DynaMo: Dynamic Community Detection by Incrementally Maximizing Modularity

Di Zhuang, *Student Member, IEEE*, J. Morris Chang, *Senior Member, IEEE*, and Mingchen Li

Abstract—Community detection is of great importance for online social network analysis. The volume, variety and velocity of data generated by today's online social networks are advancing the way researchers analyze those networks. For instance, real-world networks, such as Facebook, LinkedIn and Twitter, are inherently growing rapidly and expanding aggressively over time. However, most of the studies so far have been focusing on detecting communities on the static networks. It is computationally expensive to directly employ a well-studied static algorithm repeatedly on the network snapshots of the dynamic networks. We propose DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect communities of dynamic networks as effective as repeatedly applying static algorithms but in a more efficient way. DynaMo is an adaptive and incremental algorithm, which is designed for incrementally maximizing the modularity gain while updating the community structure of dynamic networks. In the experimental evaluation, a comprehensive comparison has been made among DynaMo, Louvain (static) and 5 other dynamic algorithms. Extensive experiments have been conducted on 6 real-world networks and 10,000 synthetic networks. Our results show that DynaMo outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm.

Index Terms—Community detection, dynamic network analysis, modularity, incremental approach

1 Introduction

With the advance of online social network analysis, more and more real-world systems, such as social networks [1], collaboration relationships [2], recommendation systems [3] and intrusion detection system [4], [5], are represented and analyzed as networks, where the vertices represent certain objects and the edges represent the relationships or connections between the objects. Most social networks have been shown to present certain community structures [6], where vertices are densely connected within communities and sparsely connected between communities. Community detection is one of the most important and fundamental problem in the field of graph mining, network science and social network analysis.

Detecting community structure is of great challenge, and most of the recent studies are proposed to detect communities in the static networks, such as spectral clustering [7], label propagation [8], modularity optimization [9], and k-clique communities [10]. However, real-world networks, especially most of the online social networks, are not static. Most popular online social networks (e.g., Facebook, LinkedIn and Twitter) are de facto growing rapidly and expanding aggressively in terms of either the size or the complexity over time. For instance, in Facebook network, the updating of its community structure could be simply caused by new users joining in, old users leaving, or certain users connecting (i.e., friend) or disconnecting (i.e., unfriend) with the other users. Facebook announced that it had 1.52 billion daily active users in the fourth quarter of 2018 [11], which shows a 9 percent increase over the same period of the previous year, and 4 million likes generated every minute as of January 2019 [12]. Hence, it is rather important and impending to enable community detection in such dynamic networks.

Designing an effective and efficient algorithm to detect communities in dynamic networks is highly difficult. First, an efficient algorithm should update the communities adaptively and incrementally depending on the changes of the dynamic networks, and avoid redundant and repetitive computations. Second, it is hard to design a dynamic algorithm that performs as effective as the static algorithms by only observing the historical community structures and the incremental changes of the dynamic networks. Third, it is still quite open about how to categorize the incremental changes of dynamic networks, and how to assess the influence of different types of the incremental changes on the community structure updates, which is rather important to design an effective and efficient dynamic algorithm.

A few algorithms have been proposed to detect communities in dynamic networks [13], [14], [15], [16], [17], [18], [19], [20], [21]. An intuitive way to detect communities in dynamic networks is to slice the network into small snapshots based on the timestamps, and directly employ well-studied static algorithms repeatedly on each network snapshot. However, these algorithms [13], [14] usually are computational expensive, since they compute the current community structures completely independent from the historical information (i.e., the previous community structures), especially when the dynamic network changes rapidly and the time interval between two consecutive network snapshots are extremely

• *The authors are with the Department of Electrical Engineering, University of South Florida, Tampa, FL 33620 USA. E-mail: {zhuangdi1990, morrisjchang, mingchenli1992}@gmail.com.*

Manuscript received 20 May 2019; revised 13 Oct. 2019; accepted 29 Oct. 2019. Date of publication 4 Nov. 2019; date of current version 1 Apr. 2021. (Corresponding author: Di Zhuang.)
Recommended for acceptance by S. Cohen.
Digital Object Identifier no. 10.1109/TKDE.2019.2951419

small. Another way to update the communities is using not only the current network changes but also the previous community structures. These algorithms [15], [16], [17], [18], [19], [20], [21] adaptively and incrementally detect communities in dynamic networks, without re-executing any static algorithms on each entire network snapshot. Those algorithms are usually more efficient than repeatedly applying static algorithms on network snapshots. However, most of those algorithms are still not practical enough to be directly used to analyze the real-world networks. For instance, some algorithms [15], [18] only considers vertices/edges additions, while vertices/edges deletions happen quite often in online social networks (e.g., "unfriend" in Facebook). Some algorithms [15], [16], [18], [20] only consider unweighted networks, which are not applicable for weighted networks. Furthermore, some algorithms [21], [22], [23] need certain prior information about the community structures (e.g., the number of communities, the ratio of vertices in overlapped communities) or need certain predefined parameters which are not available in practice.

We present DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect non-overlapped communities of dynamic networks. DynaMo is an adaptive and incremental algorithm designed for maximizing the modularity gain while updating the community structure of dynamic networks. To update the community structures efficiently, we model the dynamic network as a sequence of incremental network changes. We propose 6 types of incremental network changes: (a) intra-community edge addition/weight increase, (b) cross-community edge addition/weight increase, (c) intra-community edge deletion/weight decrease, (d) cross-community edge deletion/weight decrease, (e) vertex addition, and (f) vertex deletion. For each incremental network change, we design an operation to maximize the modularity.

In the experimental evaluation, a comprehensive comparison has been made among DynaMo, Louvain (static) [24] and 5 dynamic algorithms (i.e., QCA [15], Batch [20], GreMod [16], LBTR-LR [18] and LBTR-SVM [18]). Extensive experiments have been conducted on 6 large-scale real-world networks and 10,000 synthetic networks. Our results show that DynaMo consistently outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm. To summarize, our work has the following contributions:

- We present a novel, effective and efficient modularity-based dynamic community detection algorithm, DynaMo, capable of detecting non-overlapped communities in real-world dynamic networks.
- We present the theoretical analysis to show why/how DynaMo could maximize the modularity, while avoiding certain redundant and repetitive computations.
- A comprehensive comparison among our algorithm and the state-of-the-art algorithms has been conducted (Section 5). For the sake of reproducibility and convenience of future studies about dynamic community detection, we have released our prototype implementation of DynaMo, the experiment datasets and a collection of the implementations of the other state-of-the-art algorithms.[1]

1. https://github.com/nogrady/dynamo

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 presents the notations, the concept of dynamic networks and the definition of modularity, and introduces a baseline static community detection algorithm (i.e., Louvain algorithm). Section 4 describes our algorithm design and theoretical analysis. Section 5 presents the experimental evaluation. Section 6 concludes.

2 Related Work

To date, a few dynamic community detection approaches were proposed [13], [14], [15], [16], [18], [20], [21], [22], [23], [25], [26], [27], [28], [29], [30]. Rossetti et al. [31], a comprehensive survey on dynamic community detection, divide most of the algorithms into three categories (i.e., instant-optimal, temporal trade-off and cross-time) in terms of their ability to solve the community instability and temporal smoothing issue. However, some approaches in the literature are not belonging to any of these categories. For instance, some approaches, such as our proposed algorithm, do not consider the community instability and temporal smoothing issue, but still aim to detect communities in dynamic networks effectively and efficiently. The concept of "cross-time" approaches is also beyond the scope of this paper. After incorporating the taxonomy of [31], we consider three categories of related approaches: instant-optimal, temporal trade-off and incremental approaches (to replace the "cross-time" in [31]).

The instant-optimal approaches [13], [14], [25] have two steps: (i) static algorithms are applied on each network snapshot independently to detect static communities, (ii) communities detected on each network snapshot are matched with communities detected on the previous one. Greene et al. [25] proposed a general model for tracking communities in dynamic networks via solving a classic cluster matching problem on the communities independently detected on consecutive network snapshots. Such approaches take advantage of existing static algorithms. However, repeatedly applying static algorithms on all network snapshots of the dynamic networks is computationally expensive.

The temporal trade-off approaches [21], [22], [23], [26], [27], [28], [29] incorporate the community detection and tracking via considering the community structures of the current and historical network snapshots at the same time. Those approaches aim to maintain the evolution of the community structures of the dynamic networks, where the community structure (e.g., the number of communities, the size of communities) of the current network snapshot should be similar to that of the previous one. Tang et al. [26] propose a temporally regularized clustering algorithm to identify evolving groups in dynamic networks, where they use a metric that attempts to optimize two objectives: the quality of the current community structure and the similarity between the current and the previous community structures. However, most of those approaches, such as [22], [23], [29], require determining the number of communities to be detected/tracked in advance, which is rather impractical for the real-world dynamic networks where the number of communities changes over time.

The incremental approaches [15], [16], [18], [20], [30] adaptively update the community structures fully based on the network changes happened during the current snapshot and the community structure of the previous snapshot. For

instance, GreMod [16] is a rule-based incremental algorithm that performs the predetermined operations on different types of the edge addition changes of the dynamic network. QCA [15] is another rule-based adaptive algorithm that updates the community structures according to the predefined rules of different types of the incremental changes (i.e., vertices/edges addition/deletion) on the dynamic network. QCA is also one of the most efficient dynamic community detection algorithms in the literature. However, since the rule-based algorithms, such as GreMod [16] and QCA [15], considers each network change as an independent event, they are less efficient when abundant (i.e., a batch of) network changes appear in the same network snapshot. Chong et al. [20] propose a batch-based incremental modularity optimization algorithm that updates the community structures by initializing all of the new and changed vertices of the current network snapshot (i.e., the batch) as singleton communities and using Louvain algorithm to further update the community structures. However, since their initialization approach, that generates the intermediate community structure of a batch of network changes, is rather coarse, it is less efficient to apply Louvain algorithm on those intermediate community structures. LBTR [18] is a learning-based framework that uses machine learning classifiers and historical community structure information to predict certain vertices' new community assignments after each round of network changes. In those learning-based algorithms, once the models are being trained, the testing phase could be very efficient. However, since the supervised nature of the learning-based algorithms, it would be extremely hard to generalize the trained models. For instance, the models trained on one type of dynamic networks (e.g., social network) might be less effective to another type of dynamic networks (e.g., collaboration network). Furthermore, even for the same dynamic network, the network patterns change over time. Thus, the models have to be updated periodically, which would be rather illogical, since the network usually changes rapidly and updating models is also time consuming.

Our proposed approach, DynaMo, is an adaptive and incremental algorithm. Compared with rule-based algorithms [15], [16], our approach is capable of processing a set of network changes as a batch, and redesigns the "rules" by considering more extreme cases (Section 4.3). Compared with batch-based algorithms [20], our approach has a more fine-grained initialization phase (Section 4.3), which could reduce the computation time dramatically. Compared with learning-based algorithms [18], our approach is more generalized to real-world networks. In Section 5, we compare DynaMo with Louvain algorithm and 5 other dynamic algorithms on 6 real-world networks and 10,000 synthetic networks, showing that DynaMo consistently outperforms all the other 5 dynamic algorithms in terms of effectiveness, and much more efficient than Louvain algorithm.

3 Preliminaries

In this section, we introduce 1) the notations; 2) the dynamic network model; 3) modularity, to quantify the quality of a community structure; and 4) Louvain algorithm, a modularity-based static community detection approach.

3.1 Notations

Let $G = (V, E)$ be an undirected weighted network, where V is a set of vertices ($n = |V|$), E is a set of undirected weighted edges ($m = |E|$), and there could be more than one edge between a pair of vertices. Let C denote a set of disjoint communities associated with G, A_{ij} denote the sum of the weights of all the edges between vertices i and j, k_i denote the sum of the weights of all the edges linked to vertex i, and c_i denote the assigned community of vertex i.

3.2 Dynamic Network

Let $G^{(t)}$ denote the snapshot of a network at time t, and $\triangle G^{(t)} = (\triangle V^{(t)}, \triangle E^{(t)})$ denote the incremental change from $G^{(t)}$ to $G^{(t+1)}$ (i.e., $G^{(t+1)} = G^{(t)} \cup \triangle G^{(t)}$), where $\triangle V^{(t)}$ and $\triangle E^{(t)}$ are the sets of vertices and edges being changed during time period $(t, t+1]$. A dynamic network G is a sequence of its network snapshots changing over time: $G = \{G^{(0)}, G^{(1)}, \ldots, G^{(t)}\}$.

3.3 Modularity

Modularity [32] is a widely used criteria to evaluate the quality of given network community structure. Community structures with high modularity have denser connections among vertices in the same communities but sparser connections among vertices from different communities. Given network $G = (V, E)$, its modularity is defined as follows:

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta_{ij} = \frac{1}{2m} \sum_{c \in C}^{c} \left(\alpha_c - \frac{\beta_c^2}{2m} \right), \tag{1}$$

where $\alpha_c = \sum_{i,j \in c} A_{ij}$, $\beta_c = \sum_{i \in c} k_i$ and δ_{ij} equals to 1, if i, j belong to the same community, otherwise equals to 0.

3.4 Louvain Method for Community Detection

Since the modularity optimization problem is known to be NP-hard, various heuristic approaches are proposed [9], [33], [34]. Most of the algorithms have been superseded by Louvain algorithm [24], which attempts to maximize the modularity using a greedy optimization approach composed of three steps: (i) *Initialization*, where each vertex forms a singleton community. (ii) *Local Modularity Optimization*, where each vertex moves from its own community to its neighbor's community to maximize the local modularity gain. If there is no positive modularity gain, keep the vertex in its original community. Repeat this step over all vertices multiple times until the modularity gain is negligible. (iii) *Network Compression*, where vertices belonging to the same community are aggregated as super vertices and a new network is built with the super vertices.

Louvain algorithm repeats the last two steps, until the modularity improvements is negligible. Although the actual computational complexity of Louvain depends on the input network, it has an average-case time complexity of $O(m)$ with most of the computational effort spending on the optimization of the first level network (i.e., before creating the super vertices).

4 DynaMo: Dynamic Community Detection by Incrementally Maximizing Modularity

4.1 Problem Statement

Given a dynamic network $G = \{G^{(0)}, G^{(1)}, \ldots, G^{(t)}\}$, where $G^{(0)}$ is the initial network snapshot, let $C = \{C^{(0)}, C^{(1)}, \ldots,$

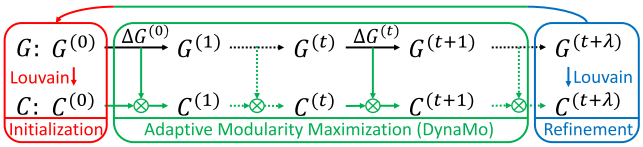


Fig. 1. The overview of DynaMo.

$C^{(t)}\}$ denote the list of community structures of the corresponding network snapshots. As illustrated in Fig. 1, we aim to design an adaptive and incremental algorithm to detect $C^{(t+1)}$, given $G^{(t)}$, $C^{(t)}$ and $\triangle G^{(t)}$.

4.2 Methodology Overview

As shown in Fig. 1, our approach has three components:

- *Initialization*: Use well-studied static algorithms (i.e., Louvain [24]) to compute $C^{(0)}$, which generates a comparatively accurate community structure of $G^{(0)}$.
- *Adaptive Modularity Maximization (DynaMo)*: Given $G^{(t)}$, $C^{(t)}$ and $\triangle G^{(t)}$, update the community structure of $G^{(t+1)}$ from $C^{(t)}$ to $C^{(t+1)}$ while maximizing the modularity gain, using predesigned strategies that fully depend on $\triangle G^{(t)}$ and $C^{(t)}$. This is the core component of our framework that relies on fine-grained and theoretical-verified strategies (Section 4.3) to maximize the modularity gain while maintaining the efficiency.
- *Refinement*: Once the obtained modularity of $C^{(t+\lambda)}$ is less than a predefined threshold, use $G^{(t+\lambda)}$ as the new initial network snapshot to restart our algorithm from the initialization step. This component prevents our frame from being trapped in the suboptimal solutions.

4.3 The DynaMo Algorithm

DynaMo is an adaptive and incremental algorithm aiming to maximize the community structure modularity gain based on the incremental changes of a dynamic network. We propose a two-step approach: (i) initialize an intermediate community structure, depending on the incremental network changes and the previous network community structure, and (ii) repeat the last two steps of Louvain algorithm (Section 3.4) on the intermediate community structure until the modularity gain is negligible.

Our algorithm benefits community detection in dynamic networks in three folds. First, in the initialization step, we categorize the incremental changes into 6 types. For each type of the incremental change, we design a strategy to initialize its corresponding intermediate community structure. Most of the strategies are theoretically verified to incrementally maximize the modularity, while avoiding redundant and repetitive computations. Second, compared with the original initialization step of Louvain algorithm, our initialization step takes advantage of the historical information, thus reduces most of the unnecessary computations happened at Louvain's first level network optimization, where Louvain spends most of its computational effort (Section 3.4). Hence, DynaMo would be much more efficient than Louvain algorithm while detecting communities in dynamic networks. Third, in the initialization, our algorithm could process a set of incremental changes as a batch, which makes the computational complexity of our algorithm less sensitive to the amount of incremental changes and the frequency of network changes. So, DynaMo can detect communities while the network changing rapidly.

In this section, 6 different types of the incremental changes have been defined, where the initialization strategy of each type is also designed accordingly. Eight propositions are proposed and proved to provide the theoretical guarantees of our strategies towards maximizing the modularity.

4.3.1 Edge Addition/Weight Increase (EA/WI)

In this scenario, an edge (i, j, w_{ij}) between two existing vertices i and j has been changed to $(i, j, w_{ij} + \triangle w)$, where $w_{ij} \geq 0$ and $\triangle w > 0$. Edge addition is a special case of edge weight increase, where $w_{ij} = 0$. Depending on the edge property, we define two sub-scenarios:

Intra-Community EA/WI (ICEA/WI). Vertices i and j belong to the same community (i.e., $c_i = c_j$). According to Proposition 1, ICEA/WI will never split i and j into different communities. And according to Remark 1, sometimes ICEA/WI will split c_i into multiple communities, while keeping i and j in the same community. Proposition 2 also provides us a convenient tool to decide when c_i should be bi-split into two smaller communities (i.e., c_p and c_q). However, this approach requires checking all the bi-split combinations of c_i, which is time consuming, especially when c_i is huge. In this case, we propose to initialize i and j as a two-vertices community, and all the other vertices in c_i as singleton communities.

Proposition 1. *Adding an edge or increasing the edge weight between vertices i and j, that belong to the same community ($c_i = c_j$), will not split i and j into different communities.*

See Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2019.2951419, for the proof.

Remark 1. Although our Proposition 1 shows that ICEA/WI between i and j, where $c_i = c_j$, will not split them into different communities, sometimes splitting c_i into smaller communities in other ways (i.e., keeping i and j in the same community after the splitting) might maximize the modularity. For instance, as shown in Fig. 2, assume all the edge

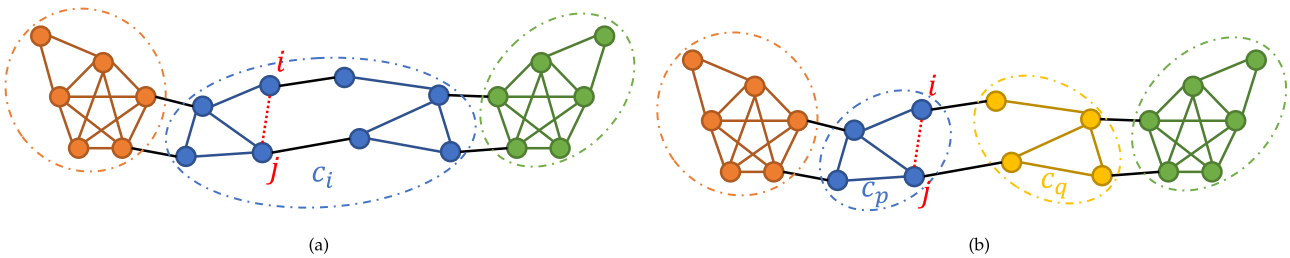


Fig. 2. Two possible behaviors of the community structure after adding an intra-community edge: (a) Unchanged and (b) splitting to smaller communities.

weights are 1.0, and the red dash line between i and j is a newly added intra-community edge. Before adding the new edge, the modularity of community structure in Fig. 2a (i.e., 0.561) is higher than that in Fig. 2b (i.e., 0.558). However, after adding the new edge, the modularity of community structure in Fig. 2a (unchanged, i.e., 0.564) becomes lower than that in Fig. 2b (split, i.e., 0.568). In this case, although an intra-community edge has been added, splitting c_i into c_p and c_q provides higher modularity. Our algorithm carefully considers these "counterintuitive" cases, which is different from QCA [15], [35], thus, leading our algorithm to be more effective (Section 5.5).

Proposition 2. *(ICEA/WI Community Bi-split) After ICEA/WI between vertices i and j, where $c_i = c_j$, if a bi-split of c_i (i.e., $c_p \subseteq c_i$ and $c_q = c_i \backslash c_p$) does not exist such that $\triangle w > \frac{m\alpha_1 - \beta_{c_p}\beta_{c_q}}{2\beta_{c_q} - \alpha_1}$, where $\alpha_1 = \alpha_{c_i} - \alpha_{c_p} - \alpha_{c_q}$, any other bi-split of c_i will not improve the modularity gain comparing with keeping the community structure unchanged.*

See Appendix A.2, available in the online supplemental material, for the proof.

Cross-Community EA/WI (CCEA/WI). Vertices i and j are from two different communities (i.e., $c_i \neq c_j$). CCEA/WI between i and j leads to three possible operations: (a) keeping the community structure unchanged; (b) merging c_i and c_j into one community; and (c) splitting $c_k = c_i \cup c_j$ into other smaller communities. According to Proposition 3, if $\triangle w$ is large enough, merging c_i and c_j into one community (e.g., c_k) provides higher modularity gain than keeping the community structure unchanged. However, if $\triangle w$ is too large (as shown in Proposition 4), CCEA/WI is equivalent to a two-step process: (a) CCEA/WI between i and j ($c_i \neq c_j$), that results in merging c_i and c_j into one community c_k (Proposition 3); (b) ICEA/WI between i and j ($c_i = c_j = c_k$). Proposition 4 provides a bi-split condition. However, Proposition 4 also requires checking all bi-split combinations of c_k. Hence, to deal with CCEA/WI, we propose: (a) if $\triangle w \leq \frac{1}{2}\big(\alpha_2 + \beta_2 - 2m + \sqrt{(2m - \alpha_2 - \beta_2)^2 + 4(m\alpha_2 + \beta_{c_i}\beta_{c_j})}\big)$, where $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$ and $\beta_2 = \beta_{c_i} + \beta_{c_j}$, we keep the community structure unchanged; (b) otherwise, we employ the same initialization approach proposed to deal with ICEA/WI on $c_k = c_i \cup c_j$, where we consider ICEA/WI has happened between vertices i and j, where $c_i = c_j = c_k$.

Proposition 3. *(CCEA/WI Community Merge) After CCEA/WI between i and j, where $c_i \neq c_j$, if and only if $\triangle w > \frac{1}{2}\big(\alpha_2 + \beta_2 - 2m + \sqrt{(2m - \alpha_2 - \beta_2)^2 + 4(m\alpha_2 + \beta_{c_i}\beta_{c_j})}\big)$, where $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$ and $\beta_2 = \beta_{c_i} + \beta_{c_j}$, merging c_i and c_j into c_k (i.e., $c_k = c_i \cup c_j$) has higher modularity gain than keeping the community structure unchanged.*

See Appendix A.3, available in the online supplemental material, for the proof.

Proposition 4. *(CCEA/WI Community Bi-split) After CCEA/WI between i and j, where $c_i \neq c_j$, $c_k = c_i \cup c_j$, and $\{c_p, c_q\}$ is another bi-split of c_k (i.e., $c_p \subseteq c_k$ and $c_q = c_k \backslash c_p$), if and only if $\triangle w > \frac{1}{2}\big(\alpha_2 + \beta_2 - 2m + \sqrt{(2m - \alpha_2 - \beta_2)^2 + 4(m\alpha_2 + \beta_{c_i}\beta_{c_j})}\big) + \frac{m\alpha_1 - \beta_{c_p}\beta_{c_q}}{2\beta_{c_q} - \alpha_1}$, where $\alpha_1 = \alpha_{c_i} - \alpha_{c_p} - \alpha_{c_q}$, $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$ and $\beta_2 = \beta_{c_i} + \beta_{c_j}$, splitting c_k into c_p and c_q has higher modularity gain than either keeping the community structure unchanged or merging c_i and c_j into c_k.*

The proof could be easily derived from Propositions 2 and 3.

4.3.2 Edge Deletion/Weight Decrease (ED/WD)

In this scenario, an edge (i, j, w_{ij}) between two existing vertices i and j has been changed to $(i, j, w_{ij} - \triangle w)$, where $w_{ij} \geq \triangle w > 0$. Edge deletion is a special case of edge weight decrease, where $w_{ij} = \triangle w$. Depending on the edge property, we define two sub-scenarios:

Intra-Community ED/WD (ICED/WD). Vertices i and j belong to the same community (i.e., $c_i = c_j$). According to Proposition 5, if i or j has one degree, decreasing the edge weight between i and j will keep the community structure unchanged. Also, intuitively, if i or j has one degree, deleting the edge between i and j will result in the same community structure plus one or two singleton communities (i.e., the vertex of one degree becomes singleton community). Except for the case above (i.e., i or j has one degree), ICED/WD between i and j leads to three other possible operations: (a) keeping the community structure unchanged, if c_i is still densely connected; (b) splitting c_i into multiple smaller communities, if c_i becomes sparsely connected; and (c) merging c_i with some of its neighbor communities (i.e., the opposite situation of Remark 1). Since the analytical approach is complex and time consuming, we propose to initiate all vertices within the communities, that adjacent to i or j (including c_i), as singleton communities.

Proposition 5. *For any pair of vertices i, j that belong to the same community (i.e., $c_i = c_j$), if i or j has only one neighbor vertex (j or i), decreasing the edge weight between i and j, does not split i and j into different communities.*

See Appendix A.4, available in the online supplemental material, for the proof.

Cross-Community ED/WD (CCED/WD). Vertices i and j are from two different communities (i.e., $c_i \neq c_j$). By Proposition 6, CCED/WD strengthens the community structure, thus, keeping the community structure unchanged.

Proposition 6. *If vertices i and j are from different communities ($c_i \neq c_j$), deleting an edge or decreasing the edge weight between i and j, will increase the modularity gain coming from c_i and c_j.*

See Appendix A.5, available in the online supplemental material, for the proof.

4.3.3 Vertex Addition (VA)

In this scenario, a new vertex i and its associated edges are added. On one hand, if i has no associated edge, we make it as a singleton community and keep the rest community structure unchanged. On the other hand, if i has one or more associated edges, some interesting cases would happen. For instance, if all of i's associated edges are connected to the same community, i.e., c_j, by Proposition 7, we should merge i into c_j and treat all of i's associated edges as ICEA/WI. A more complicated case occurs when i's associated

edges are connected to different communities. In this case, by Proposition 8, we could merge i into community c_j that has the highest $\triangle w_{ij}$. However, other than simply determining which community i should merge into, we should also consider which set of vertices could together with i to form a new community, or which community could be split into smaller communities, to further maximize the modularity. To cope with all the cases, where i has one or more associated edges, we propose to initialize i and j as a two-vertices community, where edge e_{ij} has the highest weight among all of i's associated edges (randomly selecting a vertex j if there are ties), and initialize all the other vertices within i's adjacent communities as singleton communities.

Algorithm 1. DynaMo Initialization (Init)

Input: $V^{(t+1)}$, $E^{(t+1)}$, $V^{(t)}$, $E^{(t)}$, $C^{(t)}$.
Output: $\triangle C_1$, $\triangle C_2$.
$\triangle E \leftarrow A\ set\ of\ edges\ changed\ from\ E^{(t)}\ to\ E^{(t+1)}$;
$\triangle V_{add} \leftarrow V^{(t+1)} \backslash V^{(t)}$; $\triangle V_{del} \leftarrow V^{(t)} \backslash V^{(t+1)}$;
$\triangle C_1 \leftarrow \emptyset$; $\triangle C_2 \leftarrow \emptyset$;
for $e_{ij} \in \triangle E$ **do**
  for $k \in \{i, j\}$ **do**
    if $k \in \triangle V_{del}$ **then**
      $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_k\}$;
      for $e_{kl} \in E^{(t)}$ **do**
        $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_l\}$;
    if $k \in \triangle V_{add}$ **then**
      $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_k\}$;
      $w_{max} = 0$; $c \leftarrow \emptyset$;
      for $e_{kl} \in E^{(t+1)}$ **do**
        $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_l\}$;
        if $w_{kl} > w_{max}$ **then**
          $w_{max} = w_{kl}$; $c \leftarrow \{k, l\}$;
      $\triangle C_2 \leftarrow \triangle C_2 \cup \{c\}$;
  if $i, j \notin \triangle V_{del} \cup \triangle V_{add}$ **then**
    if $e_{ij} \notin E^{(t+1)}$ **or** $w_{ij}^t > w_{ij}^{t+1}$ **then**
      if $c_i = c_j$ **then**
        $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_i\}$;
        for $k \in \{i, j\}$ **do**
          for $e_{kl} \in E^{(t)}$ **do**
            $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_l\}$;
    if $e_{ij} \notin E^{(t)}$ **or** $w_{ij}^t < w_{ij}^{t+1}$ **then**
      if $c_i = c_j$ **then**
        $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_i\}$; $c \leftarrow \{i, j\}$;
        $\triangle C_2 \leftarrow \triangle C_2 \cup \{c\}$;
      else
        $\triangle w = w_{ij}^{t+1} - w_{ij}^t$; $c_k = c_i \cup c_j$;
        $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$; $\beta_2 = \beta_{c_i} + \beta_{c_j}$;
        $\delta_1 = 2m - \alpha_2 - \beta_2$; $\delta_2 = m\alpha_2 + \beta_{c_i}\beta_{c_j}$;
        if $2\triangle w + \delta_1 > \sqrt{\delta_1^2 + 4\delta_2}$ **then**
          $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_i, c_j\}$; $c \leftarrow \{i, j\}$;
          $\triangle C_2 \leftarrow \triangle C_2 \cup \{c\}$;
return $\triangle C_1$, $\triangle C_2$.

Proposition 7. *If a new vertex i has been added and all of its associated edges are connected to the same community, i.e., c_j, merging i into c_j has higher modularity gain than keeping i as a singleton community.*

See Appendix A.6, available in the online supplemental material, for the proof.

Proposition 8. *Suppose a new vertex i has been added and its associated edges are connected to different communities. Let $\triangle w_{ij}$ denote the sum of the edge weights of vertex i's associated edges that are connected to community c_j. Given two communities c_p and c_q, if $\triangle w_{ip} > \triangle w_{iq}$, merging i into c_p has more modularity gain than merging i into c_q.*

See Appendix A.7, available in the online supplemental material, for the proof.

4.3.4 Vertex Deletion (VD)

In this scenario, an old vertex i and its associated edges are deleted. On one hand, if i has no associated edge, deleting i has no influence on the rest of the network, and hence, we should keep the community structure unchanged. On the other hand, if i has too many associated edges, deleting i might cause its community and its neighbor communities being broken into smaller communities and potentially being merged into other communities. To handle this case, we propose to initialize all the vertices within c_i and i's neighbor communities as singleton communities.

Algorithm 2. DynaMo

Input: $G^{(t+1)}$, $G^{(t)}$, $C^{(t)}$.
Output: $C^{(t+1)}$.
$\triangle C_1$, $\triangle C_2 \leftarrow$ **Init**($V^{(t+1)}$, $E^{(t+1)}$, $V^{(t)}$, $E^{(t)}$, $C^{(t)}$);
$C^{(t+1)} \leftarrow C^{(t)}$;
for $c_i \in \triangle C_1$ **do**
  $C^{(t+1)} \leftarrow C^{(t+1)} \backslash \{c_i\}$;
  for $k \in c_i$ **do**
    Create singleton community: $c_k \leftarrow \{k\}$;
    $C^{(t+1)} \leftarrow C^{(t+1)} \cup \{c_k\}$;
for $c = \{i, j\} \in \triangle C_2$ **do**
  Create two-vertices community: $c_k \leftarrow \{i, j\}$;
  $C^{(t+1)} \leftarrow (C^{(t+1)} \backslash \{c_i, c_j\}) \cup \{c_k\}$;
$C^{(t+1)} \leftarrow$ **Louvain**($C^{(t+1)}$, $G^{(t+1)}$);
return $C^{(t+1)}$.

4.4 Implementation and Analysis

4.4.1 Implementation

Algorithm 1 presents the DynaMo Initialization, where we implement the operation of each type of incremental network change to initialize the intermediate community structure towards maximizing the modularity. The input contains the current network $G^{(t+1)}$, the previous network $G^{(t)}$ and the previous community structure $C^{(t)}$. The output contains two set of communities, $\triangle C_1$ and $\triangle C_2$, that will be modified to initialize the intermediate community structure at the beginning of the second phase. $\triangle C_1$ contains a set of communities in $C^{(t)}$ to be separated into singleton communities, and $\triangle C_2$ contains a set of two-vertices communities to be created. Algorithm 2 presents the second phase, where the last two steps of Louvain algorithm is applied on the initialized intermediate community structure of $G^{(t+1)}$.

Most of the operations in Algorithm 1 are theoretically guaranteed by our propositions described in Section 4.3 to maximize the modularity, while some of the operations are heuristically designed for the sake of the efficiency. For

TABLE 1
Description of the Real-World Dynamic Networks [Notations: $|V|$ ($|E|$): # of Unique Vertices (Edges); $\mathbb{E}[|\triangle V|]$ ($\mathbb{E}[|\triangle E|]$): Avg. # of Vertices (Edges) Changed per Network Snapshots; # of Snapshots: Total Number of Consecutive Network Snapshots; Time-Interval: Period of Time Between Two Consecutive Network Snapshots; Time-Span: Total Time Spanning of Each Network Dataset]

networks	$\|\mathbf{V}\|$	$\mathbb{E}[\|\triangle\mathbf{V}\|]$	vertex-type	$\|\mathbf{E}\|$	$\mathbb{E}[\|\triangle\mathbf{E}\|]$	edge-type	# of snapshots	time-interval	time-span
Cit-HepPh	30,501	6,460	author	346,742	11,127	co-citation	31	4 months	124 months
Cit-HepTh	7,577	1,253	author	51,089	2,042	co-citation	25	5 months	125 months
DBLP	1,411,321	122,731	author	5,928,285	191,233	co-authorship	31	2 years	62 years
Facebook	59,302	12,765	user	592,406	20,943	friendship	28	1 month	28 months
Flickr	780,079	93,253	user	4,407,259	168,977	follow	24	3 days	72 days
YouTube	3,160,656	91,954	user	7,211,498	175,303	subscription	33	5 days	165 days

instance, according to Proposition 1, Remark 1 and Proposition 2, given ICEA/WI between vertices i and j, we initialize i and j as a two-vertices community to incrementally maximize the modularity, and initialize all the other vertices in c_i as singleton communities to take all the influenced vertices into consideration carefully while maintaining the algorithm efficiency (lines 26-28). According to Propositions 3 and 4, we use a designed threshold condition (lines 30-33) to determine the operation of given CCEA/WI. If the condition is true, we use the same operation of ICEA/WI to tackle CCEA/WI (lines 33-35). Otherwise, we keep the community structure unchanged to incrementally maximize the modularity. According to Proposition 5 and the analysis in Section 4.3.2, given ICED/WD, we initialize all the potentially influenced vertices as singleton communities to maintain a trade-off between the effectiveness and efficiency (lines 18-24). According to Proposition 6, given CCED/WD, we keep the community structure unchanged to maximize the local modularity gain. According to Propositions 7 and 8, given new vertex i and its associated edges, we initialize i and its most closely connected neighbor vertex as a two-vertices community (lines 12, 15-17), and initialize all the potentially influenced vertices as singleton communities (lines 10-16). After deleting vertex i, we heuristically initialize all the vertices within c_i and i's neighbor communities as singleton communities (lines 6-9). To summarize, initializing $\triangle C_2$ aims to incrementally maximize the modularity with certain theoretical guarantees, and initializing $\triangle C_1$ aims to heuristically maximize the modularity (by Algorithm 2) while maintaining the algorithm efficiency.

4.4.2 Time Complexity Analysis

The computation of our algorithm tackling one network snapshot comes from two parts: (a) the initialization, and (b) the last two steps of Louvain algorithm. In the initialization, different network changes trigger different operations, thus resulting in different computation time. For instance, if one network change is ICEA/WI (i.e., e_{ij}, $c_i = c_j$), our algorithm (line 26-28) will add c_i into $\triangle C_1$, and add $c = \{i, j\}$ into $\triangle C_2$. The time complexity of both operations are $O(1)$, thus, the time complexity to deal with single change of ICEA/WI is $O(1)$. Similarly, the time complexities to deal with single change of CCEA/WI (line 29-35) and CCED/WD (no operation needed) are also $O(1)$. To deal with single change of ICED/WD, VA or VD, our algorithm runs through the set of neighbor vertices of the changed edge, and thus, result in $O(\frac{|E|}{|V|})$ time complexity. Furthermore, as shown in Algorithm 1, each network snapshot usually has multiple network changes. Since the number of network changes is proportional to $\triangle E$, the overall time complexity of the initialization is $O(|\triangle E|)$ or $O(|\triangle E| \cdot \frac{|E|}{|V|})$.

The time complexity of the original Louvain algorithm is $O(|E|)$. However, compared with the Louvain algorithm initialization, our algorithm considers the historical information and designs an initialization phase to reduce the number of edges left for the second phase analysis as much as possible. Thus, the time complexity of the second phase of our algorithm is $O(|E|^*)$, where $|E|^* \ll |E|$. Hence, the overall best case time complexity of our algorithm is $O(|\triangle E| + |E|^*)$, and the worst case is $O(|\triangle E| \cdot \frac{|E|}{|V|} + |E|^*)$.

5 Experimental Evaluation

5.1 Experiment Environment

All the experiments were conducted on a PC with an Intel Xeon Gold 6148 Processor, 128 GB RAM, running 64-bit Ubuntu 18.04 LTS operating system. All the algorithms and experiments are implemented using Java with JDK 8.

5.2 Baseline Approaches

We compare DynaMo with *Louvain* (Section 3.4), and 5 dynamic algorithms: (i) *Batch [20]:* a batch-based incremental modularity optimization algorithm; (ii) *GreMod [16]:* a rule-based incremental algorithm that performs predetermined operations on edge additions; (iii) *QCA [15]:* a rule-based incremental algorithm that updates the community structures according to predefined rules of vertex/edge additions/deletions; (iv) *LBTR [18]:* a learning-based algorithm that uses classifiers to update community assignments. We use Support Vector Machine (SVM) and Logistic Regression (LR) as the classifiers, namely *LBTR-SVM* and *LBTR-LR*.

5.3 Experiment Datasets

We conduct our experiments on two categories of networks: real-world networks (ground-truth is unknown), and synthetic networks (ground-truth is known).

5.3.1 Real-World Dynamic Networks

As shown in Table 1, six real-world networks are used in our experiments. (i) *Cit-HepPh (Cit-HepTh) [36]* contains the citation network of high-energy physics phenomenology (theory) papers from 1993 to 2003. (ii) *DBLP [37]* contains a co-authorship network of computer science papers ranging from 1954 to 2015, where each author is represented as a

vertex and co-authors are linked by an edge. (iii) *Facebook [38]* contains the user friendship establishment information from about 52 percent of Facebook users in New Orleans area, spanning from September 26th, 2006 to January 22nd, 2009. In this network, each vertex represents a Facebook user, and each edge represents an user-to-user friendship establishment link that contains a timestamp representing the time of friendship establishment. (iv) *Flickr [39]* was obtained on January 9th, 2007, and contains over 1.8 million users and 22 million links, and each link has a timestamp that represents the time of the following link establishment. We select a sub-network, where all the user-to-user following links were established from March 6th, 2007 to May 15th, 2007. (v) *YouTube [40]* was obtained on January 15th, 2007 and consists of over 1.1 million users and 4.9 million links, and each link has a timestamp that represents the time of the subscribing link establishment. We select a sub-network, where all the user-to-user subscribing links were established from February 2nd, 2007 to July 23rd, 2007.

5.3.2 Synthetic Dynamic Networks

We use RDyn [41], a benchmark model focusing on community changes in dynamic networks, to generate synthetic networks and their ground-truth communities. It allows us to specify different parameters, such as the number of vertices (N), the number of time points (T), the maximum number of community change events (e.g., splitting or merging) per time point (M), etc.. We use various combinations of N, T and M to generate synthetic networks, where $N \in \{200, 400, 600, 800, 1000\}$, $T \in \{25, 50, 75, 100, 125\}$, $M \in \{1, 2, 3, 4\}$ and all the other parameters set by default values. For each parameter combination (out of 100 combinations in total), we randomly generate 100 synthetic networks, resulting in 10,000 synthetic networks in total.

5.4 Experimental Procedure

For each real-world network, we apply Louvain algorithm on its initial snapshot to obtain its initial community structure (Section 4.2). For each synthetic network, we use the ground-truth communities of its initial snapshot as its initial community structure. For the rest of snapshots of real-world and synthetic networks, the dynamic algorithms only use the initial community structure and the network changes between two consecutive snapshots to update the new community structures, while the static algorithm will be applied on the whole network of each snapshot. All experiments are performed for 200 times to obtain the average results.

5.5 Effectiveness Analysis

5.5.1 Effectiveness Metrics

We evaluate the effectiveness of the community detection algorithms using three metrics: modularity, Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI). Modularity (Section 3.3) is designed to measure the strength of dividing a network into communities, and does not require the ground-truth information. Hence, we use modularity to evaluate the results of the real-world networks. NMI and ARI are designed to measure the similarities between the community structure obtained from the experiments and that of the ground-truth, which are used to evaluate the results of the synthetic networks.

Let C_t denote the ground-truth community division, and C_r denote the experiment result. NMI is defined as follows:

$$NMI(C_t, C_r) = \frac{2 \times I(C_t; C_r)}{[H(C_t) + H(C_r)]}, \tag{2}$$

where $H(C_r)$ is the entropy of C_r, and $I(C_t; C_r)$ is the mutual information between C_t and C_r. NMI ranges from 0 to 1. NMI closing to 1 indicates C_r is similar to C_t, while closing to 0 means C_r is random compared with C_t.

Let a be the number of pairs of vertices in the same community in both C_t and C_r, b be the number of pairs of vertices in the same community in C_t and in different communities in C_r, c be the number of pairs of vertices in different communities in C_t and in the same community in C_r, d be the number of pairs of vertices in different communities in both C_t and C_r. ARI is defined as follows:

$$ARI(C_t, C_r) = \frac{2(ad-bc)}{b^2+c^2+2ad+(a+d)(b+c)}, \tag{3}$$

where its upper bound is 1, and the higher, the better.

5.5.2 Experimental Results

Fig. 3 shows the modularity results of 7 algorithms running on 6 real-world networks, respectively. We observe that DynaMo consistently outperforms all the other dynamic algorithms in terms of modularity. Compared with the runner-up algorithm (Batch), DynaMo obtains 2.6, 2.2, 4.3, 2.1, 1.1 and 2.2 percent higher modularity averaged over all the time points, and 3.2, 4.4, 17.3, 2.4, 1.2 and 4.7 percent higher modularity on the last time point of Cit-HepPh, Cit-HepTh, DBLP, Facebook, Flickr and YouTube, respectively. Compared with Louvain, DynaMo achieves nearly identical performance, with only 0.49, 0.38, 0.06, 0.7, 0.5 and 0.5 percent lower modularity averaged over all the time points, and only 0.52, 0.74, 0.27, 0.46, 0.5 and 1.7 percent lower modularity on the last time point of Cit-HepPh, Cit-HepTh, DBLP, Facebook, Flickr and YouTube, respectively.

Fig. 4 shows the NMI results (mean and standard deviation) of 6 dynamic algorithms running on 10,000 synthetic networks. We observe that DynaMo obtains the highest NMI value among all the dynamic algorithms regardless of any RDyn parameters. DynaMo outperforms the runner-up algorithm (QCA) by 69.1, 66.4 and 70.3 percent on average with the increase of the number of vertices, the maximum number of events per time point, and the number of time points, respectively, which is statistically significant according to the two-sample t-test with 95 percent confidence interval. The NMI standard deviation of DynaMo is also lower than that of QCA, demonstrating the consistency of DynaMo in detecting communities of various dynamic networks. Furthermore, as the maximum number of events per time point and the number of time points increase, DynaMo has the minimum NMI value loss among all the dynamic algorithms, indicating DynaMo is more robust and consistent while detecting communities of dynamic networks that last longer and have more events per time point.

Fig. 5 shows the ARI results (mean and standard deviation) of 6 dynamic algorithms running on 10,000 synthetic networks, which share similar patterns as the NMI results. DynaMo outperforms the runner-up algorithm (QCA) by 211.1, 224.5 and 257.6 percent on average with the increase of

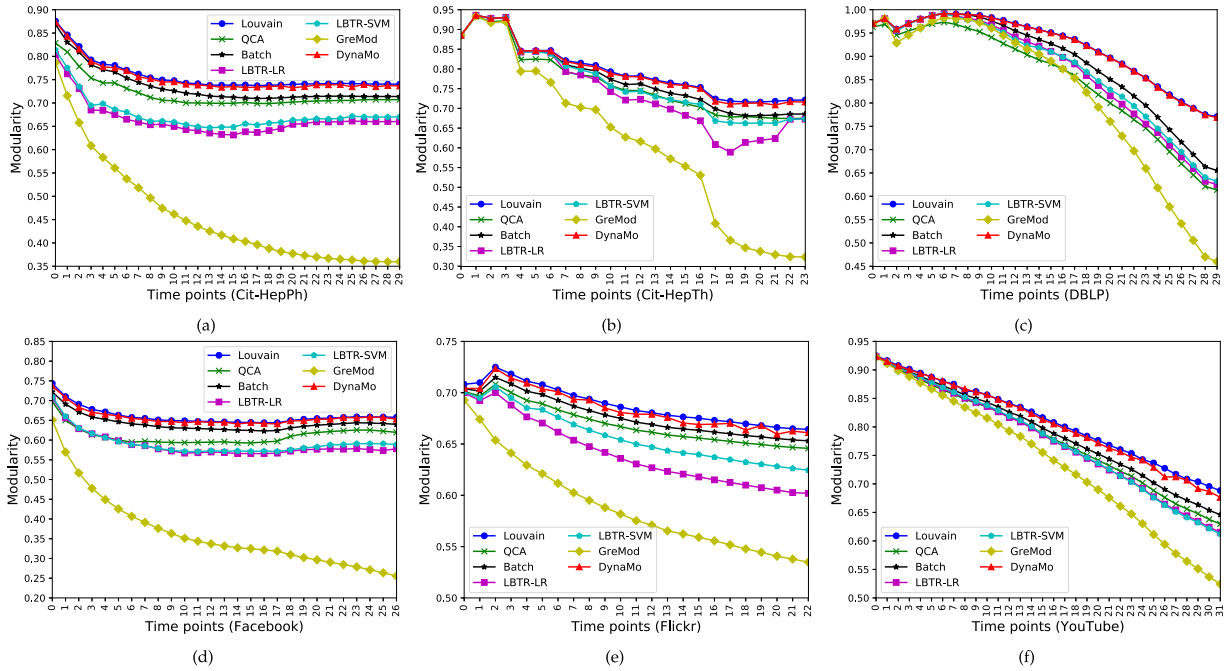


Fig. 3. The modularity results of real-world networks. (a) Cit-HepPh. (b) Cit-HepTh. (c) DBLP. (d) Facebook. (e) Flickr. (f) YouTube.

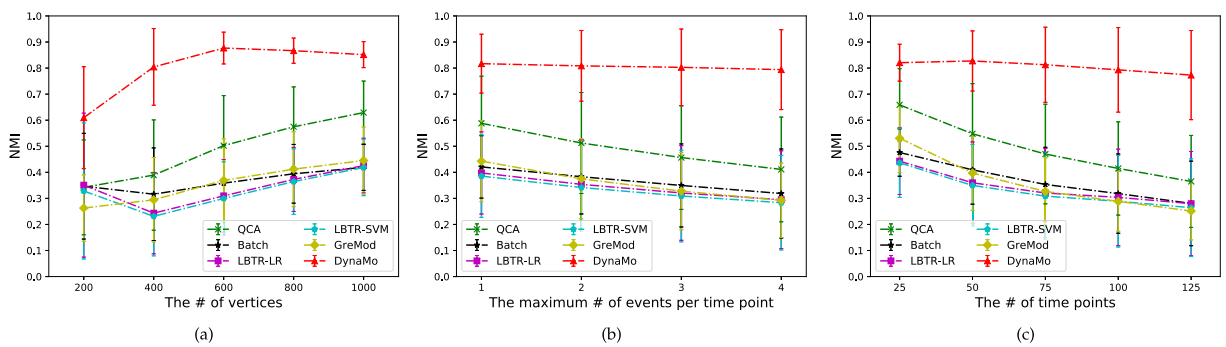


Fig. 4. The NMI results of synthetic networks. (a) The # of vertices. (b) The maximum # of events per time point. (c) The # of time points.

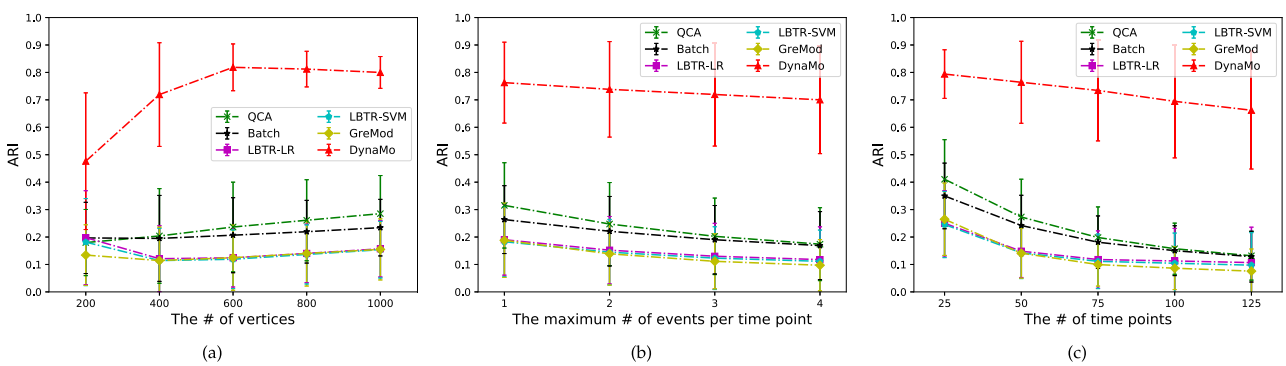


Fig. 5. The ARI results of synthetic networks. (a) The # of vertices. (b) The maximum # of events per time point. (c) The # of time points.

the number of vertices, the maximum number of events per time point, and the number of time points, respectively, which is statistically significant according to the two-sample t-test with 99 percent confidence interval. As the number of vertices increases, the ARI standard deviation of DynaMo dramatically decreases, while as the maximum number of events per time point and the number of time points increase, the standard deviation of DynaMo slightly increases. However, even considering the standard deviation difference, DynaMo still significantly outperforms all the other dynamic algorithms.

5.6 Efficiency Analysis

5.6.1 *Time Complexity Analysis*

Table 2 shows the theoretical time complexities of all the competing algorithms. DynaMo, QCA and GreMod have

TABLE 2
A Comparison of the Time Complexities of the Competing Algorithms [Notations: $n = |V|$ ($m = |E|$): # of Unique Vertices (Edges); $\upsilon = |\triangle V|$ ($\epsilon = |\triangle E|$): # of Vertices (Edges) Changed; m_b^* (m_d^*): # of Unique Vertices (Edges) after the Initialization Phase of Batch (DynaMo), and $m_b^* \ll m$ ($m_d^* \ll m$); T_{LR} (T_{SVM}): The Time Complexity of Using Logistic Regression (Support Vector Machine) in LBTR]

algorithms	the best case	the worst case
Louvain [24]	$O(m)$	$O(m)$
Batch [20]	$O((\upsilon+\epsilon)\cdot\frac{m}{n}+m_b^*)$	$O((\upsilon+\epsilon)\cdot\frac{m}{n}+m_b^*)$
DynaMo	$O(\epsilon+m_d^*)$	$O(\epsilon\cdot\frac{m}{n}+m_d^*)$
QCA [15]	$O(\epsilon)$	$O(\epsilon\cdot m)$
GreMod [16]	$O(\epsilon)$	$O(\epsilon\cdot n)$
LBTR-LR [18]	$O(\upsilon\cdot T_{LR})$	$O(\upsilon\cdot T_{LR})$
LBTR-SVM [18]	$O(\upsilon\cdot T_{SVM})$	$O(\upsilon\cdot T_{SVM})$

different time complexities while running in different scenarios (i.e., best/worst case). As discussed in Section 4.4.2, DynaMo has the best case time complexity when the network changes are ICEA/WI, CCEA/WI or CCED/WD, and otherwise, has the worst case time complexity. Similarly, QCA and GreMod have the best case time complexity if the network changes are ICEA or CCED, and otherwise, have the worst case time complexity. For the other algorithms, the time complexities of the best and the worst cases are identical. Below show the details about our analysis.

- Compared with Louvain [24], DynaMo has less time complexity, when the impact of the network changes of a given network snapshot on its community structure updating is small enough to ensure $m_d^* \ll m$. First, the evolutionary nature of the real-world dynamic networks assumes two consecutive network snapshots of the same network should have similar community structures. Therefore, each snapshot of a dynamic network should only result in a small part of its community structure being updated (i.e., $m_d^* \ll m$). Also, from our empirical studies, the assumption of $m_d^* \ll m$ always holds. Hence, DynaMo should be more efficient than Louvain for most of the time.
- Compared with Batch [20], DynaMo has less initialization time complexity (i.e., $O(\epsilon\cdot\frac{m}{n}) < O((\upsilon+\epsilon)\cdot\frac{m}{n})$), and different second phase time complexities (i.e., m_d^* versus m_b^*).
- Compared with QCA [15] and GreMod [16], who update the community structure according to certain predefined rule of each network change and one at a time (i.e., not in a batch fashion), DynaMo is more efficient if each network snapshot has more network changes, since DynaMo is capable of handling a batch of network changes.
- Compared with LBTR [18], who uses machine learning models to decide if a vertex needs to revise its community, DynaMo is more consistent and practical when dealing with different real-world networks. Since the characteristics of an dynamic network keep changing over time, LBTR has to keep updating the machine learning models to adapt the new characteristics. In such case, we have to take the training time into account. Also, the time complexity of LBTR highly depends on the machine learning algorithm used for the classification problem (e.g., $O(T_{SVM}) > O(T_{LR})$).

5.6.2 *Empirical Result Studies*

Since the theoretical time complexities always depend on the ideal scenarios or extreme cases, it is necessary to conduct empirical studies using real-world networks. To ensure the comparison is as unbiased as possible, all the algorithms are implemented using Java and running on the same environment. Fig. 6 shows the cumulative elapsed time results, and below show the details about our observations.

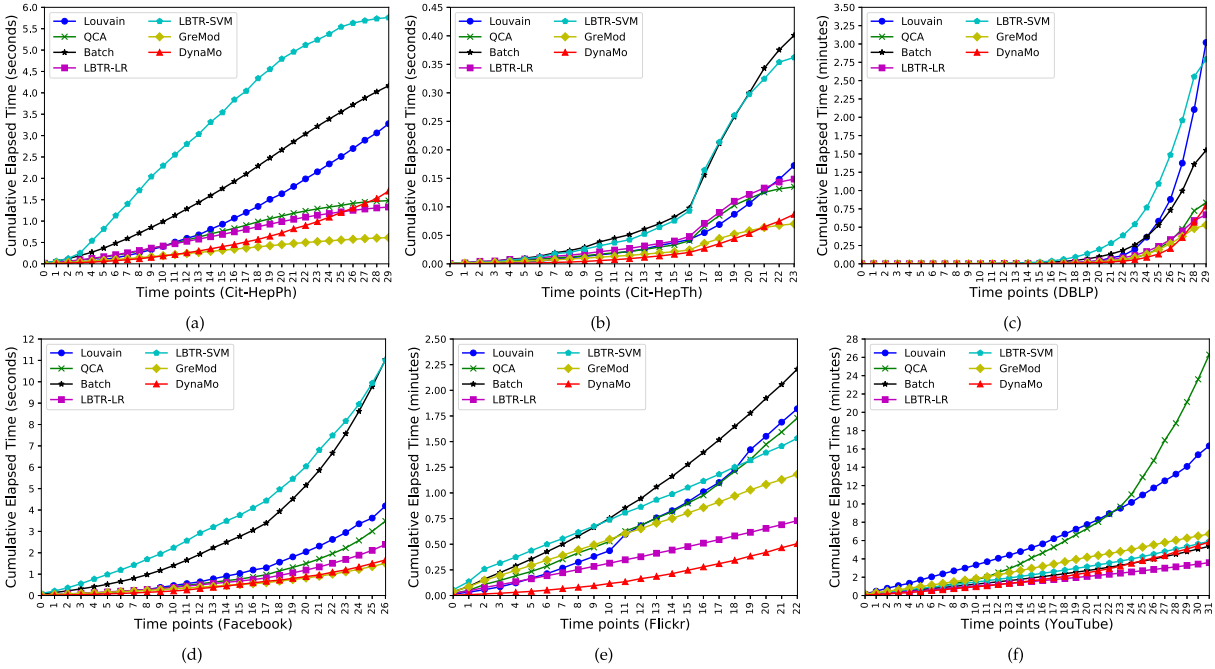


Fig. 6. The cumulative elapsed time results of real world networks. (a) Cit-HepPh. (b) Cit-HepTh. (c) DBLP. (d) Facebook. (e) Flickr. (f) YouTube.

- Compared with Louvain [24], DynaMo obtains over 2x, 2x, 4x, 3x, 4x and 3x speed up on the series of network snapshots of Cit-HepPh, Cit-HepTh, DBLP, Facebook, Flickr and YouTube, respectively.
- Compared with Batch [20], DynaMo obtains over 3x, 5x, 2x, 7x and 5x speed up on the series of network snapshots of Cit-HepPh, Cit-HepTh, DBLP, Facebook and Flickr, respectively. DynaMo spends nearly the same amount of time as Batch on YouTube network.
- Compared with QCA [15], DynaMo obtains over 2x, 2x, 4x and 5x speed up on the series of network snapshots of Cit-HepTh, Facebook, Flickr and YouTube, respectively. DynaMo is as efficient as QCA on DBLP network, and spends slightly more time on Cit-HepPh network than QCA.
- Compared with GreMod [16], DynaMo spends more time on most of the networks, and only performs better on the Flickr and YouTube network.
- Compared with LBTR [18], DynaMo is much more efficient than LBTR-SVM, and spends slightly more time than LBTR-LR on certain networks.

5.7 Summary of the Experimental Evaluation

DynaMo consistently outperforms all the other 5 dynamic algorithms on 6 real-world networks and 10,000 synthetic networks in terms of the effectiveness (i.e., modularity, NMI and ARI) of detecting communities. DynaMo has almost identical performance as Louvain in terms of the effectiveness, with only 0.27 to 1.7 percent lower modularity on certain networks. DynaMo also performs comparably well in terms of the efficiency. For instance, in terms of the cumulative elapsed time results, DynaMo outperforms Louvain, Batch and LBTR-SVM, and obtains similar performance as QCA and LBTR-LR. Even though GreMod acts more efficient than DynaMo, DynaMo is much more effective than GreMod (e.g., GreMod has the worst effectiveness performance running on nearly all datasets). In conclusion, DynaMo significantly outperformed the state-of-the-art dynamic algorithms in terms of effectiveness, and demonstrated much more efficient than the state-of-the-art static algorithm, Louvain algorithm, in detecting communities of dynamic networks, while also maintaining similar efficiency as the best set of competing dynamic algorithms.

6 Conclusion

In this paper, we proposed DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect communities in dynamic networks. We also present the theoretical guarantees to show why/how our operations could maximize the modularity, while avoiding redundant and repetitive computations. In the experimental evaluation, a comprehensive comparison has been made among our algorithm, Louvain algorithm and 5 other dynamic algorithms. Extensive experiments have been conducted on 6 real world networks and 10,000 synthetic networks. Our results show that DynaMo outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm.

References

[1] W. De Nooy, A. Mrvar, and V. Batagelj, *Exploratory Social Network Analysis with Pajek: Revised and Expanded Edition for Updated Software*, vol. 46. Cambridge, U.K.: Cambridge Univ. Press, 2018.

[2] L. Zhou, D. Wu, Z. Dong, and X. Li, "When collaboration hugs intelligence: Content delivery over ultra-dense networks," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 91–95, Dec. 2017.

[3] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 974–983.

[4] D. Zhuang and J. M. Chang, "PeerHunter: Detecting peer-to-peer botnets through community behavior analysis," in *Proc. IEEE Conf. Dependable Secure Comput.*, 2017, pp. 493–500.

[5] D. Zhuang and J. M. Chang, "Enhanced PeerHunter: Detecting peer-to-peer botnets through network-flow level community behavior analysis," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1485–1500, Jun. 2019.

[6] K. He, Y. Li, S. Soundarajan, and J. E. Hopcroft, "Hidden community detection in social networks," *Inf. Sci.*, vol. 425, pp. 92–106, 2018.

[7] Y. Zhang and K. Rohe, "Understanding regularized spectral clustering via graph conductance," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10 654–10 663.

[8] J. Meng, D. Fu, and T. Yang, "Semi-supervised soft label propagation based on mass function for community detection," in *Proc. 21st Int. Conf. Inf. Fusion*, 2018, pp. 1163–1170.

[9] M. E. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E*, vol. 74, no. 3, 2006, Art. no. 036104.

[10] F. Hao, L. Wang, Y. Sun, and D.-S. Park, "Detecting (k, r)-clique communities from social networks," in *Advanced Multimedia and Ubiquitous Engineering*. Berlin, Germany: Springer, 2018, pp. 583–590.

[11] Despite all its blunders, Facebook continues to attract new users, 2019. [Online]. Available: http://fortune.com/2019/01/30/facebook-users-increase-despite-scandals/

[12] 53 incredible Facebook statistics and facts, 2019. [Online]. Available: https://www.brandwatch.com/blog/facebook-statistics/

[13] Z. Dhouioui and J. Akaichi, "Tracking dynamic community evolution in social networks," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2014, pp. 764–770.

[14] N. İlhan and Ş. G. Öğüdücü, "Predicting community evolution based on time series modeling," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2015, pp. 1509–1516.

[15] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai, "Adaptive algorithms for detecting community structure in dynamic social networks," in *Proc. IEEE INFOCOM*, 2011, pp. 2282–2290.

[16] J. Shang *et al.*, "A real-time detecting algorithm for tracking community structure of dynamic networks," in *Proc. 6th Workshop Social Netw. Mining Anal. co-held KDD (SNA-KDD12)*, 2012.

[17] Y. Xin, Z.-Q. Xie, and J. Yang, "An adaptive random walk sampling method on dynamic community detection," *Expert Syst. Appl.*, vol. 58, pp. 10–19, 2016.

[18] J. Shang, L. Liu, X. Li, F. Xie, and C. Wu, "Targeted revision: A learning-based approach for incremental community detection in dynamic networks," *Physica A: Statistical Mech. Appl.*, vol. 443, pp. 70–85, 2016.

[19] P. Agarwal, R. Verma, A. Agarwal, and T. Chakraborty, "DyPerm: Maximizing permanence for dynamic community detection," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*, 2018, pp. 437–449.

[20] W. H. Chong and L. N. Teow, "An incremental batch technique for community detection," in *Proc. 16th Int. Conf. Inf. Fusion*, 2013, pp. 750–757.

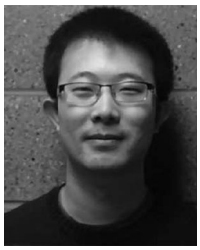
[21] G. Rossetti, L. Pappalardo, D. Pedreschi, and F. Giannotti, "Tiles: An online algorithm for community discovery in dynamic social networks," *Mach. Learn.*, vol. 106, no. 8, pp. 1213–1241, 2017.

[22] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Analyzing communities and their evolutions in dynamic social networks," *ACM Trans. Knowl. Discovery Data*, vol. 3, no. 2, 2009, Art. no. 8.

[23] Y.-R. Lin, J. Sun, H. Sundaram, A. Kelliher, P. Castro, and R. Konuru, "Community discovery via metagraph factorization," *ACM Trans. Knowl. Discovery Data*, vol. 5, no. 3, 2011, Art. no. 17.

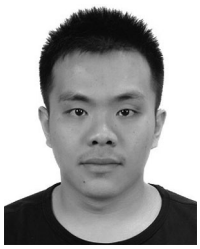
[24] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Statistical Mech.: Theory Experiment*, vol. 2008, no. 10, 2008, Art. no. P10008.

[25] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks," in *Proc. Int. Conf. Advances Social Netw. Anal. Mining*, 2010, pp. 176–183.
[26] L. Tang, H. Liu, and J. Zhang, "Identifying evolving groups in dynamic multimode networks," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 1, pp. 72–85, Jan. 2012.
[27] C. Guo, J. Wang, and Z. Zhang, "Evolutionary community structure discovery in dynamic weighted networks," *Physica A: Statistical Mech. Appl.*, vol. 413, pp. 565–576, 2014.
[28] A. Zakrzewska and D. A. Bader, "A dynamic algorithm for local community detection in graphs," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2015, pp. 559–564.
[29] X. Ma and D. Dong, "Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 5, pp. 1045–1058, May 2017.
[30] M. Cordeiro, R. P. Sarmento, and J. Gama, "Dynamic community detection in evolving networks using locality modularity optimization," *Social Netw. Anal. Mining*, vol. 6, no. 1, 2016, Art. no. 15.
[31] G. Rossetti and R. Cazabet, "Community discovery in dynamic networks: A survey," *ACM Comput. Surveys*, vol. 51, no. 2, 2018, Art. no. 35.
[32] M. E. Newman, "Modularity and community structure in networks," *Proc. Nat. Academy Sci. United States America*, vol. 103, no. 23, pp. 8577–8582, 2006.
[33] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, 2004, Art. no. 066111.
[34] J. Duch and A. Arenas, "Community detection in complex networks using extremal optimization," *Phys. Rev. E*, vol. 72, no. 2, 2005, Art. no. 027104.
[35] N. P. Nguyen, T. N. Dinh, Y. Shen, and M. T. Thai, "Dynamic social community detection and its applications," *PloS One*, vol. 9, no. 4, 2014, Art. no. e91431.
[36] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 11th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2005, pp. 177–187.
[37] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, *Graph Partitioning and Graph Clustering*. Providence, RI, USA: Amer. Math. Soc., 2013.
[38] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *Proc. 2nd ACM Workshop Online Social Netw.*, 2009, pp. 37–42.
[39] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the Flickr social network," in *Proc. 1st Workshop Online Social Netw.*, 2008, pp. 25–30.
[40] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas.*, 2007, pp. 29–42.
[41] G. Rossetti, "RDyn: Graph benchmark handling community dynamics," *J. Complex Netw.*, vol. 5, no. 6, pp. 893–912, 2017.



Di Zhuang (S'15) received the BE degree in computer science and information security from Nankai University, China. He is currently working toward the PhD degree in electrical engineering at the University of South Florida, Tampa. His research interests include cyber security, social network science, privacy enhancing technologies, machine learning, and deep learning. He is a student member of the IEEE.

J. Morris Chang (SM'08) received the PhD degree from North Carolina State University. He is a professor with the Department of Electrical Engineering, University of South Florida. His past industrial experiences include positions at Texas Instruments, the Microelectronic Center of North Carolina, and AT&T Bell Labs. He received the University Excellence in Teaching Award at the Illinois Institute of Technology in 1999. His research interests include cyber security, wireless networks, and energy efficient computer systems. In the last six years, his research projects on cyber security have been funded by DARPA. Currently, he is leading a DARPA project under the Brandeis program focusing on privacy-preserving computation over Internet. He is a handling editor of the *Journal of Microprocessors and Microsystems* and an editor of *IEEE IT Professional*. He is a senior member of the IEEE.



Mingchen Li received the MS degree in electrical engineering from the Illinois Institute of Technology. He is currently working toward the PhD degree in electrical engineering at the University of South Florida, Tampa. His research interests include cyber security, synthetic data generation, privacy enhancing technologies, machine learning, and data analytics.