Physica A 443 (2016) 70–85

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Targeted revision: A learning-based approach for incremental community detection in dynamic networks

Jiaxing Shang [a,*], Lianchen Liu [a], Xin Li [b], Feng Xie [a], Cheng Wu [a]

[a] Department of Automation, Tsinghua University, Beijing 100084, PR China

[b] Department of Information Systems, College of Business, City University of Hong Kong, Hong Kong Special Administrative Region

HIGHLIGHTS

- We propose a learning-based targeted revision (LBTR) approach for efficient incremental community detection.
- We provide mathematical analysis on how the vertex classifier can affect the community detection time complexity.
- Experiment results show that our approach can significantly reduce the running time while maintaining high community detection quality.
- To make our approach effective, one should increase the precision of the vertex classifier while keeping recall at a reasonable level.

ARTICLE INFO

Article history:
Received 1 April 2015
Received in revised form 12 May 2015
Available online 30 September 2015

Keywords:
Incremental community detection
Dynamic networks
Targeted revision
Computational complexity

ABSTRACT

Community detection is a fundamental task in network analysis. Applications on massive dynamic networks require more efficient solutions and lead to incremental community detection, which revises the community assignments of new or changed vertices during network updates. In this paper, we propose to use machine learning classifiers to predict the vertices that need to be inspected for community assignment revision. This learning-based targeted revision (LBTR) approach aims to improve community detection efficiency by filtering out the unchanged vertices from unnecessary processing. In this paper, we design features that can be used for efficient target classification and analyze the time complexity of our framework. We conduct experiments on two real-world datasets, which show our LBTR approach significantly reduces the computational time while keeping a high community detection quality. Furthermore, as compared with the benchmarks, we find our approach's performance is stable on both growing networks and networks with vertex/edge removals. Experiments suggest that one should increase the target classification precision while keeping recall at a reasonable level when implementing our proposed approach. The study provides a unique perspective in incremental community detection.

1. Introduction

Many real-world systems can be represented as networks, such as social networks [1], biological networks [2], citation networks [3], etc. Complex networks often exhibit a sense of community structure, where vertices form groups and have much denser connection within groups than between groups [4]. Community structure is a basic structural property of networks and can be used in various applications. For example, communities in protein interaction networks can be used to

* Corresponding author. Tel.: +86 15110099654.
E-mail address: shangjiaxing@gmail.com (J. Shang).

http://dx.doi.org/10.1016/j.physa.2015.09.072
0378-4371/

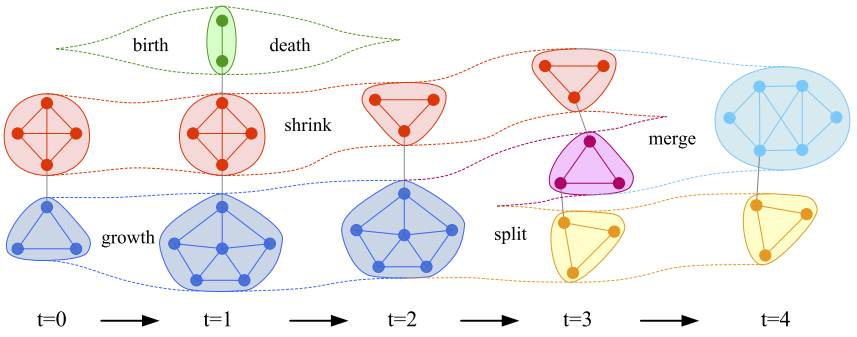


Fig. 1. Community evolution in a dynamic network.

predict the protein functions [5]. Communities in Webpage networks can be used for topic identification [6]. Previous works also studied the impact of community structure on epidemic spreading [7–9]. Community detection is a fundamental task in complex network analysis nowadays [4].

There are many community detection algorithms. A comprehensive review can be found in Ref. [10]. However, these algorithms generally consider networks to be static, while most real-world networks change over time. Recently, community detection in dynamic networks has attracted attention [11], which can help us understand how communities evolve [12–15], e.g., birth, death, growth, contracting (shrink), merging, splitting, etc., as shown in Fig. 1.

Community detection in dynamic networks can be addressed by applying static community detection algorithms multiple times on snapshots of the networks. However, it is more cost effective to incrementally revise the community structure of the old network when networks are updated [16–18], i.e., incremental community detection. Incremental community detection is more valuable when the networks are in mega-scale or change frequently, which is often the case in real networks.

Note that the essence of incremental community detection is the revision of community structure, if we can identify the high-risk vertices that need to be inspected, we argue it is possible to improve the efficiency of incremental algorithms. Holding this conjecture, in this paper, we propose a learning-based targeted revision (LBTR) approach. In this approach, we first classify the vertices that need to be revised. Then, we revise the community assignments of such vertices according to local modularity maximization [19]. In this paper, we provide mathematical analysis on how the classifier can affect the community detection time complexity. We conduct experiment on two real-world datasets to evaluate our approach, which shows that our proposed approach can significantly reduce the running time while maintaining community detection quality. Furthermore, as compared with the benchmarks, we find our approach's performance is stable on both growing networks and networks with vertex/edge removals. Experiments also suggest that, to make our approach effective, one should increase the precision of the vertex classifier while keeping recall at a reasonable level.

The rest of this paper is organized as follows. Section 2 reviews the literature on community detection. Section 3 elaborates the preliminaries of the problem to be addressed. Section 4 introduces our LBTR approach. Section 5 gives the evaluation framework, including the datasets, evaluation metrics, baseline methods, and experimental procedure. Experiment results are presented in Section 6. Section 7 concludes this paper.

2. Literature review

2.1. Community detection studies on static networks

There are many algorithms on community detection in static networks. Reader may refer to Ref. [10] for a comprehensive review. Here we only review a small number of papers that are more relevant to our paper.

Modularity-based algorithms are one major type of community detection algorithms, although they may have a resolution limit problem [20]. Modularity, first proposed by Newman et al. [21], is the fraction of connections within communities subtracts the expected fraction of connections within communities when vertices are randomly connected. Generally, a higher modularity value corresponds to a better community structure. Thus, the problem of community detection can be transformed to optimizing modularity. In Ref. [22] Clauset et al. proposed a greedy search algorithm (the CNM algorithm) to address this problem. It gradually merges communities that lead to the largest gain in modularity until convergence. In Ref. [23] Wakita et al. improved the time efficiency of the CNM algorithm by merging the communities in a more balanced way. To the best of our knowledge, the fastest modularity-based static algorithm is the Louvain algorithm [19]. The algorithm uses greedy strategies in a local manner. Initially, each vertex is put in a singleton community. Then the algorithm iteratively moves each vertex to its neighbor communities to maximize the gain in modularity. The procedure is then repeated at community level. Since moving a vertex to its neighbor community can be computed in $O(1)$ time, the algorithm is very efficient. The quality of the detected community structure also outperforms the CNM algorithm, as evaluated by the modularity measure.

In addition to modularity maximization, there are many other community detection paradigms, such as graph partitioning [24], spectral clustering [25], label propagation [26], heuristic algorithm [27], etc.

A small number of community detection studies address the problem by taking advantage of machine learning models. In Ref. [28] Backstrom et al. applied decision-trees to identify the most important structural determinants of community evolution. In Ref. [29] Newman and Leicht built a mixture model with the EM algorithm to model the group structure of networks. In Ref. [30] Hofman et al. build a stochastic block model using a variational Bayes algorithm to infer community assignments. In Ref. [31] Yang et al. proposed a dynamic stochastic block model for modeling communities and their evolution. In such studies, machine learning methods are generally used to model distribution of vertices and describe the community structure.

2.2. Incremental community detection

In massive and dynamic networks, incremental community detection becomes necessary for efficient processing [16–18]. The incremental community detection algorithms revise the old community assignments to generate new community structure when the network is updated.

Modularity maximization is one major approach to incremental community detection, which contains two basic methods: iterative execution and rule-based revision.

Iterative execution methods are built upon base algorithms such as the CNM algorithm or Louvain algorithm. Their design rationale is to convert the community structure in the last period to inputs upon which to re-run the base algorithms.

In Ref. [32] Dinh et al. developed an iterative execution method based on the CNM algorithm. When the network is updated, new vertices, neighbor vertices of new or removed vertices/edges (which are called changed vertices since they involve in network updates) are placed into singleton communities. Then the authors aggregate the network to community level and reapply the CNM algorithm to obtain the new community structure. Since the community-level network is much smaller than the original network, the algorithm is more efficient than the CNM algorithm. In Ref. [17] Bansal et al. conducted CNM-based iterative execution at vertex level. The algorithm (records and) replicates the vertex merging process of the last period's CNM before reaching new vertices or neighbor vertices of new or removed edges. Then the regular CNM is conducted to finish the community detection process. As compared with Ref. [32], Bansal et al.'s algorithm essentially puts new/changed vertices into singleton communities and also breaks their original community into smaller pieces for further processing by CNM.

The Louvain algorithm also naturally supports iterative execution. As done in Ref. [16], Aynaud et al. took community structure from the last period as the initial state and put each new vertex into a singleton community. Then the Louvain algorithm was applied to revise the community assignments of the vertices to maximize modularity locally. Chong et al. took a similar approach in Ref. [33], where new vertices and neighbor vertices of new or removed vertices/edges were put into singleton communities before reapplying the Louvain algorithm. Since this algorithm increased the vertices to be inspected for revision, its time complexity is increased and community detection quality is improved.

Rule-based revision uses predefined rules to specify how to revise vertices' community assignments. In Ref. [18] Nguyen et al. proposed a QCA algorithm which falls into this category. The algorithm provides revision strategies for each type of network updates: *newVertex*, *removeVertex*, *newEdge*, and *removeEdge*, where the community assignments of neighbor vertices are adjusted to maximize the gain in modularity. A similar approach was proposed by Shang et al. in Ref. [34].

There are also some incremental algorithms that are not based on modularity maximization. In Ref. [35] Falkowski et al. proposed a density-based incremental algorithm motivated by DBSCAN [36]. The authors defined core vertices, border vertices, and noise vertices based on a specified distance function. When the network updates, the algorithm recomputed the distances and locally adjusted the vertices whose distances to the core vertices were changed. In Ref. [37] Xie et al. proposed an incremental algorithm, LabelRankT, based on a Label Propagation algorithm, LabelRank [38]. When the network updated, LabelRankT re-initialized the labels of changed vertices and reapplied LabelRank to update the label probability distributions. Thus, it is also an iterative execution algorithm. In Ref. [39] Ning et al. proposed an spectral clustering algorithm for incremental community detection. They used incidence vector/matrix to represent the network updates (changes of vertices and edges). Then the incidence vector /matrix was used to incrementally update the eigenvalues of the vertices, which was used to derive community structure in spectral clustering. In Ref. [40] Sun et al. took a minimum description length (MDL) approach to detect communities, which aims at encoding a graph with a minimum number of bits. They divided the dynamic network into multiple sequential segments, and incrementally initialize the algorithm parameters by taking the community structure of last segment as input.

These aforementioned methods are all for non-overlapping communities, which is the focus of our research. There are also incremental algorithms for detecting overlapping community structure, such as iLCD [41], AFOCS [42], FacetNet [43], and the algorithm of Ref. [44].

Table 1 summarizes these incremental algorithms. As we can see, the incremental community detection algorithms are often based on the re-execution of static community detection algorithms. Thus the incremental algorithms' time efficiency depends on the base algorithm's efficiency. The first two algorithms are relatively slower due to CNM's higher cost than the Louvain algorithm.

Table 1
Summary of incremental community detection algorithms.

Studies	Algorithm type	Design	Base algorithm	Trigger changes
Dinh et al., 2009 [32]	Modularity maximization	Iterative execution	CNM	New and changed vertices → singleton communities
Bansal et al., 2011 [17]	Modularity maximization	Iterative execution	CNM	New and changed vertices → singleton communities; Some old communities → broken into small ones
Aynaud et al., 2010 [16]	Modularity maximization	Iterative execution	Louvain	New vertices → singleton communities
Chong et al., 2013 [33]	Modularity maximization	Iterative execution	Louvain	New and changed vertices → singleton communities
Nguyen et al., 2011 [18]	Modularity maximization	Rule-based	Louvain	New or removed vertices/ edges
Shang et al., 2012 [34]	Modularity maximization	Rule-based	Louvain	New edges
Falkowski et al., 2008 [35]	Density-based	Rule-based	DBSCAN	Vertices with changed distances to the core vertices
Xie et al., 2013 [37]	Label propagation	Iterative execution	LabelRank	New and changed vertices → label probability distribution re-initialized
Ning et al., 2007 [39]	Clustering	–	Spectral clustering	New or removed vertices; Edges whose weight were changed
Sun et al. 2007 [40]	Graph encoding	Iterative execution	MDL	A batch of new or removed vertices/edges

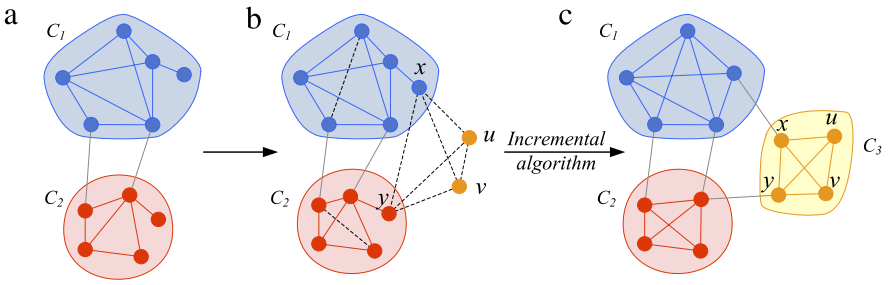


Fig. 2. An example of incremental community detection: (a) Network G^{t-1} with community structure $\mathcal{C}^{t-1}$ at time $t-1$; (b) Network G^t where the difference from G^{t-1} is shown by dotted lines; (c) Community structure $\mathcal{C}^t$ obtained based on $\mathcal{C}^{t-1}$ and G^t.

Furthermore, the incremental algorithms are generally based on the inspection of new or changed vertices, or other involved vertices/communities. If we can reduce the number of vertices to be inspected, it is possible we can improve the time efficiency of incremental community detection.

3. Preliminaries

Without loss of generality, we define an undirected unweighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. G has adjacency matrix A, where:

$$A_{i,j} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The community structure of G is a collection of vertex sets $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where community C_i has multiple vertices. (In this work, we only consider non-overlapping communities, so there is no overlap across C_i.)

Definition 1. A dynamic network is defined as a series of graphs $\mathcal{G} = \{G^0, G^1, G^2, \ldots, G^T\}$ where $G^t = (V^t, E^t)$ is the snapshot of the graph at time point t. The community structure of G^t is represented as $\mathcal{C}^t$.

Definition 2. Given a dynamic network $\mathcal{G}$, the problem of incremental community detection is to detect the community structure $\mathcal{C}^t$ of G^t based on G^t and $\mathcal{C}^{t-1}$, where the initial community structure $\mathcal{C}^0$ is provided (usually using static community detection algorithms).

Fig. 2 shows an example of the incremental community detection algorithm procedure. Fig. 2(a) shows the network at $t-1$ that consists of two communities C_1 and C_2. During period $t-1$ to t, two new vertices u and v are introduced and some edges are added, as shown in Fig. 2(b). The incremental algorithm adjusts the community structure of $\mathcal{C}^{t-1}$ and creates the new community C_3, forming $\mathcal{C}^t$, as shown in Fig. 2(c).

From time $t-1$ to t, the changes on the network include new vertices ($V^t \setminus V^{t-1}$), removed vertices ($V^{t-1} \setminus V^t$), new edges ($E^t \setminus E^{t-1}$), and removed edges ($E^{t-1} \setminus E^t$), where "\" is the symbol for set-theoretic difference. These changes cause the

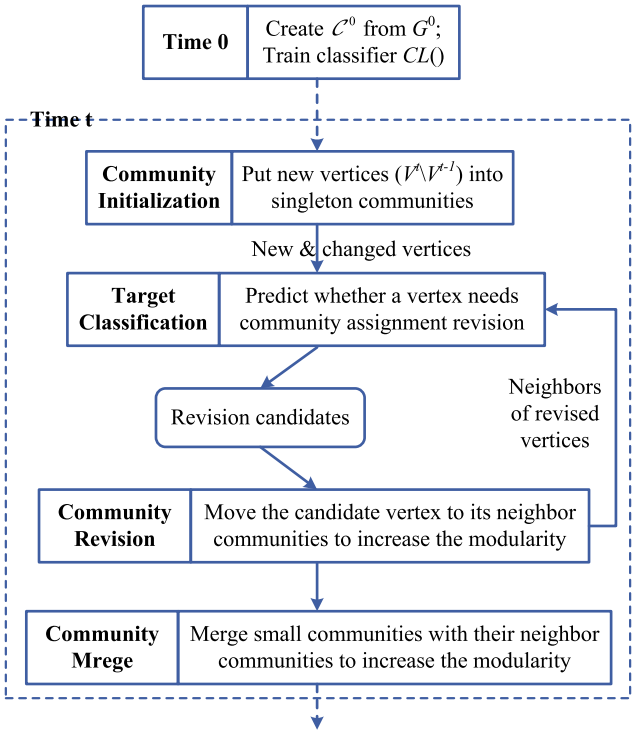


Fig. 3. The rationale for the learning-based targeted revision.

change of community structure. Together with these changes, the topological characteristics of vertices that are connected to the new or removed vertices or edges are also changed.

Definition 3. A changed vertex in a dynamic network is defined as a vertex adjacent to new or removed vertices/edges across time periods.

As shown in previous research, the changed vertices are critical to incremental community detection [16–18,32–35,37,39,40].

4. Learning-based targeted revision approach

4.1. Framework

Fig. 3 illustrates the rationale of our learning-based targeted revision (LBTR) approach, which is based on three rationales:

1. Local modularity maximization as in the framework of the Louvain algorithm.
2. Learning-based targeted revision to reduce the computations for local modularity maximization.
3. Small community merging to improve community detection quality.

Similar as previous studies, when the network evolved from G^{t-1} to G^t, we put the new vertices into singleton communities and move the community assignments of changed vertices to maximize the modularity gain.

However, different from existing methods, we do not inspect every new and changed vertex for modularity maximization. As highlighted in the target classification step of Fig. 3, we classify whether the new and changed vertices may need a community assignment revision when the network is updated from G^{t-1} to G^t. If we have an accurate (and efficient) classifier, we should be able to save time on unnecessary inspections of the vertices that will not need community revision. Note that if a vertex's community assignment is revised, it may also change the classification features of its neighbors. We will need to run the classifier on these neighbors and inspect their community assignments.

The final step is merging the small communities to maximize modularity. Specifically, we merge communities whose size is smaller than a threshold to their neighbor communities if that can increase modularity. The purpose of this extra step is to address the problem of dramatically increased number of communities as found in Ref. [18]. Experiments show that the use of this naive approach does not significantly change our overall time efficiency and community detection quality. (We can also employ the learning-based targeted revision approach to improve the time efficiency of this step. However, since the computational requirement of this step is low, we choose not to employ heuristics so that we can focus on the performance benefit in the first step in this paper.)

Algorithm 1 reports the pseudo-code of our proposed approach, which is called LBTR Algorithm in this paper.

Algorithm 1 The LBTR algorithm

Input:
Dynamic network $\mathcal{G} = \{G^0, G^1, G^2, \ldots, G^T\}$
Output:
Dynamic community structure $\mathcal{DC} = \{\mathcal{C}^0, \mathcal{C}^1, \mathcal{C}^2, \ldots, \mathcal{C}^T\}$
Initialize: Let $\mathcal{C}^0 = Louvain(G^0)$.
Generate training set S from G^0 and use S to train classifier CL.
for $t = 1$ to T **do**
  Let $\mathcal{X} = \Phi$
  for each new vertex u **do**
    Put u in a new singleton community.
  end for
  for each new and changed vertex u **do**
    Use CL to classify u. If u is positive, add u to $\mathcal{X}$
  end for
  while $\mathcal{X} \neq \Phi$ **do**
    Fetch next element u from $\mathcal{X}$, move u to its neighbor community to maximize ΔQ.
    If u is moved, for each neighbor v of u, use CL to classify v. If v is positive, add v to $\mathcal{X}$.
  end while
  for each community C **do**
    if $|C| \leq THRESHOLD$ **then**
      Merge C with its neighbor communities to maximize ΔQ.
    end if
  end for
  Output the current community structure as $\mathcal{C}^t$.
end for

Table 2
Candidate features for the learning module.

Feature	Description	Maintain cost
k_u	Degree of vertex u	$O(1)$
k_u^{in}	Number of connections from u to its community $C(u)$	$O(1)$
$\|C(u)\|$	Size of u's community $C(u)$	$O(1)$
$\|NC(u)\|$	Number of neighbor communities of vertex u	$O(k_u)$
k_u^{maxOut}	The maximum number of connections from u to its neighbor communities	$O(k_u)$

4.2. Detecting vertices with possible community revision

The detection of vertices whose community assignment should be revised can be considered as a binary classification problem in the machine learning paradigm. Although the idea is quite straightforward, the design of this classifier is non-trivial. We require the classifier to (1) have a good classification performance, i.e., high precision and recall, and (2) have a low computational cost. The first requirement is to ensure that the vertices with community assignment change are appropriately processed. The second requirement is to ensure the time efficiency of the overall algorithm.

4.2.1. Features

Feature design is the most critical part that affects the classifier's time efficiency in this paper. We do not assume knowing any specific information on the vertices and edges. We only employ the network's topological features to build the classifier. Since the network is dynamic, the topological features change over time and need to be recalculated continuously. This is where the computational cost comes from.

Table 2 lists some candidate features to build the classifier and the time complexity of maintaining them under simple network change events, such as adding/removing vertices/edges, moving a neighbor to another community, etc. In our approach, we select k_u and k_u^{in} to build the classifier. Both of them take $O(1)$ time to update and provide a good performance in classification. We have also tried some other features and found that they either provide limited performance improvement or require more time to update.

4.2.2. Classification algorithm

Selection of the classification model should also consider efficiency and effectiveness. In this research we did not make an effort to search classification algorithms. We choose two widely used classification models: Logistic Regression [45] and Support Vector Machines (SVM) [46]. Although the training of the two algorithms takes some time, in the prediction stage

that is being used in incremental community detection both models are quite efficient, since they are both based on linear combination of features.

4.2.3. Training data for classifier building

To build the classifier, we generate the training sets based on the initial network G^0. Assuming having more detailed timestamps within G^0, we convert G^0 to a dynamic networks $\{G_1^0, G_2^0\}$, where G_1^0 contains the first 80% of the network and G_2^0 is the update of G_1^0. With this setup, we can train the classifier based on the community reassignment of vertices across the two time periods. We run the Louvain algorithm on G_1^0 to obtain the initial community structure $\mathcal{C}_1^0$. Then the network is updated to G^0. Then, each new vertex in this update is put into a new singleton community. The Louvain algorithm is conducted to iteratively move the new and changed vertices to their neighbor communities to maximize the gain in modularity, in which the new and changed vertices that involved with community revision are recorded and used as positive instances of training data to build the classifier.

4.3. Algorithm analysis

4.3.1. Effect of the classifier

The classifier determines the vertices whose community assignment will be inspected and later revised. It significantly affects the time cost and community detection result.

When the classifier has high recall, i.e., most of the vertices that need to be inspected are inspected, the quality of the final result will be good. When the recall is low, a lot of vertices that need to be inspected for community revision are not inspected and the quality of the final result will be reduced.

When the classifier has a high precision, i.e., most of the inspected vertices later do involve community revision, the time complexity will be reduced. In contrast, when precision is low, the algorithm will process many vertices that do not lead to further community revision and the time complexity remains high.

Overall, we need a reasonable high recall to guarantee the quality of the community detection results. Given that, we should try to increase the precision to reduce the time complexity of the algorithm.

4.3.2. Time complexity

Now we give a mathematical analysis of the time complexity of this algorithm. The time complexity of the community merge step at the end of our algorithm is much lower than that of the vertex moving step; thus we only analyze the time complexity for the first step.

Proposition 1. *Assuming the modularity calculation of moving a vertex to one of its neighbor communities is a unit, the approximate time complexity of the LBTR algorithm is* $O(|\Delta V|\frac{\langle k\rangle rR}{(1-\langle k\rangle rR)P})$*, i.e.,* $O(|\Delta V|)$*, where* $|\Delta V|$ *is number of new and changed vertices from* $t-1$ *to* t*; R and P are the recall and precision of the classifier;* $\langle k\rangle$ *is the average vertex degree; and r is the probability that an examined vertex actually needs community assignment revision.*

Proof. For the first step of our algorithm, we have $|\Delta V|$ new or changed vertices go through the classifier. Among them there are $|\Delta V|r$ vertices that actually need community assignment revision. The classifier will hit a total number of $|\Delta V|rR$ vertices. The number of vertices that are classified as "need community revision" is $B_1 = |\Delta V|rR/P$, which is the number of vertices our algorithm will inspect.

Note that if a vertex is successfully moved, the algorithm will further consider moving its neighbors. In the $i-1$th iteration, the number of inspected vertices is B_{i-1}. There are $B_{i-1}P$ vertices successfully revised. These vertices have about $B_{i-1}P\langle k\rangle$ neighbors (some vertices may share neighbors which further reduces this number), which need to be inspected/classified in the ith iteration. Among them, the classifier further identifies $(B_{i-1}P\langle k\rangle)rR/P$ for modularity maximization to be conducted in the ith iteration.

Here we assume a unified r, R, and P for the multiple iterations of the algorithm. In practice, the algorithm will converge, and the number of vertices with revised communities will reduce, i.e., $B_i < B_{i-1}$ and $\langle k\rangle rR < 1$. So the total number of inspections over all the iterations is $\sum_i B_i \approx B_1/(1-\langle k\rangle rR)$.

The time complexity of inspecting one vertex for local modularity maximization is $O(\langle k\rangle)$ since the algorithm has to check with communities of all the vertex's neighbors to calculate modularity and find the best community to move. So the approximate time complexity of the LBTR algorithm is $O(|\Delta V|\frac{\langle k\rangle rR}{(1-\langle k\rangle rR)P})$, which is essentially $O(|\Delta V|)$. In this formula R and P depend on the classifier. $\langle k\rangle$ and r are determined by the characteristics of the data. In generic networks, we expect $\langle k\rangle$ and r change in a slow manner. □

5. Evaluation

5.1. Dataset

ArXiv: The arXiv citation dataset published in the KDD Cup 2003 includes approximately 29,000 papers. We consider each paper as a vertex and each citation as an edge, in which we treat the edges as unidirectional. After cleaning the data and removing the errors, the dataset has 27,769 papers from 1992 to 2003 and 351,798 edges among the papers.

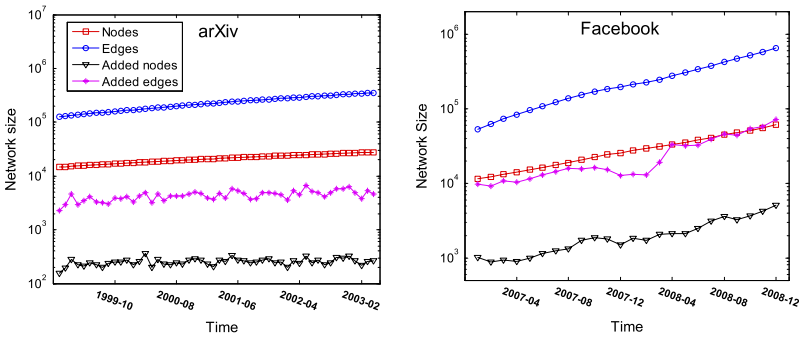


Fig. 4. The evolution of the cumulative network.

Facebook: The Facebook dataset [47] contains activities of Facebook users in the New Orleans area from October 2004 to January 2009, including their friendship relations and wall-post interactions. Each activity is associated with a timestamp indicating when a user appeared in another user's friend list or posted on another user's wall. After cleaning the data and removing the errors, the dataset has 61,382 people and 655,015 friendship edges.

5.2. Dynamic network construction

We take two approaches to build the dynamic networks from the two datasets for our experiments. First, we build a cumulative network containing all vertices and edges that have been introduced to the dataset from the beginning of the dataset to the examined time point. Second, we take a sliding window approach and build networks that consider only the vertices and edges that have been introduced in a time window before the examine time point. To practitioners, the cumulative network represents the network of registered users and the sliding window network represents the network of "active users". From an incremental community detection perspective, cumulative networks only have new vertices and edges between time points. Sliding window networks may contain the removal of vertices and edges.

5.2.1. Cumulative network

For the arXiv dataset, each paper and its citations have a timestamp (publication time). We build a series of cumulative networks till each month of 1999–2003, resulting in 52 networks. We use the papers published before 1999 as the first (initial) network to build the classifiers.

For the Facebook dataset, each person's activity (tagging new friends, leaving comments on walls) has a timestamp. We use the activities before 2007 to generate the initial network and build a series of cumulative networks till each month of 2007–2009, resulting in 24 incremental time points.

Fig. 4 reports the numbers of vertices, edges, new vertices, and new edges in the cumulative networks built up on the two datasets. As we can see, both networks show an exponential increase (showing as a straight line on the log-scaled chart). The number of new vertices and new edges also show an exponential increase over time. The growth of Facebook network is much faster than the arXiv network.

5.2.2. Sliding window network

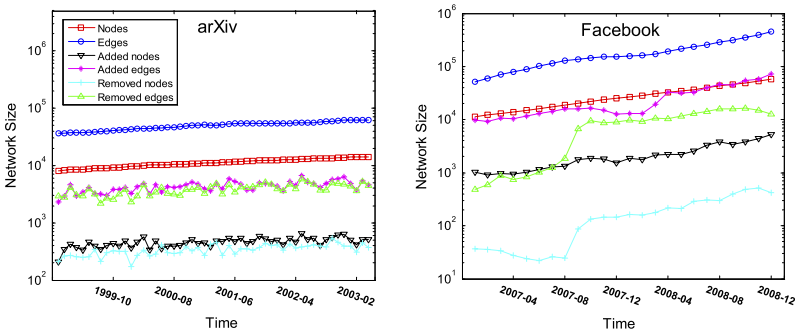
We build the sliding window networks for same time points as for cumulative networks in the two datasets. The networks only contain the published papers/user activities in the time window before those time points. In Fig. 5, we present the network size evolution for three sliding windows: one year, half a year, and one month. As we can see, the two datasets' sliding window networks generally follow an exponential growth as in cumulative networks.

Besides, a larger window size leads to large network size in Fig. 5. In fact, when the window size is one month (which is equal to the difference across our time points), all edges of the network will be replaced (as shown in Fig. 5(c)). In that case, it will not be reasonable to take an incremental community detection approach. Thus, we only experiment on window sizes as one year and half a year.

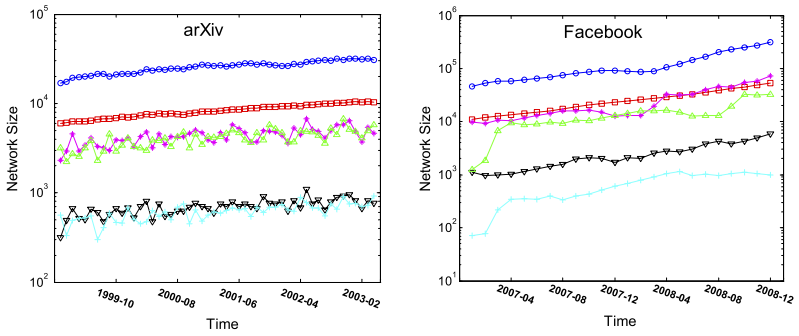
5.3. Baseline algorithms

We compare the performance of our algorithm with three state-of-the-art baseline algorithms:

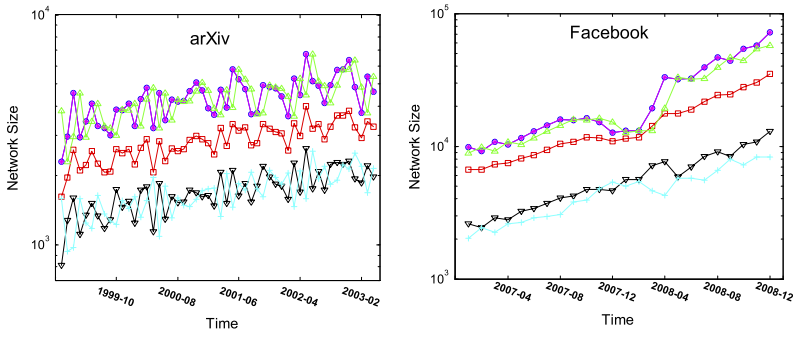
- *Louvain algorithm*: Louvain algorithm [19] is for static community detection. In each time point t, we apply the Louvain algorithm on G^t to get $\mathcal{C}^t$. Since the two datasets do not provide ground truth community structure, we also use the Louvain algorithm outputs as ground truth.



(a) Sliding window network (window size: one year).



(b) Sliding window network (window size: half a year).



(c) Sliding window network (window size: one month).

Fig. 5. The evolution of the sliding window networks.

- *BatchInc algorithm*: BatchInc [33] is an incremental Louvain algorithm that updates the community structure by putting new and changed vertices into singleton communities and then reapplying the Louvain algorithm to revise the community structure.
- *QCA*: QCA [18] is a rule-based incremental algorithm that updates vertices' community assignments according to different types of change events on the network. It is one of the most efficient incremental community detection algorithms when dealing with cumulative networks.

In this paper, we do not use the CNM-based algorithms as baseline since they are generally less efficient as compared to the Louvain-based algorithms.

5.4. Evaluation metrics

Similar to Ref. [18], we employ four metrics to evaluate the performance of our algorithm:

(1) *Running time*: Running time evaluates the time efficiency, which is the main purpose of incremental algorithms and the most important evaluation metric of this study.
(2) *Modularity*: Modularity [21] evaluates how well the community structure captures edges within communities. Higher modularity indicates more edges are captured within communities as compared with random networks.

Table 3
Precision and recall of the target vertex classifiers.

Network	Dataset	Logistic Regression		SVM	
		Precision	Recall	Precision	Recall
Cumulative network	ArXiv	0.387	0.998	0.463	0.991
	Facebook	0.386	0.997	0.410	0.996
One year sliding window	ArXiv	0.502	0.997	0.540	0.995
	Facebook	0.376	0.998	0.424	0.995
Half a year sliding window	ArXiv	0.633	0.998	0.586	0.995
	Facebook	0.415	0.998	0.443	0.996

(3) *NMI*: NMI [48] measures the similarity between the detected community structure and a gold standard, which is widely used to evaluate community detection algorithms. Here, we applied the static Louvain algorithm on each snapshot of the network to get the community structure as the gold standard NMI.
(4) *Number of communities*: The number of detected communities is not often used in community detection. However, a previous study [18] found a problem of dramatically increased number of communities in incremental community detection. Thus we also include it in this paper.

In addition to the absolute measures, we also use relative measures, i.e., the ratio of the measures to the Louvain algorithm's measures, in the evaluation. The static version of the Louvain algorithm generally provides the best results and takes the longest time in our experiments.

5.5. Experimental procedure

In the experiments, for each dynamic network we apply the Louvain algorithm on the initial network G^0 to extract the initial community structure $\mathcal{C}^0$. Then, we apply our proposed approach and the baseline methods on the incremental updates of networks in each month and evaluate their effectiveness using the four metrics.

Our proposed approaches have two instantiations, LBTR-LR and LBTR-SVM, which apply Logistic Regression and SVM on (k_u, k_u^{in}) to build the vertex classifiers, respectively. Here we apply logarithmic transformation on the features since they are counted data. We program the logistic regression model ourselves and use LIBSVM [49] to build the SVM classifier. The parameters of the Logistic Regression and SVM classifiers are obtained using 10-fold cross-validation. The threshold for merging small communities is 16 for the arXiv dataset and 6 for the Facebook dataset which are decided through small scale experiments.

Since the Louvain algorithm is non-deterministic, we repeat it on the initial network 30 times, which results in 30 realizations of initial community structure. Our follow-up experiments are thus also conducted 30 times, which allows us to calculate the error bands. In the experiments, we vary the vertex classifier parameters and the setup of the incremental detection task to inspect factors affecting our approach's performance.

The experiments are carried out on a computer with 2.5 GHz quad-core Intel Core i7 CPU and 16 GB memory. Our programs are in single process and single thread.

6. Results

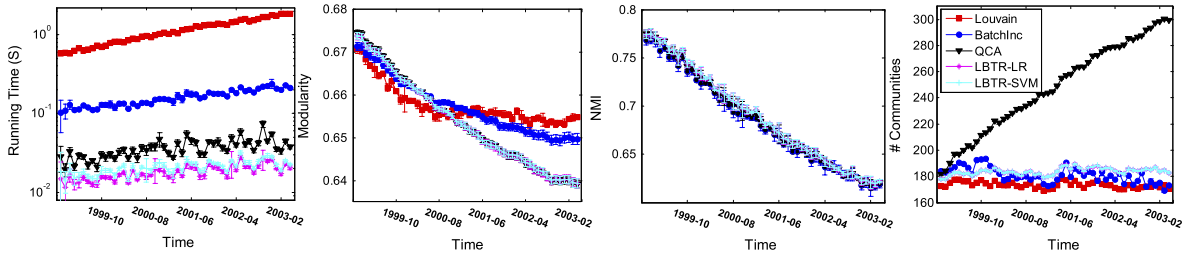
6.1. Community detection performance

Table 3 shows the precision and recall of the two classifiers in 10-fold cross-validation during parameter tuning. We built classifier based on both training data of cumulative network and sliding window network, which showed similar results. Generally, we tune all the classifiers to have a high recall, which ensures a low chance of missing the vertices that need to be inspected for community revision. The SVM algorithm has slightly higher precision and lower recall than the logistic algorithm in classifying vertices. The prediction results on sliding window network are generally more effective than that of the cumulative networks.

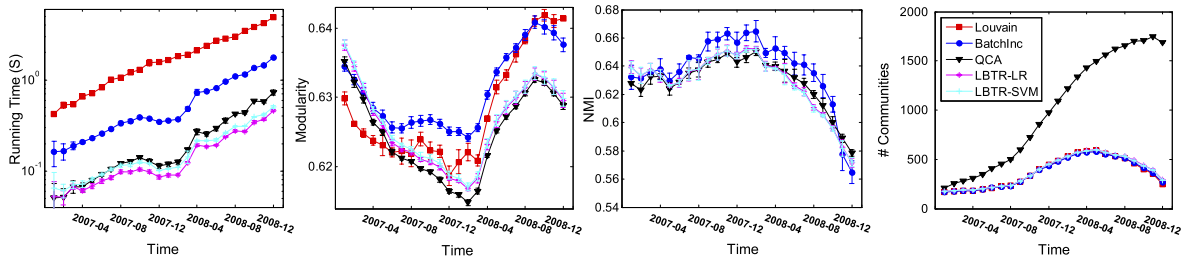
6.1.1. Results on cumulative networks

Fig. 6 shows the experimental results on cumulative networks. First, we observe that the LBTR algorithms outperform the three baseline algorithms in terms of running time. The Louvain algorithm takes the longest time, since it needs to reprocess the entire network at every time point. The BatchInc algorithm slightly improves on the Louvain algorithm but is still very slow.

The QCA algorithm is the fastest incremental algorithm in literature. However, it is slower than our proposed LBTR algorithms. Pair-wise t tests show that the LBTR-LR and LBTR-SVM algorithms spend significantly less time than it at the 99.9% confidence level ($p < 0.001$). The time difference is larger on larger networks, which shows the lower time complexity of LBTR algorithms according to network size. As compared with QCA, our approach on average reduces computational time by



(a) Results on arXiv dataset.



(b) Results on Facebook dataset.

Fig. 6. Experiment results on cumulative networks, as evaluated by running time, modularity, NMI, and number of communities. The error bars show std. dev.

about 50% on the arXiv dataset and about 30% on the Facebook dataset. The learning-based targeted revision approach is effective since it only needs to revise a small number of vertices' community assignment when the network updates overtime.

Second, in terms of modularity, the Louvain and BatchInc algorithms have relatively higher modularity in the second half of the time periods. The QCA algorithm and the two LBTR algorithms have similar performances on modularity. Pair-wise t tests show that they do not have significant difference.

In terms of NMI, all algorithms have a similar NMI (note that the Louvain algorithm is not on the figure since it is used as the gold standard to calculate NMI).

Last, all algorithms except QCA generate a similar number of communities. In fact, the number of communities problem is a recognized problem of QCA and is more severe on larger networks. For example, on the Facebook dataset it generates more than 1600 communities in the last time point while the other algorithms only generate about 250 to 300 communities. The last part of our algorithm successfully addressed this problem. Since that part has a low computational complexity, it can also be combined with the QCA algorithm to address the number of communities issue.

6.1.2. Results on sliding window networks

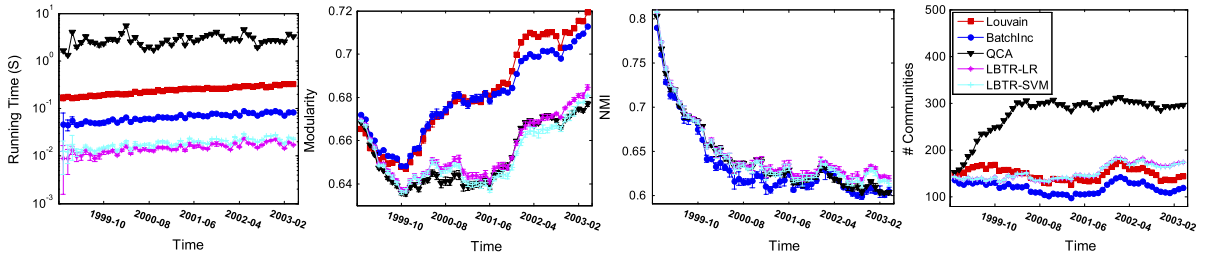
Fig. 7 shows the results on sliding window networks. Particularly, the time complexity of LBTR algorithms becomes more impressive on this set of networks.

First, we observe that the Louvain and BatchInc algorithms remain slow as compared with the LBTR algorithm. Although they are faster than the experiments on cumulative networks due to the reduced network size, their running times are about 5 to 50 times more than the LBTR algorithms.

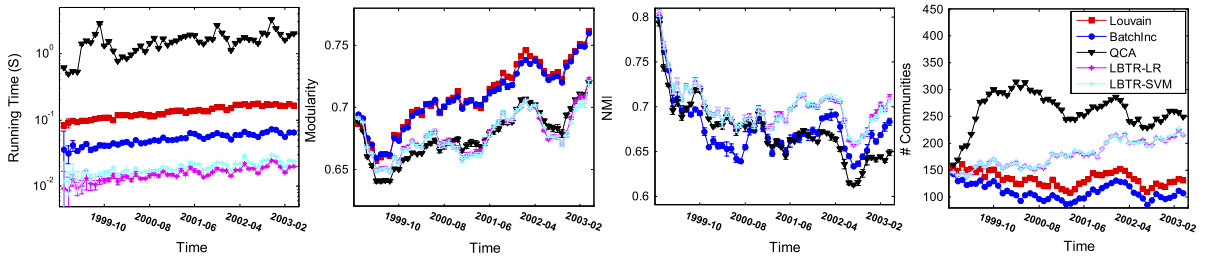
Second, we observe that the QCA algorithm is even slower than the Louvain and BatchInc algorithms. After carefully inspection, we noticed that the QCA algorithm was reported to be efficient on growing networks [18]. If the network contains removal of vertices or edges, the QCA algorithm needs to find quasi-cliques within communities, which is NP-hard [50]. In our experiments, we implemented an efficient greedy algorithm [51] for this step. However, it is still very slow to be dealt with, making QCA even slower than the Louvain algorithm. Before any new algorithms that fixed this problem, the QCA algorithm's running time is about 100 to 1000 times more than our proposed LBTR algorithms. (On the Facebook dataset, there are rapid increases of running time of the QCA algorithm at some time points. This is mainly due to the rapid increase of removed edges instead of the algorithm itself.)

In terms of the other measures, the Louvain and BatchInc algorithms are generally better, i.e., have higher modularity, higher NMI, and lower number of communities. However, the difference between them and the two LBTR algorithms is small. There is no statistically significant difference between QCA and the two LBTR algorithms in terms of modularity and NMI. On the number of community measure, QCA remains to have much more communities than other methods. These observations are consistent with the findings on cumulative methods.

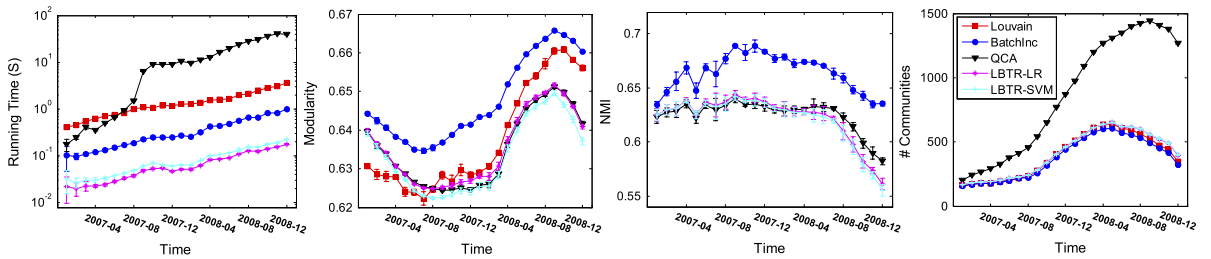
Overall, through the experiments on cumulative and sliding window networks, we find that the LBTR algorithms significantly reduced the time complexity while maintaining the community detection quality (in terms of modularity, NMI, and number of communities). Particularly, as compared with the baseline incremental algorithms, such as the QCA



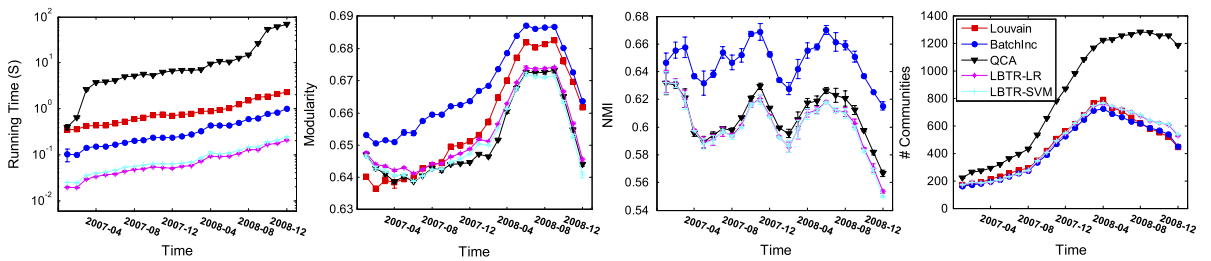
(a) Results on arXiv dataset (window size: one year).



(b) Results on arXiv dataset (window size: half a year).



(c) Results on Facebook dataset (window size: one year).



(d) Results on Facebook dataset (window size: half a year).

Fig. 7. Experiment results on sliding window networks, as evaluated by running time, modularity, NMI, and number of communities. The error bars show std. dev.

algorithms, our proposed approach provides stable performances on both growing networks and networks with vertex/edge removals, which is needed in real-world applications.

6.2. Effect of classifier performance

As shown in Proposition 1, the time complexity of the algorithm depends on the precision and recall of the classification model. To inspect the effect of classifiers on our algorithms' community detection performance, we experiment on 10 Logistic Regression and SVM classifiers with different precision and recall values.

The precision and recall values are modified by changing the ratio of positive and negative samples in the training data. Specifically, we randomly remove the positive or negative samples so that the ratio of positive and negative samples varies from about 1000:1 to 1:100. This variation provides us model parameters that deliver different precision and recall.

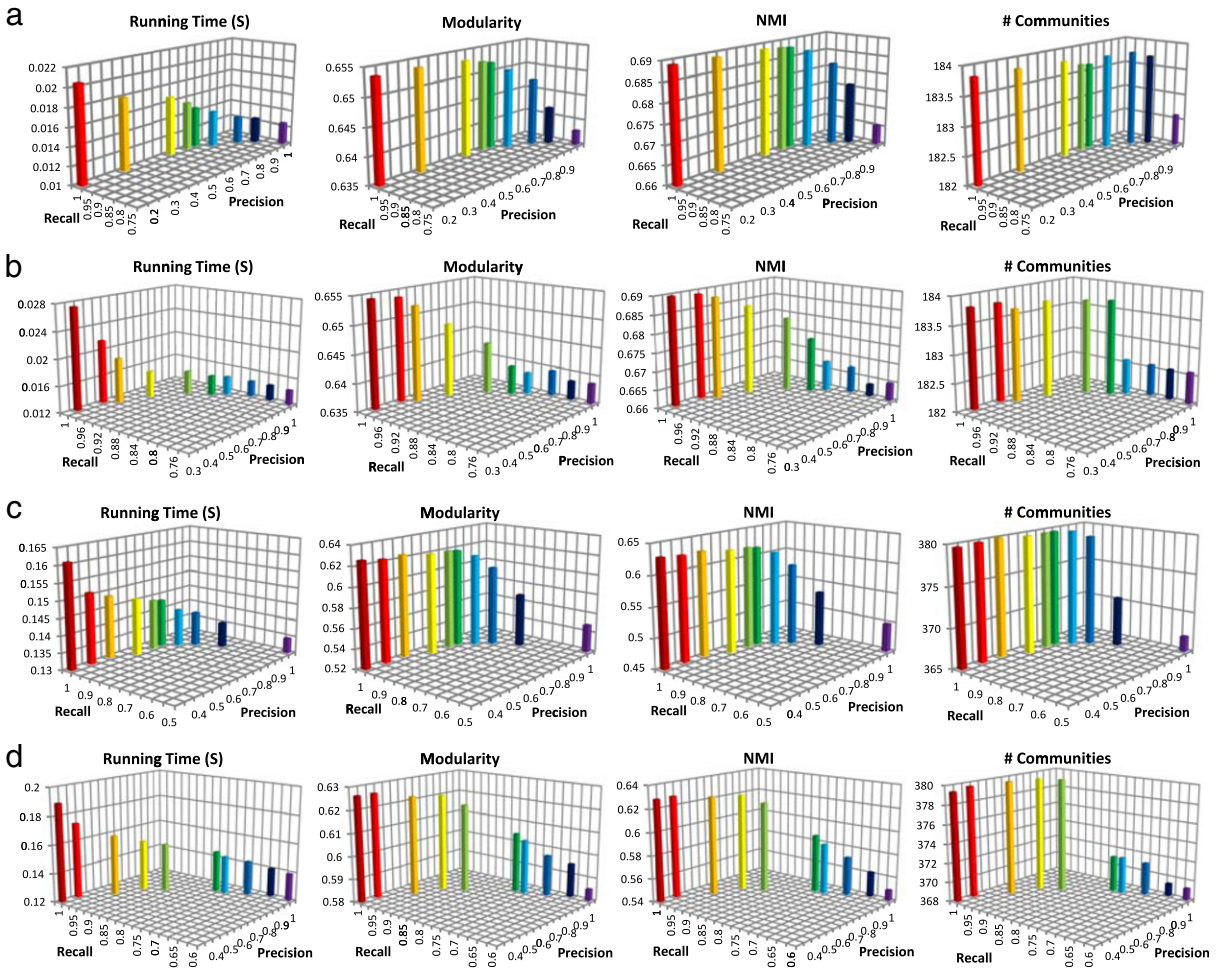


Fig. 8. The effect of precision and recall on cumulative networks. (a) Results on arXiv dataset with LR classifier; (b) Results on arXiv dataset with SVM classifier; (c) Results on Facebook dataset with LR classifier; (d) Results on Facebook dataset with SVM classifier. (Figures of (a) only show 9 bars, since there are two bars too close to be differentiated on the figure.)

For each algorithm setting, we conduct 30 experiments and report the average value of the evaluation metrics over time (for each month) in these experiments. Since there are two control variables, i.e., precision and recall, the results are represented as a 3D column chart, where the x axis is precision, the y axis is recall, and the z axis is the evaluation metrics.

We conducted experiments on all generated networks. The results are very much consistent across networks. Fig. 8 shows precision/recall's effects on the cumulative networks. Fig. 9 shows the results on the sliding window network with window size equal to one year. The results on the network with window size equal to half a year are omitted here.

From Figs. 8 and 9, we see that a lower recall and higher precision leads to a significant decrease of running time. This fits our theoretical analysis of the algorithm. For example, on the cumulative network of the arXiv dataset with the Logistic Regression classifier, the average running time can be reduced from 0.0205 to 0.0125 s, about a 40% reduction, if the recall reduces from 1 to 0.75 and precision increases from 0.16 to 1, as shown in Fig. 8(a).

At the same time, a higher recall leads to higher modularity, NMI, and the number of communities. The effect is more obvious on the Facebook dataset than on the arXiv dataset. We can observe that the change styles of modularity and NMI are different from that of running time. The performance reduces significantly when the recall decreases even if the precision increase is small. When recall remains at a relatively high level, the community detection quality is generally high. For example, by applying the Logistic Regression classifier on the cumulative network of the Facebook dataset, modularity only reduces from 0.625 to 0.615 when recall is reduced from 1 to 0.95, as shown in Fig. 8(c). Note that in the same process, precision increases from 0.4 to 0.85. When the recall further reduces to 0.5 and precision increases to 1, modularity reduces to 0.54. This phenomenon indicates that community detection quality is mainly driven by recall rather than precision.

In practice, precision and recall are correlated. Higher recall often appears with a lower precision. The findings here provide us directions to tune learning-based targeted revision models: we should increase precision (to reduce running time) until recall begins to reduce significantly.

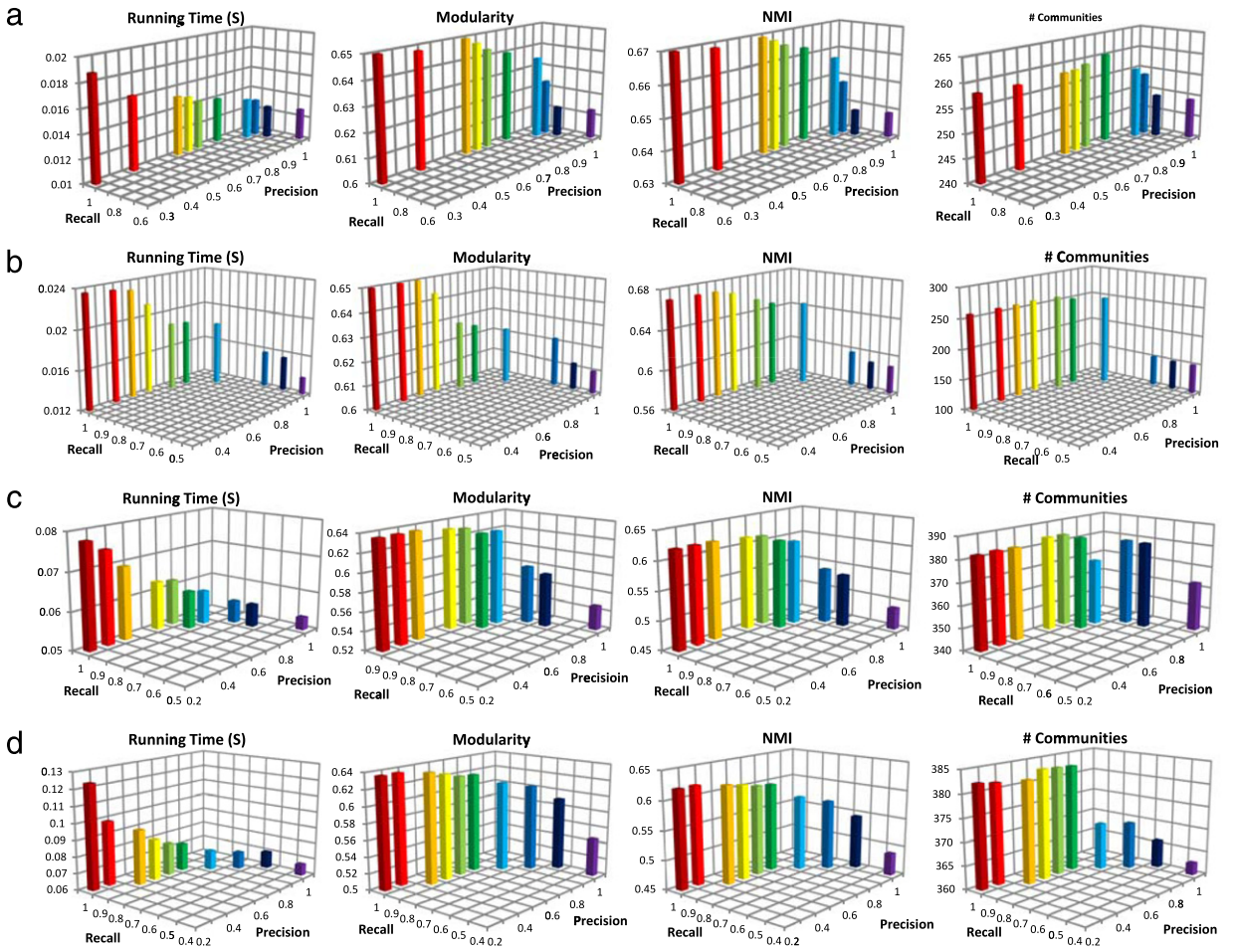


Fig. 9. The effect of precision and recall on one year sliding window networks. (a) Results on arXiv dataset with LR classifier; (b) Results on arXiv dataset with SVM classifier; (c) Results on Facebook dataset with LR classifier; (d) Results on Facebook dataset with SVM classifier.

6.3. Effect of network change

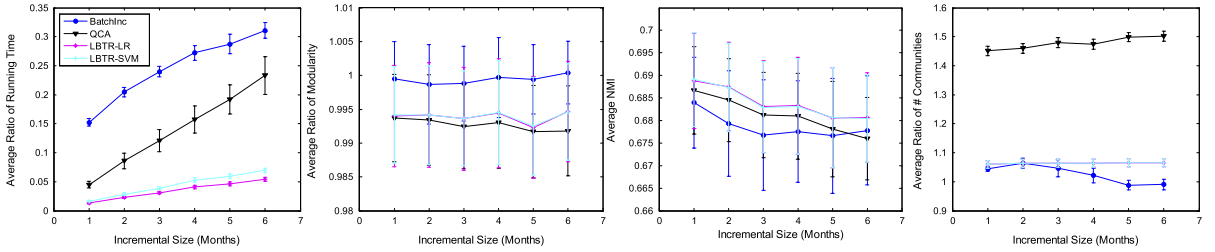
According to Proposition 1, the time complexity of the LBTR incremental algorithm is proportional to $|\Delta V|$, i.e., more changes in networks lead to higher running time. To further illustrate this effect, we conduct experiments by varying the length of the period to run the community detection algorithm from one month to six months.

In our experiments the cumulative networks and sliding window network show similar results, so we only report the results on the cumulative networks, as shown in Fig. 10. To make it easier to read, we report the average ratio of different algorithms' measures to the Louvain algorithm in 30 runs of experiments.

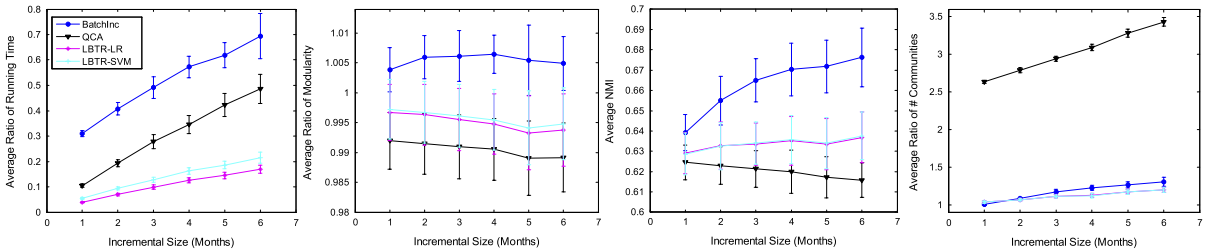
As shown in Fig. 10, the running times of the algorithms are generally proportional to the length of update period, since longer update period leads to a larger number of updated vertices and edges. However, the increase in running time of our LBTR approach is much slower than other methods, which further shows the advantage of our approach as compared with others. In terms of community detection quality, the modularity and NMI slightly decrease with the increase of update period. There are no significant performance differences across algorithms (the error bars on standard deviation overlap with each other). The update period increase also increases the number of communities in QCA, which does not affect other algorithms much, including our algorithm.

7. Discussion and conclusion

In this paper, we propose a learning-based targeted revision approach for incremental community detection in dynamic networks. We employ machine learning models to identify the vertices whose community assignment needs to be revised. This approach significantly reduces the computational power needed to process changed vertices. We provide mathematical analysis on the relationship between the approach's time complexity and the accuracy of the vertex classifier. We employ



(a) Results on arXiv dataset.



(b) Results on Facebook dataset.

Fig. 10. The effect of incremental size on cumulative networks. The error bars show std. dev.

two real-world datasets to validate our approach and find that our proposed LBTR approach can significantly reduce the computational time while maintaining community detection quality. Particularly,

(1) Our approach's performance is stable on both growing networks and networks with vertex/edge removals. On the cumulative networks, our approach can reduce running time for about 30%–50% as compared with the most efficient benchmark, the QCA algorithm. On the sliding window networks, where QCA algorithm fails, our approach can reduce running time for about 80%–90% as compared with the most efficient benchmark BatchInc.
(2) The running time of our approach increases much slower than other approaches when the inspection period increases and the involved updated vertices and edges increase.

Experiments also show that to maintain a high community detection quality and reduce time complexity, we should increase the precision of the target vertex classifier while keeping its recall at a relatively high level.

To the best of our knowledge, our approach makes the first effort to apply machine learning in community assignment revision for incremental community detection, which provides a new perspective in investigating incremental community detection.

Our work can be extended in several directions in the future. Firstly, it is necessary to explore other classification models and features to improve the performance of the vertex classifier and the incremental community detection algorithm. Secondly, in this paper, we only consider the detection of non-overlapping communities. In the future, we will study how to apply the LBTR approach on incremental detection of overlapping community structures.

Acknowledgments

We appreciate the anonymous reviewer's valuable comments. This work was partly supported by the foundation from the "China Equipment and Resource Sharing" project (Nos. 025-226009002, 226009003, Tsinghua University). This work was also partially supported by the National Natural Science Foundation of China grant 71572169, GuangDong Natural Science Foundation grant 2015A030313876, and CityU SRG 7004287. All opinions are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] S. Wasserman, Social Network Analysis: Methods and Applications, Vol. 8, Cambridge University Press, 1994.
[2] M.E. Newman, The structure and function of complex networks, SIAM Rev. 45 (2) (2003) 167–256.
[3] M.E. Newman, Coauthorship networks and patterns of scientific collaboration, Proc. Natl. Acad. Sci. 101 (Suppl. 1) (2004) 5200–5205.
[4] M. Girvan, M.E. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. 99 (12) (2002) 7821–7826.
[5] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature 435 (7043) (2005) 814–818.
[6] Y. Dourisboure, F. Geraci, M. Pellegrini, Extraction and classification of dense communities in the web, in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 461–470.
[7] X. Wu, Z. Liu, How community structure influences epidemic spread in social networks, Physica A 387 (2) (2008) 623–630.

[8] J. Chen, H. Zhang, Z.-H. Guan, T. Li, Epidemic spreading on networks with overlapping community structure, Physica A 391 (4) (2012) 1848–1854.
[9] J. Shang, L. Liu, X. Li, F. Xie, C. Wu, Epidemic spreading on complex networks with overlapping and non-overlapping community structure, Physica A 419 (2015) 171–182.
[10] S. Fortunato, Community detection in graphs, Phys. Rep. 486 (3) (2010) 75–174.
[11] T. Aynaud, E. Fleury, J.-L. Guillaume, Q. Wang, Communities in evolving networks: Definitions, detection, and analysis techniques, in: Dynamics On and of Complex Networks, Vol. 2, Springer, 2013, pp. 159–200.
[12] G. Palla, A.-L. Barabási, T. Vicsek, Quantifying social group evolution, Nature 446 (7136) (2007) 664–667.
[13] S. Asur, S. Parthasarathy, D. Ucar, An event-based framework for characterizing the evolutionary behavior of interaction graphs, ACM Trans. Knowl. Discov. Data (TKDD) 3 (4) (2009) 16.
[14] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, B.L. Tseng, Analyzing communities and their evolutions in dynamic social networks, ACM Trans. Knowl. Discov. Data (TKDD) 3 (2) (2009) 8.
[15] D. Greene, D. Doyle, P. Cunningham, Tracking the evolution of communities in dynamic social networks, in: Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on, IEEE, 2010, pp. 176–183.
[16] T. Aynaud, J.-L. Guillaume, Static community detection algorithms for evolving networks, in: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2010 Proceedings of the 8th International Symposium on, IEEE, 2010, pp. 513–519.
[17] S. Bansal, S. Bhowmick, P. Paymal, Fast community detection for dynamic complex networks, in: Complex Networks, Springer, 2011, pp. 196–207.
[18] N.P. Nguyen, T.N. Dinh, Y. Xuan, M.T. Thai, Adaptive algorithms for detecting community structure in dynamic social networks, in: INFOCOM, 2011 Proceedings IEEE, IEEE, 2011, pp. 2282–2290.
[19] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Stat. Mech. Theory Exp. 2008 (10) (2008) P10008.
[20] S. Fortunato, M. Barthélemy, Resolution limit in community detection, Proc. Natl. Acad. Sci. 104 (1) (2007) 36–41.
[21] M.E. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[22] A. Clauset, M.E. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004) 066111.
[23] K. Wakita, T. Tsurumi, Finding community structure in mega-scale social networks: [extended abstract], in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 1275–1276.
[24] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (2) (1970) 291–307.
[25] H.-W. Shen, X.-Q. Cheng, Spectral methods for the detection of network community structure: a comparative analysis, J. Stat. Mech. Theory Exp. 2010 (10) (2010) P10020.
[26] U.N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (3) (2007) 036106.
[27] D. Chen, Y. Fu, M. Shang, A fast and efficient heuristic algorithm for detecting community structures in complex networks, Physica A 388 (13) (2009) 2741–2749.
[28] L. Backstrom, D. Huttenlocher, J. Kleinberg, X. Lan, Group formation in large social networks: membership, growth, and evolution, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 44–54.
[29] M.E. Newman, E.A. Leicht, Mixture models and exploratory analysis in networks, Proc. Natl. Acad. Sci. 104 (23) (2007) 9564–9569.
[30] J.M. Hofman, C.H. Wiggins, Bayesian approach to network modularity, Phys. Rev. Lett. 100 (25) (2008) 258701.
[31] T. Yang, Y. Chi, S. Zhu, Y. Gong, R. Jin, Detecting communities and their evolutions in dynamic social networks—a Bayesian approach, Mach. Learn. 82 (2) (2011) 157–189.
[32] T.N. Dinh, Y. Xuan, M.T. Thai, Towards social-aware routing in dynamic communication networks, in: Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International, IEEE, 2009, pp. 161–168.
[33] W.H. Chong, L.N. Teow, An incremental batch technique for community detection, in: Information Fusion (FUSION), 2013 16th International Conference on, IEEE, 2013, pp. 750–757.
[34] J. Shang, L. Liu, F. Xie, Z. Chen, J. Miao, X. Fang, C. Wu, A real-time detecting algorithm for tracking community structure of dynamic networks, in: 6th SNA-KDD Workshop, ACM, 2012.
[35] T. Falkowski, A. Barth, M. Spiliopoulou, Studying community dynamics with an incremental graph mining algorithm, in: AMCIS 2008 Proceedings, Vol. 29 (2008).
[36] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, Vol. 96, 1996, pp. 226–231.
[37] J. Xie, M. Chen, B.K. Szymanski, Labelrankt: Incremental community detection in dynamic networks via label propagation, in: Proceedings of the Workshop on Dynamic Networks Management and Mining, ACM, 2013, pp. 25–32.
[38] J. Xie, B.K. Szymanski, Labelrank: A stabilized label propagation algorithm for community detection in networks, in: Network Science Workshop (NSW), 2013 IEEE 2nd, IEEE, 2013, pp. 138–143.
[39] H. Ning, W. Xu, Y. Chi, Y. Gong, T.S. Huang, Incremental spectral clustering with application to monitoring of evolving blog communities, in: SDM, SIAM, 2007, pp. 261–272.
[40] J. Sun, C. Faloutsos, S. Papadimitriou, P.S. Yu, Graphscope: parameter-free mining of large time-evolving graphs, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2007, pp. 687–696.
[41] R. Cazabet, F. Amblard, C. Hanachi, Detection of overlapping communities in dynamical social networks, in: Social Computing (SocialCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 309–314.
[42] N.P. Nguyen, T.N. Dinh, S. Tokala, M.T. Thai, Overlapping communities in dynamic networks: their detection and mobile applications, in: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, ACM, 2011, pp. 85–96.
[43] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, B.L. Tseng, Facetnet: a framework for analyzing communities and their evolutions in dynamic networks, in: Proceedings of the 17th International Conference on World Wide Web, ACM, 2008, pp. 685–694.
[44] M. Takaffoli, R. Rabbany, O.R. Zaïane, Incremental local community identification in dynamic social networks, in: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM, 2013, pp. 90–94.
[45] D.A. Freedman, Statistical Models: Theory and Practice, Cambridge University Press, 2009.
[46] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[47] B. Viswanath, A. Mislove, M. Cha, K.P. Gummadi, On the evolution of user interaction in facebook, in: Proceedings of the 2nd ACM workshop on Online social networks, ACM, 2009, pp. 37–42.
[48] L. Danon, A. Diaz-Guilera, J. Duch, A. Arenas, Comparing community structure identification, J. Stat. Mech. Theory Exp. 2005 (09) (2005) P09008.
[49] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 27.
[50] J. Hastad, Clique is hard to approximate within n 1-&epsiv, in: Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, IEEE, 1996, pp. 627–636.
[51] J. Abello, M.G. Resende, S. Sudarsky, Massive quasi-clique detection, in: LATIN 2002: Theoretical Informatics, Springer, 2002, pp. 598–612.