanded to their origin. The stability of communities is also an improvement over the original algorithm. Given that only parts of the network change during iterations, the non-determinism of the algorithm will have reduced effect on the community assignment. Most node community assignments remain unchanged between snapshots, providing better community stability than its static counterpart. In the evaluation performed, the algorithm had significant performance improvements in terms of execution times and size of the

network in direct comparison with the original Louvain executed in network snapshots. The size of the upper level network was always maintained at the minimum size, reducing the computation cost of both steps 1 and 2 of the Louvain method. In terms of the community structure quality, the results indicated that there was no penalty on the calculated global modularity when considering the locally modularity optimization only for the regions where the network suffered addition or removal of nodes or edges. As expected, the results also demonstrate that the number of edges that are being added or removed should be tailored taking into account the size of the network. The probability of disbanding communities increases proportionally to the number of edges being changed at each iteration.

Acknowledgments This work was supported by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT), and by European Commission through the project MAESTRA (Grant number ICT-2013-612944).

References

1. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of community hierarchies in large networks. CoRR abs/0803.0476
2. Chen M, Nguyen T, Szymanski B (2013) On measuring the quality of a network community structure. In: Social computing (SocialCom), 2013 international conference on, pp 122–127. doi:10.1109/SocialCom. 25
3. Chen M, Nguyen T, Szymanski BK (2015) A new metric for quality of network community structure. CoRR abs/1507.04308. arXiv:1507.04308
4. Fortunato S (2010) Community detection in graphs. Phys Rep 486(3–5):75–174. doi:10.1016/j.physrep.2009.11.002
5. Fortunato S, Barthelemy M (2007) Resolution limit in community detection. In: Proceedings of the national academy of sciences. http://www.pnas.org/cgi/content/abstract/104/1/36
6. Greene D, Doyle D, Cunningham P (2010) Tracking the evolution of communities in dynamic social networks. In: ASONAM, pp 176–183. IEEE computer society, Washington, DC. doi:10.1109/ASONAM.2010.17
7. Greene D, Doyle D, Cunningham P (2010) Tracking the evolution of communities in dynamic social networks. In: Proceedings–2010 international conference on advances in social network analysis and mining, ASONAM 2010, pp 176–183. doi:10.1109/ASONAM.2010.17
8. Haynes J, Perisic I (2009) Mapping search relevance to social networks. In: Proceedings of the 3rd workshop on social network mining and analysis, SNA-KDD '09, pp 2:1–2:7. ACM, New York, NY. doi:10.1145/1731011.1731013
9. Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining–KDD '05, p. 177. ACM Press, New York. doi:10.1145/1081870.1081893
10. Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2), 026,113+. doi:10.1103/physreve.69.026113
11. Nguyen NP, Dinh TN, Tokala S, Thai MT (2011) Overlapping communities in dynamic networks: their detection and mobile applications. In: P. Ramanathan T, Nandagopal BN, Levine (eds) Proceedings of the 17th annual international conference on mobile computing and networking, MOBICOM 2011, Las Vegas, Nevada, September 19–23, pp 85–96. ACM. doi:10.1145/2030613.2030624
12. Nguyen NP, Dinh TN, Xuan Y, Thai MT (2011) Adaptive algorithms for detecting community structure in dynamic social networks. In: INFOCOM, pp 2282–2290. IEEE
13. Palla G, Barabasi AL, Vicsek T (2007) Quantifying social group evolution. Nature 446(7136):664–667. doi:10.1038/nature05670
14. Pujol JM, Erramilli V, Rodriguez P (2009) Divide and conquer: partitioning online social networks. CoRR abs/0905.4918
15. Shang J, Liu L, Xie F, Chen Z, Miao J, Fang X, Wu C (2012) A real-time detecting algorithm for tracking community structure of dynamic networks. SNAKDD, 18th ACM SIGKDD 12
16. Xie J, Chen M, Szymanski BK (2013) LabelRankT: incremental community detection in dynamic networks via label propagation
17. Xie J, Kelley S, Szymanski B (2013) Overlapping community detection in networks: the state-of-the-art and comparative study. ACM Comput Surv (CSUR). doi:10.1145/2501654.2501657
18. Xie J, Szymanski B (2013) Labelrank: a stabilized label propagation algorithm for community detection in networks. Network Science Workshop (NSW)
19. Xie J, Szymanski BK (2012) Towards linear time overlapping community detection in social networks. In: Lecture notes in computer science, vol 7301 LNAI, pp 25–36
20. Xie J, Szymanski BK, Liu X (2011) SLPA: uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: IEEE international conference on data mining, ICDM, pp 344–349
21. Ye Z, Hu S, Yu J (2008) Adaptive clustering algorithm for community detection in complex networks. Phys. Rev E 78, 046,115. doi:10.1103/PhysRevE.78.046115