Community Detection in Social Networks: An In-depth Benchmarking Study with a Procedure-Oriented Framework *

Meng Wang[1], Chaokun Wang[1], Jeffrey Xu Yu[2], Jun Zhang[1]
[1] Tsinghua University, Beijing 100084, China
[2] The Chinese University of Hong Kong, Hong Kong, China
{meng-wang12, zhang-jun10}@mails.tsinghua.edu.cn, chaokun@tsinghua.edu.cn, yu@se.cuhk.edu.hk

ABSTRACT

Revealing the latent community structure, which is crucial to understanding the features of networks, is an important problem in network and graph analysis. During the last decade, many approaches have been proposed to solve this challenging problem in diverse ways, i.e. different measures or data structures. Unfortunately, experimental reports on existing techniques fell short in validity and integrity since many comparisons were not based on a unified code base or merely discussed in theory.

We engage in an in-depth benchmarking study of community detection in social networks. We formulate a generalized community detection procedure and propose a procedure-oriented framework for benchmarking. This framework enables us to evaluate and compare various approaches to community detection systematically and thoroughly under identical experimental conditions. Upon that we can analyze and diagnose the inherent defect of existing approaches deeply, and further make effective improvements correspondingly.

We have re-implemented ten state-of-the-art representative algorithms upon this framework and make comprehensive evaluations of multiple aspects, including the efficiency evaluation, performance evaluations, sensitivity evaluations, etc. We discuss their merits and faults in depth, and draw a set of take-away interesting conclusions. In addition, we present how we can make diagnoses for these algorithms resulting in significant improvements.

1. INTRODUCTION

Intrinsic community structures are possessed by many real-world networks, e.g. biological data, communication networks and social graphs, to name but a few. Given a network, it is particularly interesting as well as challenging to detect the inherent and hidden communities. *Communities*, which have no quantitative definition, are also called clusters. They are usually considered as groups of nodes, in which intra-group connections are much denser than those inter-group ones. Just as many classic puzzles, community detection is intuitive at first sight but actually an intricate problem.

Community detection [8] aims at grouping nodes in accordance with the relationships among them to form strongly linked subgraphs from the entire graph [26]. Since networks are usually modeled as graphs, detecting communities in multifarious networks is also known as the graph partition problem in modern graph theory [7, 2], as well as the graph clustering [1] or dense subgraph discovery problem [16] in the graph mining area. In the last decade, lots of solutions have emerged in the literature [5, 9, 19, 24, 14, 25, 12, 3, 29, 4, 11], trying to solve this problem from various perspectives.

The extensive research work has promoted the prosperity of the family of community detection approaches. However, it also raises a new difficulty, how to choose the most appropriate approach in specific scenarios, since many latest approaches have not been compared with each other upon unified platforms with same datasets and uniform configurations. Given the huge diversity of various approaches, it is usually not easy to analyze, compare and evaluate the extensive existing work. In this sense, a general benchmark for community detection is quite necessary and beneficial. In this paper, we make a benchmarking study for community detection, which contains a universal procedure-oriented framework and a comprehensive evaluation system. Upon that, we are able to analyze, evaluate, diagnose and further improve the existing approaches thoroughly, and get interesting and credible conclusions.

1.1 Challenges

An in-depth benchmarking study for community detection is non-trivial and poses a set of unique challenges.

Firstly, considering the various existing approaches, the lack of a procedure-oriented framework for community detection makes it a puzzle to understand, compare and diagnose them. Since these approaches are of various categories, a universal framework of community detection is quite difficult to be summarized and abstracted.

Secondly, to make a fair comparison and build a general benchmark for evaluation, it is a necessity to re-implement different approaches of various categories based on a common code base. Actually, the re-implementation is really a tough work.

Finally, when proposing a new approach, authors often testify their work via limited metrics that perform well. In our benchmarking evaluation, we need a suite of metrics which can embody full structural characteristics of communities to evaluate the approaches as comprehensively and thoroughly as possible.

There exist two pieces of research work similar to ours. Yang et al. only investigated the performances of different metrics for communities with ground-truth [28]. Xie et al. made an evaluation on overlapping community detection [27]. However, they failed to present a universal framework. Instead, we conduct a systematic in-depth benchmarking study to solve the above challenges.

1.2 General Benchmark

In this paper, we have designed a benchmark for community detection. As shown in Fig. 1, our benchmark consists of four core

*Corresponding author: Chaokun Wang.

Proceedings of the VLDB Endowment, Vol. 8, No. 10

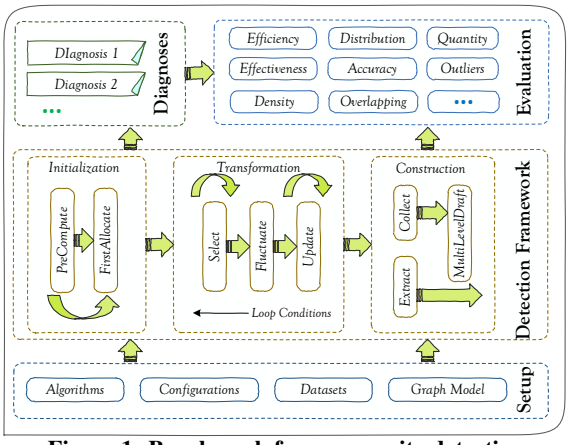


Figure 1: Benchmark for community detection

modules: (1) **Setup**, including a set of algorithms (Sec. 2.2), real-world and synthetic datasets (Sec. 6.1), parameter configurations (Sec. 6.2), and a unified graph model converted from the datasets; (2) **Detection Framework**, a generalized detection procedure with high abstraction of the common workflow of community detection (the details of the framework are introduced in Sec. 3; the procedure mappings in Sec. 4); (3) **Diagnoses**, which provide targeted diagnoses on these algorithms based on our framework, leading to directions of improvement over the existing work (Sec. 5); (4) **Evaluation**, a comprehensive evaluation system for community detection from different aspects (Sec. 6.3–6.11).

The benchmark contains a universal framework which abstracts the key factors, phases and steps from many approaches to community detection tasks, and makes it easy to implement classical or latest algorithms for comparison. Moreover, it consists of a comprehensive suite of widely-recognized metrics for evaluation of various concerned aspects, including the efficiency evaluation on the time cost, performance evaluations on accuracy and effectiveness, sensitivity evaluations on network density and mixture degree, and additional evaluations on community distribution and the ability to avoid excessive outliers. By modularizing and separating key factors and steps, our framework allows us to study the strength and weakness of each algorithm thoroughly, and make diagnoses and targeted prescriptions for improvement. In this benchmark we provide a common code base with algorithms implemented in the same environment, and thus make the comparison more fair and credible.

1.3 Contributions

We have conducted a comprehensive benchmarking study which focuses on the in-depth analysis, evaluation and comparison of the extensive work. To the best of our knowledge, this is the first work on the benchmarking study with a generalized framework on non-overlapping community detection techniques. We make the following main contributions:

- We propose a novel procedure-oriented framework by formulating a generic workflow of community detection via abstracting and modularizing the key factors and steps.
- We review the family of community detection approaches, and re-implement ten state-of-the-art representative algorithms in a common code base (using standard C++) by mapping them to the framework based on their specifics.
- We make in-depth evaluations on these approaches based on our benchmark using both real-world and synthetic datasets.
- We draw a set of interesting take-away conclusions, and provide intuitive and brief ratings on concerned algorithms.
- We also present how to make diagnoses for existing approaches, leading to significant performance improvements.

The remainder of this paper is organized as follows. We formulate the problem of community detection and sketch out existing work in Sec. 2. In Sec. 3 we propose a universal framework for benchmarking in community detection, and then in Sec. 4 we map the existing approaches to the framework. Afterwards we present how to make targeted diagnoses based on the framework in Sec. 5. We evaluate these approaches with our benchmark and report the results and findings in Sec. 6, and conclude this study in Sec. 7.

2. PRELIMINARY AND BACKGROUND

As preliminaries, we first define basic concepts and the problem of community detection, and then review the existing approaches.

2.1 Problem Definition

Social Networks. A *social network* with n individuals and m social ties can be denoted as $G(V,E)$, where V is the set of nodes, $|V| = n$, and E is the set of undirected relationships, $E \subseteq V \times V$, $|E| = m$. A social network is also referred to herein as a *graph*.

Communities. Non-overlapping communities are not confined to a graph partition, and clusters which incompletely cover the graph are usually more desirable. Here we define the *communities* as a list of non-empty node subsets: $Coms = \{V_1^{'}, \cdots, V_{cn}^{'}\}$, where $\bigcup_{i=1}^{cn} V_i^{'} \subseteq V$, and cn is the total number of communities. Please note $Coms$ should try to satisfy $V_i^{'} \bigcap V_j^{'} = \emptyset$. A community is also referred to as a *cluster* or a *part*.

Outliers. Since community detection does not force each node into a certain group, some independent nodes, which cannot be grouped into any communities, are allowed far outside the detected groups [13]. We define them as *outliers*: $Outs = \{v | v \in V, \nexists V_i^{'} \in Coms \wedge v \in V_i^{'}\} = V - \bigcup_{i=1}^{cn} V_i^{'}$. It is worth mentioning that outliers can be directly identified by original algorithms or be produced by disbanding the tiny groups, whose sizes are less than the predefined threshold of *minimal valid size* (*mvs*) of communities.

PROBLEM DEFINITION 1. *Generally, given a network $G(V,E)$, and an mvs, the community detection problem aims at finding the optimal community assignment $R(Coms, Outs)$ from G, s.t. (1) $Coms \cap Outs = \emptyset$ and (2) $Coms \cup Outs = V$. Herein the optimal assignment refers to closely connected groups of nodes (Coms) and a moderate number of disparate outliers (Outs).*

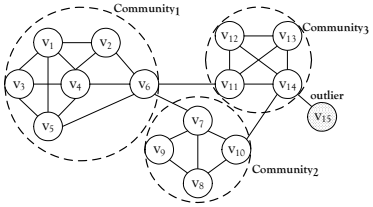


Figure 2: An example of community detection

An intuitive example of the results of community detection is illustrated in Fig. 2. When $mvs = 2$ (a general setting which means only singletons will be eliminated), there are three communities: $Coms = \{\{v_1, v_2, v_3, v_4, v_5, v_6\}, \{v_7, v_8, v_9, v_{10}\}, \{v_{11}, v_{12}, v_{13}, v_{14}\}\}$, and one outlier: $Outs = \{v_{15}\}$.

2.2 Detection Algorithms

Community detection has been studied unremittingly all these years, and a particularly large number of effective approaches have been proposed. In this study we focus on the fundamental problem of non-overlapping community detection, which aims at finding the definite group (community) that each node belongs to in the graph. We categorize the existing approaches according to the formation process of communities, as shown in Fig. 3 which covers most representatives from all popular approaches proposed.

The first kind of approaches starts from the original graph and decomposes the entire graph to local parts gradually, trying to separate out communities from the entire graph.

Division algorithms in **hierarchy clustering** methods, such as Radicchi [23] and Spectral [19], gradually separate the entire network into local parts by the edge clustering coefficient or the eigenvalue of modularity matrix.

Direct partitioning methods separate the entire network into disjoint communities. The *Scalable Community Detection* (SCD) algorithm [22] partitions the network by maximizing the weighted community clustering [21], a recently proposed metric of community. *Maximal k-Mutual-Friends* (M-KMF) [29] algorithm incrementally filters out the connections by the number of mutual friends between nodes to let the communities spontaneously emerge.

Conversely, the second kind of approaches takes a bottom-up manner from local structures to the whole graph, and the communities are formed during this process.

Label propagation methods start from local neighborhood to recognize communities automatically. The *Label Propagation Algorithm* (LPA) [24] adopts an asynchronous update strategy where nodes join in groups under their neighbors' choices. The HANP algorithm [17] based on *Hop Attenuation* and *Node Preference* adopts additional rules to ensure more stable and robust results.

Leadership expansion methods find communities according to local leader groups, since members always gather together around some core nodes with high centralities to form communities. The TopLeaders [12] algorithm gradually associates nodes to the nearest leaders and locally reelects new leaders during each iteration.

Clique percolation methods assume communities are constructed by multiple adjacent cliques. Based on the original approach [20], the *Sequential Clique Percolation* (SCP) [14] algorithm sequentially generates cliques to form connected communities.

The third kind of the approaches maintains a tree, which is a multi-level structure reorganized from the original graph, aiming at finding communities corresponding to the branches of the tree.

Agglomeration algorithms in **hierarchy clustering** methods usually build an explicit hierarchical tree from small clusters to large ones. Based on the *Newman Fast Greedy Algorithm* (NFGA) [18], Claust et al. proposed an agglomeration algorithm CNM [5] which starts from single nodes, maintains the change of modularity, and iteratively generates the optimal level of the hierarchy structure.

Matrix Blocking technique can also be utilized in community detection by constructing a hierarchy tree to order nodes in a network. As a representative, the *Matrix Blocking Dense Subgraph Extract* (MB-DSGE) algorithm [4] reorders the network, and extracts dense subgraphs as communities.

Skeleton clustering methods reveal dense connected clusters based on the skeleton of the original network, which is an efficient way of finding communities. The SCOT+HintClus algorithm [3] detects the hierarchical cluster boundaries of a network to extract the meaningful cluster tree. Inspired by this idea, the *Graph Skeleton Clustering* (gCluSkeleton) algorithm [11] projects the network to its core-connected maximal spanning tree, and then detects the optimized core-connected clusters on it.

3. FRAMEWORK FOR BENCHMARKING

In this section, we present our procedure-oriented framework for benchmarking in community detection, which consists of two fundamental concepts abstracted from existing detection algorithms and a generalized procedure of community detection.

3.1 Fundamental Concepts

Existing algorithms usually solve the community detection problem with various methods based on different assumptions. This

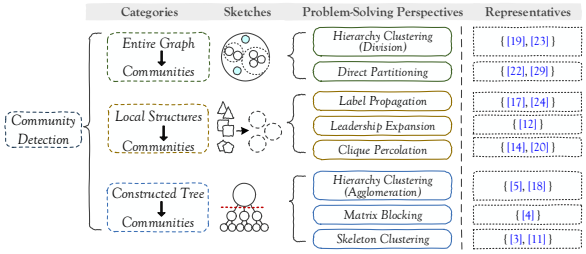


Figure 3: Categories of community detection approaches

makes it difficult to comparatively analyze these algorithms thoroughly. For the sake of a better understanding of the underlying principles of community detection algorithms, we abstract two fundamental concepts, including the **propinquity measure** and the **revelatory structure**, which play critical roles in the community detection task and can be used to distinguish different approaches.

Definition 1. (*Propinquity Measure*). Given a subset M of the elements (such as nodes, relationships or other specific structures) in a graph G, the propinquity measure of M, denoted as $\phi(M)$, is the measurement of the nearness of M by the inner-connections, and is the primary criterion to estimate the priority of the elements when they are transformed to make the communities emerge.

Definition 2. (*Revelatory Structure*). Given a graph G, the revelatory structure corresponding to G, denoted as Π, is an assistant structure derived from G, provides yet another way to organize the massive graph elements and enlightens us on the community structure from the intertwined connections among them.

The two concepts lay the basis for different community detection algorithms. The former determines the tendency of grouping nodes to communities, while the latter records the gradual formation of communities and leads to a more effective detection process especially for approaches of the third category.

It should be noted that the specific definitions of the above concepts could be quite different in various approaches and thus we only give a general definition here. The propinquity measure may be the modularity [5], node centrality [12], etc., and the revelatory structure may appear as the hierarchy tree [4, 18], G^* graph [14] or other particular structures. We will discuss how the two concepts are defined specifically in different algorithms in Sec. 4.

3.2 The Generalized Procedure

We formulate a generalized procedure of community detection in this framework. As illustrated in the "Detection Framework" module in Fig. 1, the procedure consists of three phases, including *initialization*, *transformation* and *construction*, and characterizes the generic workflow of community detection via a series of the key steps. The details of the procedure are shown in Alg. 1.

Phase 1: Initialization

In this phase, the graph elements need to get their initial propinquity values and form a primary community assignment (R^0_{tmp}). As shown in Alg. 1, after the propinquity measure (ϕ) and the revelatory structure (Π) are defined (Line 1), the procedure calculates initial propinquity values via PRECOMPUTE (Line 3) and allocates the primary assignment via FIRSTALLOCATE (Line 4).

Phase 2: Transformation

In this phase, the inner structures and relations underlying the network elements are transformed and clarified iteratively, resulting in a set of intermediate detection results $S_{R_{tmp}}$. We abstract three key steps for each iteration, including SELECT, FLUCTUATE and UPDATE (Line 6–8). First of all, in SELECT, the candidate elements Cad^T are picked out from the graph. After that, in FLUCTUATE, the revelatory structure Π correlated with these elements will

Algorithm 1 GENERICDETECTPROC

Input: $G(V,E)$, mvs and T_{max}
Output: $R(Coms, Outs)$
1: initialize ϕ and Π;
2: $T \leftarrow 0$, $S_{R_{tmp}} \leftarrow \emptyset$, $S_R \leftarrow \emptyset$, $Cad^T \leftarrow \emptyset$;
3: PRECOMPUTE(G, ϕ);
4: $R^T_{tmp} \leftarrow$ FIRSTALLOCATE(G, Π);
5: **while** $T! = T_{max}$&&!STABLE(Π)&&!OPTIMAL(ϕ) **do**
6: $Cad^T \leftarrow$ SELECT(G, Π);
7: $R^T_{tmp} \leftarrow$ FLUCTUATE(Cad^T, Π);
8: UPDATE(INVOLVE$(Cad^T), \phi)$;
9: $S_{R_{tmp}} \leftarrow S_{R_{tmp}} \cup R^T_{tmp}$;
10: T++;
11: **end while**
12: **if** $S_{R_{tmp}}$ has multiple results **then**
13: **for** each $level \in \Pi$ **do**
14: $S_R \leftarrow S_R \cup$ COLLECT$(S_{R_{tmp}})$;
15: **end for**
16: $R \leftarrow$ MULTILEVELDRAFT$(S_R, \phi$ or $\psi)$;
17: **else if** $S_{R_{tmp}}$ has no obvious result **then**
18: $R \leftarrow$ EXTRACT$(\Pi, \phi$ or $\psi)$;
19: **else** R is obtained in the iteration;
20: **end if**
21: $R.Outs \leftarrow R.Outs \cup$ ERASE$(R.Coms, mvs)$;
22: ORDER$(R.Coms)$;
23: **return** R;

be transformed, i.e. being fluctuated to form a more apparent intermediate result (R^T_{tmp}). Following these structural transformations, in UPDATE, the propinquity of other involved elements will be recomputed and the next iteration starts. These three steps are conducted iteratively until the iteration terminates. During the iterations, the latent communities can form in many ways, as the aforementioned categories, i.e. from the whole graph to communities, from local structures to communities or from trees to communities.

Three indicators are usually employed to terminate the iterative procedure (as shown in Line 5): (1) whether the iteration number reaches the fixed threshold T_{max} (which is usually chosen to ensure an approximate convergence of an algorithm); (2) whether the revelatory structure Π is stable (STABLE(Π)); and (3) whether the value of the propinquity measure ϕ is optimal (OPTIMAL(ϕ)). According to specific algorithms, these indicators may be chosen and designed specifically in different approaches.

Phase 3: Construction

In the last phase, the final result R is constructed by refining the current intermediate results $S_{R_{tmp}}$. If $S_{R_{tmp}}$ has multiple choices, COLLECT (Line 14) needs to gather the result at each level of Π. Then MULTILEVELDRAFT (Line 16) weighs the collected results and picks up the best-performing one. Usually, based on Π, EXTRACT (Line 18) is employed to let the communities emerge. In most cases, EXTRACT or MULTILEVELDRAFT may also adopt ϕ as a measure, nevertheless, sometimes another measure ψ can be adopted, e.g. density in [4]. Eventually, the invalid communities are removed by ERASE according to *mvs*, and *R.Coms* is sorted via ORDER by the community size (Line 21–23).

3.3 Highlights

Composed of two fundamental concepts and a generalized procedure, our framework is beneficial for understanding and analyzing the community detection approaches. The two concepts are the basis for solving the problem of community detection, and different implementations may lead to quite different performances, even for the same approach (as illustrated in Sec. 5.1). The procedure is the modularization of the critical steps of this problem, and uncovers the generic detection workflow, making it easy to study various approaches deeply within the identical framework.

Furthermore, the framework provides flexible combinations of the optional steps, making itself adaptable to different kinds of algorithms. Generally, all of the three categories of algorithms discussed in Sec. 2.2 can be mapped to this framework.

The framework is light-weight. It unifies the input and output, handles the overall detection process, and provides necessary interfaces. Algorithms can be integrated easily by defining the factors in Sec. 3.1 and implementing the functions in Sec. 3.2. Both the framework and algorithms are developed using standard C++.

4. IMPLEMENTATION UPON FRAMEWORK

The present section recaps ten of the state-of-the-art algorithms for community detection and describes how they work under this framework. Fig. 4 shows the overall procedure mapping of these algorithms to the framework. Based on the specific perspectives, we consider the following representatives:

- Agglomeration algorithm CNM [5], and division algorithms Radicchi [23] and Spectral [19] (hierarchy clustering, Sec. 4.1).
- M-KMF [29] (direct partitioning, Sec. 4.2).
- LPA [24] and HANP [17] (label propagation, Sec. 4.3).
- TopLeaders [12] (leadership expansion, Sec. 4.4).
- SCP [14] (clique percolation, Sec. 4.5).
- MB-DSGE [4] (matrix blocking, Sec. 4.6).
- gCluSkeleton [11] (skeleton clustering, Sec. 4.7).

4.1 Hierarchy Clustering

The hierarchy clustering algorithms (CNM [5], Radicchi [23], Spectral [19], etc.) form communities in a multi-level structure progressively on the basis of the original graph. This process falls into two types, i.e. agglomeration or division, depending on their construction order of the hierarchy structure.

The revelatory structure. Agglomeration algorithms usually maintain an explicit *hierarchy tree* as the revelatory structure Π, in which the leaves denote nodes of the graph and the branches combine nodes or groups at different levels. Actually, without constructing a hierarchy tree, most division algorithms, such as Radicchi [23] and Spectral [19], start from the entire graph and split out the communities gradually. For the lack of space here, we only introduce the procedure mapping of the classical agglomeration algorithm CNM.

The propinquity measure. CNM employs *modularity* [5] as the propinquity measure ϕ to make a greedy choice, trying to optimize the global modularity of the final community partition: $\phi(R) = \sum_{i=1}^{cn} \left[\frac{I_i}{m} - \left(\frac{2I_i+O_i}{2m}\right)\right]$, where I_i indicates the total number of internal relationships within the community C_i, O_i the number of outgoing relationships between nodes in C_i and any node outside. For CNM, any node would be grouped into a community during the transformation, and thus the *cn* communities cover all nodes in the graph.

The initialization phase. At the beginning, PRECOMPUTE obtains an initial value of $\phi(G)$ and FIRSTALLOCATE takes each node as a single tree (tiny mono-community with only one member) to form a primary forest (R^0_{tmp}).

The transformation phase. In the iteration T, SELECT chooses two candidate trees (i.e. current communities) Cad^T, whose combination may lead to the maximum increase of $\phi(R^{T-1}_{tmp})$. Then, FLUCTUATE combines them to form a new tree (a new community of R^T_{tmp}). Afterwards UPDATE recalculates the new value of $\phi(R^T_{tmp})$, and then the algorithm turns to the next iteration.

The construction phase. In some early methods, COLLECT needs to gather the result in each level of Π, and then MULTILEVELDRAFT selects the best one (with the maximum $\phi(R)$) [18]. The improvement in CNM lies in SELECT and UPDATE. It breaks the iteration once the latest combination cannot increase the modularity any more ($\Delta\phi < 0$ and OPTIMAL$(\phi(R))$ is true). Consequently, the

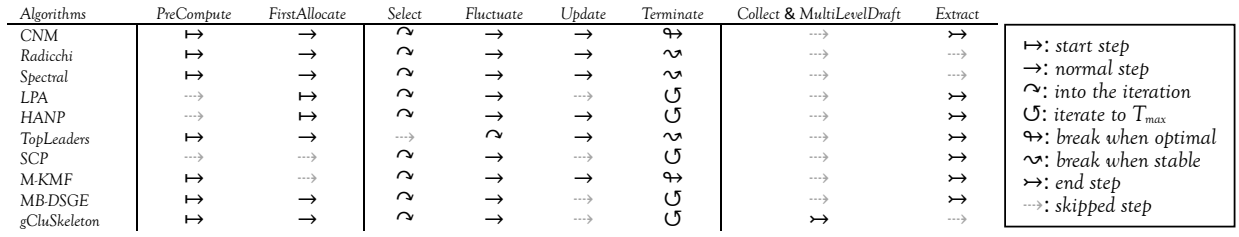


Figure 4: The overall procedure mapping of ten studied algorithms

current forest makes up the result. Without a direct output during the iteration, CNM needs to take all the leaves in each individual tree as the members of a same community via EXTRACT.

4.2 Direct Partitioning

Partitioning a graph directly is an intuitive way to get the closely connected communities. We take M-KMF [29] as a representative and present its mapping to our framework as follows.

The revelatory structure. In M-KMF the *k-mutual-friends subgraphs* are taken as Π and the most appropriate ones get separated out as communities. Each node $v \in \Pi$ has the degree $d(v) \geq k$ and each relationship belongs to at least k *triangles*.

The propinquity measure. In M-KMF, the $\phi(r)$ of each relationship r is defined as the number of triangles $TR(r)$ containing r.

The initialization phase. Without an initial assignment of communities, M-KMF merely executes PRECOMPUTE to initialize the $\phi(r)$ of each relationship. Before that, it also handles a degree filter which pulls nodes with $d(\cdot) < k+1$ out as outliers.

The transformation phase. In each iteration T, SELECT picks out the relationships (Cad^T) for whom $\phi(r) < k$, and FLUCTUATE removes all these candidates from the current graph. Once a batch of relationships are erased, new disconnected parts (R^T_{tmp}) may arise. Hence, they are closer to the k-mutual-friends subgraphs we look for. Then, UPDATE copes with each relationship involved by these deletions and recomputes $\phi(r)$ for the next iteration.

The construction phase. The iteration ends precisely when there is no relationship to be marked. At this moment, each living relationship belongs to k triangles, so that the mutual friends between each pair of connected nodes is maximum (OPTIMIZE($\phi(r)$) is true). To fetch the final communities, EXTRACT needs to traverse the current Π to get the connected components.

4.3 Label Propagation

Label propagation algorithms (LPA [24], HANP [17], etc.) identify communities via different labels spreading among neighbors.

The revelatory structure. Π herein is defined as the *label distribution* where different labels stand for different communities.

The propinquity measure. LPA has no special propinquity measure, whereas HANP adopts *hop score* and *preference* of each node v_i to improve its robustness: $\phi(v_i) = S(i) \cdot P(i)^{\omega}$ where ω is a regulatory factor. Please note that the preference $P(i)$ can be $d(i)$, and the hop score $S(i) = \max_{j \in N_L(i)} S(j) - \sigma$ where $N_L(i)$ is the neighbors having the same label with i and σ is the attenuation factor.

The initialization phase. In FIRSTALLOCATE, nodes are assigned unique labels indicating different mono-communities (R^0_{tmp}).

The transformation phase. In the iteration T, SELECT generates a random permutation of nodes (Cad^T). For LPA, in FLUCTUATE, each node changes its label to the most frequent one of the labels of its neighbors. While for HANP, the process is extended by introducing the hop score and the node preference to the label frequency. The new label distribution makes up the new temporary result R^T_{tmp}. Without ϕ, LPA needs no UPDATE while HANP needs UPDATE to recompute the $S(i)$ of $\phi(v_i)$ w.r.t. σ, which controls how far the particular label of the newly fluctuated node can spread.

The construction phase. Empirically, label propagation algorithms limit the number of iterations to a maximum value T_{max} to ensure that the label distribution achieves a steady status approximately. At last, EXTRACT will classify nodes with same labels to same communities.

4.4 Leadership Expansion

Leadership expansion methods (e.g., TopLeaders [12]) regard a community as a set of followers congregating around a potential leader, and form communities by identifying promising leaders and then iteratively assembling followers.

The revelatory structure. All leaders try to expand their own *leader groups* (i.e. Π) respectively to form the final communities.

The propinquity measure. The TopLeaders algorithm adopts *degree centrality* $\phi(v_i) = d(i)$ of a node v_i to measure the leadership.

The initialization phase. Specifically, PRECOMPUTE first computes the global degree centrality of each node in the graph. Then, FIRSTALLOCATE elects the ld most pivotal nodes as the leaders by the FNiC strategy (Few Neighbors in Common, which assures neighborhoods of any two leaders have an intersection less than λ_{max}) and forms initial states of the groups (R^0_{tmp}).

The transformation phase. In each iteration T, TopLeaders skips SELECT and executes FLUCTUATE directly. In FLUCTUATE, each node explores its neighbors within l hops using the BFS strategy and counts the common neighbors with each leader. Once the number of common neighbors with any leader exceeds the threshold λ_{min} before the BFS reaches the maximum depth, dp, the node joins the corresponding group around that leader. Then, the leader group gets enlarged (R^T_{tmp}). Please notice that any node who has multiple choices and cannot make this decision until l reaches the upper bound will be regarded as a hub, i.e. a special outlier. Besides, any node whose common neighbors cannot reach λ_{min} anyhow will be regarded as an outlier. In consequence of the expansion, UPDATE needs to recompute $\phi(v_i)$ locally and tries to reelect new leaders.

The construction phase. The transformation process repeats until no new leader comes to power any more (STABLE(Π) is true). Once all the leaders win a second term, each current leader group can be obtained via EXTRACT as a final community.

4.5 Clique Percolation

In clique percolation algorithms, a *clique* is an atomic element in the graph. As a representative, SCP [14] detects communities sequentially based on the idea of clique percolation. Unlike other algorithms, SCP does not define an explicit propinquity measure.

The revelatory structure. SCP maintains the G^* *graph* as the revelatory structure Π, in which the κ-1-cliques are taken as nodes. If two κ-1-cliques belong to a same κ-clique, they are connected in the G^* graph. Generally, SCP assumes $\kappa = 3$ or 4. If $\kappa = 3$, for the endpoints of a relationship, the κ-cliques can be directly obtained by their common neighbors. If $\kappa = 4$, each pair of the connected common neighbors can make up a 4-clique together with the two endpoints. In the experiments later, we will show the effect of κ.

The transformation phase. SCP goes directly into the transformation. In each iteration T, SELECT first gets the common neigh-

bors of the endpoints of a relationship, and forms the corresponding κ-cliques (Cad^T). Then, in FLUCTUATE, all inner κ-1-cliques are inserted into the G^* graph (R^T_{tmp}). All the κ-1-cliques which belong to a same upper clique will be linked in this graph.

The construction phase. The iteration is nothing but a sequential traversal of the relationships and ends after dealing with all κ-cliques ($T_{max} = m$). At last, EXTRACT hunts for all connected components in the G^* graph and takes them as communities.

4.6 Matrix Blocking

Matrix blocking technique (e.g., MB-DSGE [4]) finds the dense subgraphs, i.e. the communities, based on the rearrangement of G.

The revelatory structure. The *hierarchy tree* is adopted as Π.

The propinquity measure. MB-DSGE utilizes the *cosine similarity* to measure the propinquity of two nodes: $\phi(v_i, v_j) = \frac{<A(:,i),A(:,j)>}{|A(:,i)||A(:,j)|}$ where A is the adjacent matrix of G.

The initialization phase. PRECOMPUTE first computes $\phi(v_i, v_j)$ of each pair of nodes (not only those who have connections). Then, triples $(i, j, \phi(v_i, v_j))$ are sorted in descending order according to the value of ϕ to form a queue CQ. In FIRSTALLOCATE, trees representing single nodes form the original forest (R^0_{tmp}).

The transformation phase. In the iteration T, SELECT pops a triple $(i, j, \phi(v_i, v_j))$ from CQ with the maximal cosine similarity and finds the trees corresponding to node i and j (Cad^T). Then, FLUCTUATE combines the two trees to form a new branch (R^T_{tmp}) in Π if they have no common ancestor. Since CQ has been obtained before, UPDATE is unnecessary in MB-DSGE.

The construction phase. The entire iteration ends once the whole tree is built ($T_{max} = n - 1$). Since an obvious result cannot be obtained now, MB-DSGE constructs the final communities using another measure *density*. EXTRACT firstly counts the wrap-up nodes $V(b)$ and relationships $E(b)$ for each branch b of Π in a bottom-up way. Then, it traverses Π in a top-down way and computes the density ($\psi(b) = \frac{2|E(b)|}{|V(b)|(|V(b)|-1)}$) of each branch. Once the density of b is higher than the predefined threshold ρ_{min}, b is pruned out and all the leaves within it turn out to be the tightly interconnected members of a community.

4.7 Skeleton Clustering

Skeleton clustering algorithms (e.g., gCluSkeleton [11]) try to find closely connected communities based on the skeleton of the original graph G to make the detection process more efficient.

The revelatory structure. The gCluSkeleton algorithm makes the skeleton-based clustering just on the core-connected maximal spanning tree (CCMST) of G. Therefore, the CCMST and the *hierarchy tree* together make up the revelatory structure Π.

The propinquity measure. The gCluSkeleton algorithm adopts the *structural similarity* (denoted as ε) to measure the propinquity of the endpoints of a relationship r. The propinquity measure $\phi(r_{ij}) = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| \cdot |N(j)|}}$ where N is the self-contained neighborhood of a node. The propinquity measure ϕ here owns a suite of derivatives: (1) *core-similarity* (*CS*), (2) *reachability-similarity* (*RS*), (3) *core-connectivity-similarity* (*CCS*), and (4) *attachability-similarity* (*AS*). For node i, $CS(i)$ is the maximum structural similarity $\hat{\varepsilon}$. It means node i should have at least μ adjacent relationships whose structural similarities are greater than $\hat{\varepsilon}$. $RS(r_{ij}) = \min\{CS(i), \phi(r_{ij})\}$ stands for the reachability from node j to i. $CCS(r_{ij}) = \min\{RS(i, j), RS(j, i)\}$, ensures the bidirectional reachability of the two nodes. $AS(i)$ is the maximum $RS(j, i)$ obtained with all neighbors of node i, and the corresponding neighbor j is called an attractor of i.

The initialization phase. In PRECOMPUTE, the *CCS* of all the relationships is computed, and meanwhile the triples of $(i, j, AS(i))$ are initialized and stored in a priority queue AIQ. Taking *CCS* as weight, FIRSTALLOCATE produces the CCMST of G and stores all unique *CCS* in CCMST in the ε-queue in descending order.

The transformation phase. In the iteration T, SELECT takes the first ε out of the ε-queue and filters all relationships in CCMST with $CCS < \varepsilon$ to reveal the current core-connected clusters (Cad^T). When necessary ($\mu > 2$), SELECT fetches attractors from AIQ to enlarge the clusters. FLUCTUATE then takes these clusters as branches in the hierarchy tree and the clusters produced in the previous iteration as subbranches of them. Along with the decreasing of ε, each previous branch definitely belongs to a current one.

The construction phase. Without a wise indicator of the termination state, the iteration goes on until all the branches form a whole tree ($T_{max} = n - 1$, at most), when the ε-queue is empty. Now, COLLECT brings the homologous clusters w.r.t. the branches into buckets (S_R) with different ε values (different levels in Π). MULTILEVELDRAFT then picks up the best result R using an extended modularity function $\psi(R)$.

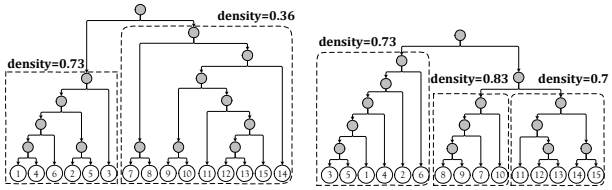
5. DIAGNOSES AND IMPROVEMENTS

With high abstraction and modularization of the fundamental factors and key steps of the general detection approaches, the proposed framework provides not only a testbed to compare various approaches with a fair benchmark, but also a microscope for us to study the strength and weakness of each algorithm. It allows us to conduct diagnoses for algorithms which do not perform well, and further makes targeted prescriptions for improvement. Due to the space limitation, here we discuss directions for diagnosing and improving the algorithms with two examples.

5.1 Diagnosing Key Factors

As analyzed earlier, the propinquity measure ϕ plays a critical role in many approaches to community detection. Different approaches usually adopt various implementations of ϕ, but not all of them are the best choices. Thus one promising direction for improvement is to look again at the propinquity measures inside existing approaches and seek more appropriate implementations.

We take MB-DSGE as an example. In Fig. 5(a) we illustrate how MB-DSGE works on the given network shown in Fig. 2. According to the current propinquity measure ϕ defined using cosine similarity, v_9 is closer to v_{10}, with whom it has two common neighbors, than v_7 and v_8. Thus, in MB-DSGE, v_9 would be merged with v_{10} first. However, without even a relationship, they are factually not so close. On the contrary, v_9 is closer to the directly-connected v_7 and v_8. Consequently, it may result in sparse communities with low density. Another possible drawback of the current definition lies in that the individual outliers cannot be identified clearly since they usually have few neighbors and may be prematurely merged (e.g. v_{15} in Fig. 5(a)).



(a) MB-DSGE (b) MB-DSGE*

Figure 5: Comparison of the hierarchy tree and the results of MB-DSGE and MB-DSGE* ($\rho_{min} = 0.35$)

We can improve MB-DSGE by replacing ϕ, leading to an improved algorithm MB-DSGE*. Since the propinquity measure ϕ in MB-DSGE does not take the direct connections between two nodes into consideration, we can replace it with the corresponding ϕ in gCluSkeleton, i.e. the structure similarity, to correct this omission. Note that for MB-DSGE*, ϕ needs to be computed for each pair of

nodes, while for gCluSkeleton it only needs to be computed for the endpoints of each relationship.

The execution processes of MB-DSGE and MB-DSGE* on the network of Fig. 2 are visualized in Fig. 5 with the built hierarchy tree and the communities denoted with the dotted boxes. Compared with Fig. 5(a), the new hierarchy tree in Fig. 5(b) seems more orderly since MB-DSGE* generates a combination order $\{3,5,1,4,2,6\}$ in accordance with the nearness of nodes in the subgroup. MB-DSGE* produces three dense subgraphs while MB-DSGE cannot differentiate the two groups in the right part of the graph. Consequently, the rationality of the result gets heightened a lot.

5.2 Diagnosing Key Steps

In Sec. 3.2, we abstract the main phases and key steps of general approaches to community detection. As presented in Sec. 4, algorithms can be implemented within our framework via step mapping. This enables us to examine the performance of each single step and analyze their influence on the whole algorithm in depth. Upon that we can propose targeted improvements by adding a beneficial step, removing an unnecessary one, or modifying an inefficient one.

We illustrate this with the LPA algorithm as example. In original LPA, no propinquity measure is defined. Thus there do not exist important steps of evaluating and updating the propinquity of elements in the network. In this way the varying importance of nodes and relationships is neglected.

This inspires a direction to improve LPA. The nodes with different confidences should be considered differently during the label propagation process. Thus we introduce the *activeness* of nodes, which shows how many times a node changes its label in the whole iteration. Intuitively, there is a negative correlation between the confidence of a node and its activeness, and thus we define the propinquity measure as $\phi(v_i) = 1/Activeness(i)^{\omega}$. In the transformation phase, we modify the FLUCTUATE step, in which nodes also choose their labels according to the neighbors' confidence, and add a critical step UPDATE to recompute the propinquity of nodes.

As demonstrated in the experiments later in Sec. 6.10, the specific diagnoses achieve great improvements over original algorithms. It should be noted that in this study we only take two examples for illustration due to the limit of space. More diagnoses may be conducted for other existing approaches based on our framework. For instance, instead of a trivial MULTILEVELDRAFT, a smart EXTRACT step may be preferred in gCluSkeleton. It may also make the iteration in TopLeaders more efficient by adding an extra SELECT for choosing the candidates.

6. BENCHMARKING EVALUATION

In this section, we conduct in-depth evaluations for the community detection algorithms within our framework using the proposed benchmark, which covers the efficiency, accuracy, effectiveness, density sensitivity, mixture sensitivity, outliers, community distribution and diagnosis effects. We introduce the datasets and parameter configurations in the benchmark at first, and then report our thorough evaluation methodology and results. We summarize our findings and rate the algorithms intuitively at last. All experiments are conducted on a computer running Windows Server 2008 with an Intel Xeon 2.0 GHz CPU and 256 GB RAM.

6.1 Datasets

Towards a better and thorough evaluation of the existing approaches, in this study we adopt three categories of datasets.

In the first category, we consider two widely-used small-scale real-world networks, including **Strike** and **Football** [12], for which we can obtain the authentic communities. The statistics are shown in Table 1, where *cn* is the number of real communities.

Table 1: Small-scale real-world networks with ground-truth

Name	n	m	cn	Supp.
Strike	24	38	3	/
Football	180	788	11	61 outliers, 4 hubs

We also employ seven large-scale real-world networks without ground-truth from the Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/), ranging from co-author networks (**CA-HepPh**, **DBLP**), co-purchasing networks (**Amazon**), communication networks (**Email-Enron**) to friendship networks (**BrightKit**, **Gowalla** and **YouTube**). We report the statistics in Table 2, where cc_{avg} is the average clustering coefficient, and *diam* the network diameter. We cannot obtain the real communities of these networks and thus we adopt a widely-recognized metrics to evaluate the performance in terms of the *quality of clusters* instead of the *accuracy*.

Table 2: Large-scale real-world networks

Name	n	m	cc_{avg}	$diam$
CA-HepPh	12,008	118,505	0.6115	13
Email-Enron	36,691	183,830	0.4970	11
BrightKit	58,228	214,078	0.1723	16
Gowalla	196,591	950,327	0.2367	14
DBLP	317,080	1,049,866	0.6324	21
Amazon	334,863	925,872	0.3967	44
YouTube	1,134,890	2,987,624	0.0808	20

Furthermore, we use the Lancichinetti-Fortunato-Radicchi (LFR) networks [15] with ground-truth. Tens of synthetic datasets are produced for in-depth evaluations, and we only list some representatives in Table 3. Here d is the average degree of nodes, d_{max} the maximum node degree, c_{min} the minimum community size, and c_{max} the maximum size. In LFR, another critical parameter is the mixing parameter u (0.2 by default [11]) which indicates the proportion of relationships a node shares with other communities.

Table 3: Synthetic networks with ground-truth

Name	n	m	d	d_{max}	c_{min}	c_{max}
S_10K	10,000	50,302	10	50	20	100
S_100K	100,000	504,371	10	50	50	100
S_200K	200,000	953,230	10	100	60	200
S_500K	500,000	2,938,555	10	250	100	500

6.2 Configurations

In our benchmarking evaluation, all requisite parameters of the studied algorithms are set as the recommended values in their original papers, as shown in Table 4.

In particular, for Radicchi, we choose the weak option, i.e. the community definition Ω =weak (see in Sec. 6.5), to avoid excessive outliers. For TopLeaders, the number of leaders ld needs to be provided manually or in advance by other means. For MB-DSGE, although the prevalent value of ρ_{min} is 0.1, we may need to run the algorithm repeatedly for better results (e.g., 0.5 on Football), since a slightly inappropriate ρ_{min} may lead to awful results as the scale of the network increases. More details about the effects of the parameters are presented in the following evaluations.

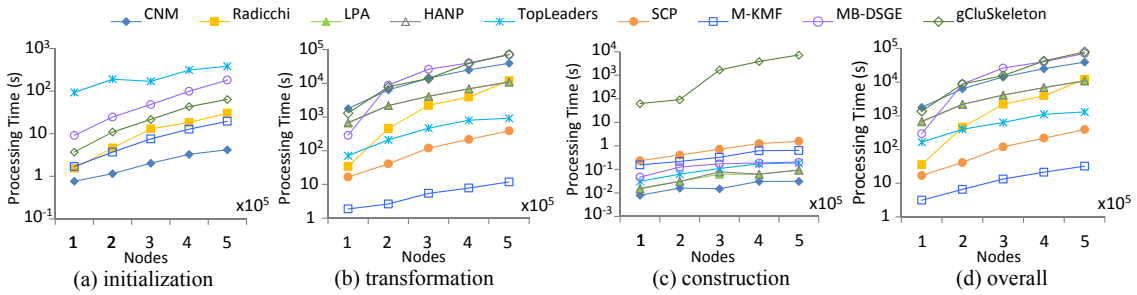
Table 4: Parameter setting

Algorithm	Value	Algorithm	Value
Radicchi	Ω=weak	TopLeaders	λ_{max}=5, λ_{min}=0, dp=2
LPA	T_{max}=6	HANP	T_{max}=20, ω=0.1, σ=0.05
SCP	$\kappa \in \{3, 4\}$	M-KMF	$k \in \{1, 2, 3\}$
MB-DSGE	ρ_{min}=0.1	gCluSkeleton	$\mu \in \{2, 3\}$

6.3 Efficiency

In this subsection, we discuss the efficiency of the concerned algorithms from both theoretical and experimental aspects.

Theoretical analyses. The time complexities of the ten algorithms in each phase are illustrated in Table 5. Most social networks are sparse, and we usually have $O(m) \approx O(n)$. The community structures become non-significant when m tends to n^2 [8].



(a) initialization (b) transformation (c) construction (d) overall

Figure 6: Time costs with node scale under different phases in the framework w.r.t. various algorithms

Theoretically, many classical hierarchy clustering algorithms have high time overheads, such as Radicchi, Spectral and NFGA. However, CNM can greatly reduce the time complexity since it needs no MULTILEVELDRAFT after the transformation. Obviously, label propagation algorithms traverse all nodes and their neighbors for T_{max} times while TopLeaders executes this process for an uncertain time T. Moreover, TopLeaders has a tough work in FIRSTALLOCATE to find the original leaders. The time cost for SCP depends on the quantity of non-repeating κ-1-cliques (N_c) in the network. For M-KMF, the involved elements are mostly relative to the average degree d_{avg} of nodes in each iteration, and the worst case is that all relationships are removed. MB-DSGE spends $O(m \log m)$ time in PRECOMPUTE, and $O(n \log n)$ in tree building. For gCluSkeleton, N_ε is the size of ε-queue (i.e. T_{max}), which equals to $n-1$ at most.

Table 5: Overview of time complexity

Algorithm	Initialization	Transformation	Construction
CNM	$O(m+n)$	$O(m\log^2 n)$	$O(n)$
Radicchi	$O(m+n)$	$O(m^4/n^2)$	—
Spectral	$O(n^2)$	$O(n^2 \log n)$	—
LPA & HANP	$O(n)$	$O(T_{max}(m+n))$	$O(n)$
TopLeaders	$O(n\log n + ld^2)$	$O(T(m+n))$	$O(n)$
SCP	—	$O(N_c)$	$O(m+n)$
M-KMF	$O(m+n)$	$O(d_{avg}m)$	$O(m+n)$
MB-DSGE	$O(m\log m)$	$O(n\log n)$	$O(n)$
gCluSkeleton	$O(m+n\log n)$	$O(m\log n)$	$O(N_\varepsilon(m+n))$

Experiments on synthetic networks. We evaluate the efficiency and scalability of these algorithms on synthetic networks, whose scales vary from 100K nodes to 500K nodes (Fig. 6(a) – (d)). We find Spectral even cannot handle S_100K due to an enormous consumption of 40 GB of RAM for a single-pass iteration, which is an unbearable space overhead in practice. Thus we only evaluate other approaches according to the three phases of the workflow.

Fig. 6(a) shows the initialization time of six algorithms varying with the node scale (with logarithmic coordinate for Y axis). The time costs of LPA and HANP are much lower than others and thus omitted for brevity, and SCP does not have an initialization phase. In accordance with the above analyses, TopLeaders spends the most time in PRECOMPUTE, followed by MB-DSGE and gCluSkeleton, both of which need to compute and sort values of ϕ in this step.

The time costs of the transformation phase of each algorithm are shown in Fig. 6(b). M-KMF outperforms others significantly. For SCP, the iteration performs slightly faster by utilizing the OpenMP API (http://openmp.org) to optimize the process in FLUCTUATION. TopLeaders can hardly get stabilized on large-scale datasets, and thus we relax the proportion of the two-term leaders to 95% approximately. Consequently, this tiny modification increases the efficiency significantly in our experiments with little influence on performance. CNM, MB-DSGE and gCluSkeleton cost much more time in this phase than others. For CNM, the iteration number is supposed to be $O(\log(n))$, but actually, it nearly reaches $O(n)$ (e.g., the number of iteration is 99,404 on S_100K which contains 100,000 nodes). Although the iteration of gCluSkeleton is similar to that of Kruskal algorithm ($O(m \log n)$), the high time cost may be attributed to the pairwise association of clusters in FLUCTUATE.

In the construction phase, except gCluSkeleton, most algorithms complete this process within negligible time, as shown in Fig. 6(c), since EXTRACT usually only traverses Π to get the final results. The gCluSkeleton costs much time due to the existence of the time-consuming COLLECT and MULTILEVELDRAFT. The overall time costs of these algorithms are shown in Fig. 6(d), almost the same as those in the transformation phase.

Experiments on real-world networks. We also conduct experiments on the efficiency of the algorithms on real-world networks. The overall time costs on datasets CA-HepPh, Email-Enron, Amazon and DBLP are presented in Fig. 7. In general, the ranking of the processing time is consistent with those on synthetic networks, and thus we do not discuss them in detail for brevity.

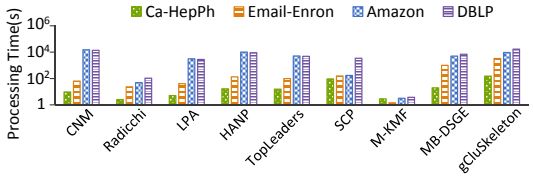


Figure 7: Time overheads on real-world networks

Influence of framework implementation. To evaluate the influence on efficiency of the framework standardization of the original algorithms, we compare the overall processing time of them in both their native implementation and framework implementation on S_500K. The result of the comparison is shown in Fig. 8. Since our re-implementation based on the framework is just an encapsulation of original algorithms with the generalized detection procedure, it has almost no effect on the efficiency of the algorithms.

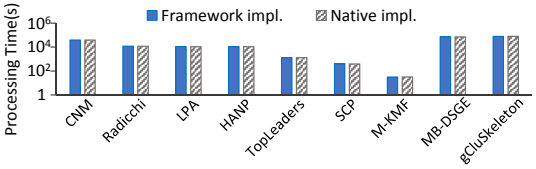


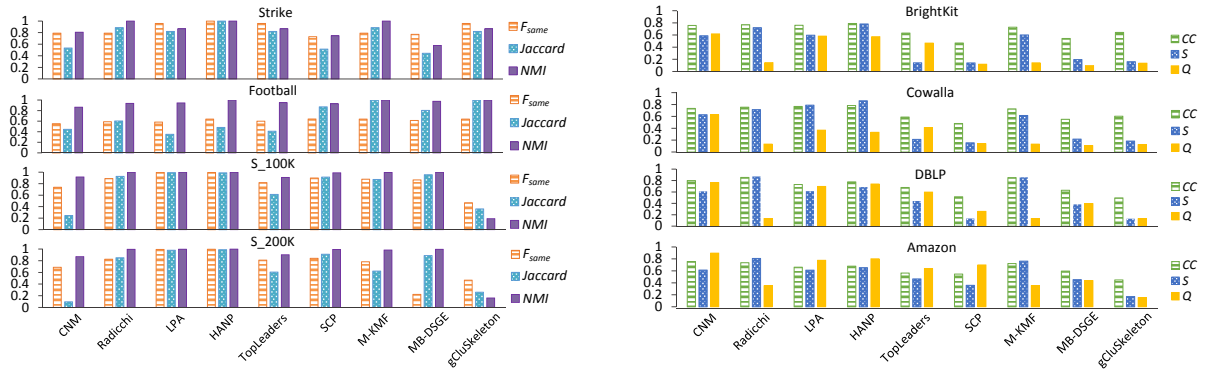
Figure 8: Comparison on the efficiency of framework implementation vs. native implementation

Remarks. Considering the overall time overheads, we find that M-KMF performs more efficiently than others. Generally, these algorithms can be ordered w.r.t. the efficiency as *M-KMF* > *SCP* > *TopLeaders* > *Radicchi* > (*LPA, HANP*) > (*CNM, MB-DSGE, gCluSkeleton*). The approaches themselves are efficient under the time complexity $O(n \log n)$, but the transformation is the most time-consuming phase.

6.4 Accuracy

In this subsection, we evaluate the accuracy of an algorithm on datasets with ground-truth by examining to what extent the communities delivered by an algorithm are consistent with the real ones.

Metrics. *Cross Common Fraction* (F_{same}) compares each pair of communities, in which one comes from the detected result and the



(a) accuracy (b) effectiveness

Figure 9: Accuracy and effectiveness of various algorithms on six metrics (accuracy: Cross Common Fraction (F_{same}), Jaccard Index (*Jaccard*), Normalized Mutual Information (*NMI*); effectiveness: Cluster Coefficient (*CC*), Strength (*S*), modularity (*Q*))

other comes from the real one, to find the maximal shared parts. Formally, it is defined as

$$F_{same} = \frac{1}{2}\sum_{i=1}^{cn}\max_j |C_i \cap C'_j| + \frac{1}{2}\sum_{j=1}^{cn'}\max_i |C_i \cap C'_j|, \quad (1)$$

where cn and cn' are the quantities of detected and real communities respectively, and C_i and C'_j are the i-th detected community and j-th real community respectively.

Jaccard Index [10] is a widely used similarity measure. It is more sensitive for it can overcome the little variance of the cross common fraction when nodes from several different communities in one result join together as a single community in another result [24]. It can be defined as $\frac{P_s}{P_s+P_{s1}+P_{s2}}$ where P_s stands for the number of node pairs which are respectively classified into the same community in both results, P_{s1} stands for the number of node pairs appearing in the same community in the algorithm-produced results, but in different communities in the truth, and P_{s2} vice versa.

Normalized Mutual Information (*NMI*) [6] is a popular criterion for evaluating the accuracy of community detection based on the information theory. The score of NMI stands for the agreement of two results and it is defined as

$$NMI = \frac{-2\sum_{i,j} N_{ij}\log\frac{N_{ij}N_t}{N_{i.}N_{.j}}}{\sum_i N_{i.}\log\frac{N_{i.}}{N_t} + \sum_j N_{.j}\log\frac{N_{.j}}{N_t}}, \quad (2)$$

where N is the confusion matrix whose element N_{ij} is the number of the shared members between a detected community C_i and a real one C'_j. $N_{i.}$ and $N_{.j}$ are the sum over row i and column j respectively, and $N_t = \sum_i\sum_j N_{ij}$.

Remarks. The scores of the above metrics are shown in Fig. 9(a). Based on the accuracy evaluation, we can find that SCP, M-KMF, MB-DSGE and gCluSkeleton perform outstandingly on networks containing many outliers, such as Football. The results of these algorithms show high stability, except MB-DSGE which may be easily influenced by the scale of nodes. Moreover, it is obvious that LPA and HANP produce eminently believable and accurate results on networks without any outliers, such as synthetic networks.

6.5 Effectiveness

Most large-scale real-world networks have no ground-truth. In this situation, we evaluate the effectiveness of algorithms by examining the quality of detected communities via various metrics.

Metrics. *Clustering Coefficient* (*CC*) focuses on the tendency of community members to cluster together. Thus, high clustering coefficient means high probability that the connections inside the detected communities are dense. The global clustering coefficient of an undirected network can be computed as

$$CC = \frac{1}{cn}\sum_{i=1}^{cn}\left(\frac{1}{|C_i|}\sum_{v\in C_i}\frac{2|\{e_{ts}: v_t, v_s \in N(v)\bigcap C_i, e_{ts}\in E\}|}{d(v)\,(d(v)-1)}\right), \quad (3)$$

where $N(v)$ consists of the neighbors of node v, $d(v)$ is the degree of v, cn is the community quantity, and C_i is the i-th community.

Strength (*S*) measures the intensity of detected communities. Evidence shows that the result usually seems valid if most members are in the same community with their neighbors. We inherit Radicchi's community definitions [23] to get the strength score of a result. Suppose that $d(v)^{in}$ and $d(v)^{out}$ stand for degrees inside and outside the community C containing node v, respectively. If $d(v)^{in} > d(v)^{out}$ for $\forall v \in C$, C is a strong community; if $\sum_v d(v)^{in} > \sum_v d(v)^{out}$, C is a weak community; otherwise C is invalid. To make the global evaluation, we define the strength as: $S = \frac{1}{cn}\sum_{i=1}^{cn} score(C_i)$. In this work, we let $score(C_i) = 1$ for a strong community; 0.5 for a weak one; and 0 for an invalid one.

Modularity (*Q*) is the most widespread quality function for community detection. It compares the result with a randomized one to indicate how reasonably the nodes are assigned into different groups. The definition of this metric has been given in Sec. 4.1.

Remarks. As illustrated in Fig. 9(b), almost all algorithms show highly-consistent effectiveness on different datasets. Most algorithms get satisfactory *CC* so that communities produced by these algorithms have high internal aggregation. Specifically, with the similar ϕ defined based on the number of triangles, Radicchi and M-KMF perform very well w.r.t. *S*, showing strong preference for intra-group links than inter-group ones. However, their scores of modularity are much worse than those of others. Differently, CNM gets the best *Q* among all the algorithms since it takes this metric as ϕ and achieves the optimal value at the end of transformation. It is worth mentioning that without prior knowledge of the leader number, TopLeaders does not perform well any more. In general, communities detected by HANP, LPA and CNM have higher qualities than those produced by other algorithms.

6.6 Density Sensitivity

We are also interested in the sensitivity of algorithms when they are applied to datasets with various network densities. In this experiment we generated synthetic networks on the basis of S_10K by gradually increasing its density by modifying the average degree d (from 10 to 70, $\Delta d = 10$, note that $|E| = \eta|V|$ and $d \approx 2\eta$).

The variations of the overall time costs are shown in Fig. 10. Except SCP, CNM, LPA and HANP, the growing tendency of the rest algorithms are in line with that demonstrated in Fig. 6(d). We also studied the variations of effectiveness, accuracy as well as community quantity of the concerned algorithms on datasets with different densities, and present the results in Fig. 11. For space limitation, here we take the *CC* and F_{same} as representatives of effectiveness and accuracy, respectively, and similar tendencies are observed on

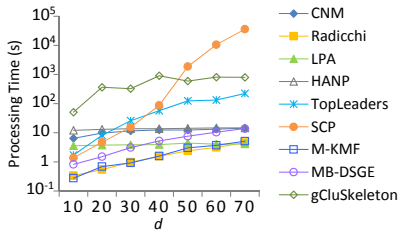


Figure 10: Overall time costs of varying d

other metrics. With the increase of density, most algorithms except SCP produce communities with higher CC values as expected. From the perspective of accuracy evaluated with F_{same}, Radicchi, LPA and HANP not only outperform others significantly, but also show more stable performance with less sensitivity on the density. On the contrary, the performances of SCP, M-KMF and MB-DSGE decrease obviously when networks become denser. Furthermore, we find the quantities of detected communities decrease consistently with the increasing density. Since $k=1$ here, M-KMF is more likely to form large connected k-mutual-subgraphs so that the total number of communities drops rapidly.

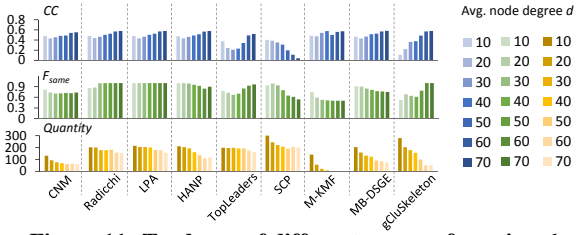


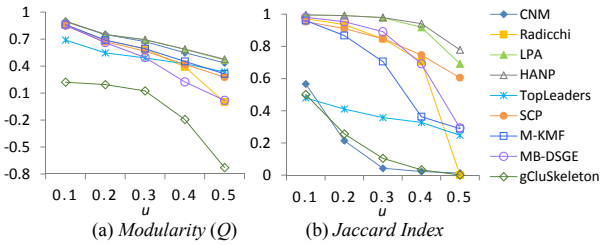
Figure 11: Tendency of different scores of varying d

Remarks. The increase of density of networks usually leads to higher clustering coefficients, and for most algorithms it produces fewer detected communities. Moreover, since there are more links among nodes so that there are fewer outliers produced by these algorithms as the density increases. In general, Radicchi, LPA and HANP are superior to others on both the overall performance and stability. The growth in the density of a network has the strongest impact on both the efficiency and the performance of SCP.

6.7 Mixture Sensitivity

Mixing (or overlapping) structures are common in real-world social networks, which make it more difficult to differentiate communities from the entire network. In this experiment, we study the performance and sensitivity of these algorithms on datasets with different mixture degrees.

For synthetic networks generated by LFR, the mixture degree depends on the mixing parameter u, as we discussed in Sec. 6.1. A small value of u leads to large diversities among different communities while a high value of u may produce overlapping communities. We generated synthetic datasets using LFR with 100K nodes and different settings of u from 0.1 to 0.5 (Δu=0.1). Usually, it is meaningless to find communities on network where more than half of them are overlapped.



(a) *Modularity* (Q) (b) *Jaccard Index*

Figure 12: Effectiveness and accuracy of varying u

We evaluate the performance in terms of the effectiveness (modularity score Q) and the accuracy (*Jaccard index*) of the concerned algorithms and illustrate the results in Fig. 12. The modularity of most algorithms except gCluSkeleton declines linearly when u grows, while the Jaccard index of them shows different change tendencies: CNM and gCluSkeleton deteriorate much faster than others while LPA and HANP are more stable and insensitive to the mixture degree.

Remarks. We can find that all the algorithms perform worse on datasets with higher mixture degrees, and both the effectiveness and accuracy would decrease when the communities overlap more with each other and become less discriminative. In comparison, the label propagation algorithms are more stable and show better capability of distinguishing communities from confusing mixtures.

6.8 Outliers

Focusing only on the produced clusters in the community detection tasks, sometimes we may be hoodwinked and come to a conclusion unilaterally. Because of the existence of outliers, we should also care about how many nodes are discriminated outside the groups meanwhile. As we discussed in Sec. 2.1, outliers can be affirmed automatically by the original algorithms or the predefined threshold *mvs*. In the former case, dense groups of nodes are revealed from the network so that the rest parts are naturally regarded as outliers. In the latter case, a proper partition of the entire network is obtained so that the components whose sizes are less than *mvs* will be disbanded, and the members of them are considered as outliers. In this subsection, we evaluate the extent to which these algorithms identify nodes as outliers, as shown in Fig. 13.

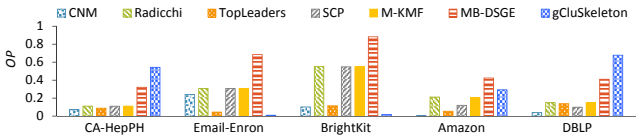


Figure 13: Proportion of produced outliers

We test the ability of discovering outliers for these algorithms on Football which contains 65 labeled outliers. When $mvs=2$, CNM, Radicchi, SCP, M-KMF, MB-DSGE and gCluSkeleton can identify outliers with the F_1-Score higher than 0.90, while others rarely consider outliers. We also conducted experiments on large-scale real-world networks without any priori knowledge about the exact outliers, and illustrate the proportion of outliers ($OP=|Outs|/|V|$) in Fig. 13. Here *mvs* is set to 3 since on large-scale networks, we think only two people cannot form a community [28]. LPA and HANP are not included in the figure since they produced rare outliers. Once the label distribution tends to be steady, there are few tiny groups flagged with unique labels (2 on both Amazon and DBLP for HANP, 57 and 32 on them for LPA). Similarly, outliers found by CNM and TopLeaders are also quite few. Since MB-DSGE has higher preference towards local dense connected groups, it produces more outliers. Especially, if we set κ to 4 for SCP, or increase k from 1 to 3 for M-KMF, more outliers will be produced for stricter limitation of Π. For gCluSkeleton, we set $\mu=3$ only for Amazon and DBLP to ensure better effectiveness of communities, which may lead to more outliers.

Remarks. By label propagation algorithms, we can hardly get outliers in a network. Compared with other algorithms we studied, generally, MB-DSGE produces the most outliers and has the lowest coverage ratio ($1-OP$) of a network. The outliers produced by SCP, M-KMF and gCluSkeleton are too sensitive to the value of the key parameter, i.e. κ, k and μ, respectively. Moreover, we can find that outliers are easily to be revealed on datasets with lower cc_{avg} (e.g. only 0.1723 of BrightKit).

Table 6: Comparison between LPA and LPA*

Alg.	CA-HepPH			Email-Enron			Gowalla			DBLP		
	CC	*S*	*Q*	*CC*	*S*	*Q*	*CC*	*S*	*Q*	*CC*	*S*	*Q*
LPA	0.7037	0.5985	0.4201	0.7340	0.7912	0.2983	0.7685	0.7902	0.3690	0.7317	0.6149	0.6930
LPA*	**0.7227**	**0.6617**	**0.4449**	**0.7408**	**0.8118**	**0.4954**	**0.7751**	**0.8058**	**0.5303**	**0.7381**	**0.6404**	**0.7032**

Alg.	S_10K			S_100K			S_200K			S_500K		
	F_{same}	*Jaccard*	*NMI*	*F same*	*Jaccard*	*NMI*	F_{same}	*Jaccard*	*NMI*	F_{same}	*Jaccard*	*NMI*
LPA	0.9681	0.9945	0.9992	0.9967	0.9917	0.9992	0.9935	0.9783	0.9969	0.9980	0.9944	0.9989
LPA*	**0.9980**	0.9940	**0.9993**	**0.9979**	**0.9945**	**0.9995**	**0.9984**	**0.9956**	**0.9994**	**0.9992**	**0.9976**	**0.9996**

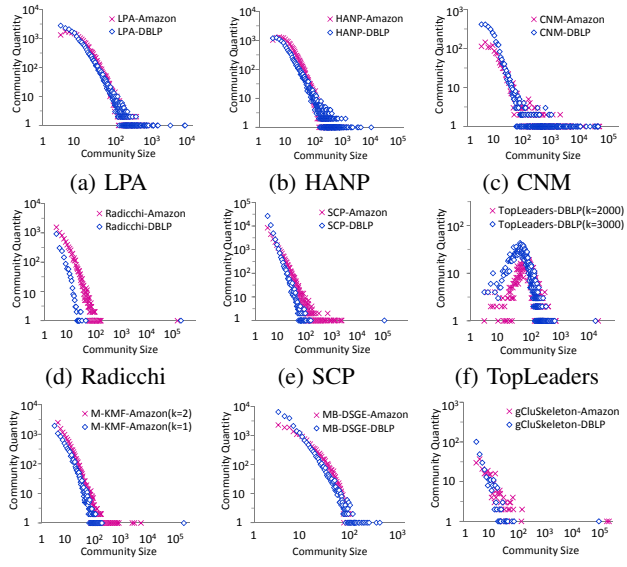
Table 7: Comparison between MB-DSGE and MB-DSGE*

Alg.	Strike						Football					
	CC	*S*	*Q*	F_{same}	*Jaccard*	*NMI*	*CC*	*S*	*Q*	F_{same}	*Jaccard*	*NMI*
MB-DSGE	0.7508	0.5000	0.2864	0.7708	0.4438	0.5777	0.4266	0.8000	0.5179	0.6139	0.8040	0.9767
MB-DSGE*	**0.7747**	**0.8333**	**0.5474**	**0.9583**	**0.8208**	**0.8660**	**0.5097**	0.6923	**0.5408**	**0.6389**	0.7154	**1.0000**

Alg.	CA-HepPH			BrightKit			Gowalla			DBLP		
	CC	*S*	*Q*	*CC*	*S*	*Q*	*CC*	*S*	*Q*	*CC*	*S*	*Q*
MB-DSGE	0.5865	0.2834	0.3929	0.5443	0.1952	0.0961	0.5508	0.2165	0.1075	0.6303	0.3830	0.3964
MB-DSGE*	**0.6223**	**0.3106**	**0.4437**	**0.6672**	**0.2715**	**0.1940**	**0.6269**	**0.2579**	**0.1883**	**0.6686**	**0.4413**	**0.5166**

6.9 Distribution of Communities

The distribution of communities, i.e. the distribution of quantities of communities with different sizes, is also a key indicator of the validity of an algorithm. We visualize the community distributions produced by various algorithms on Amazon and DBLP in Fig. 14 (with double logarithmic coordinates) for comparison. We find that LPA and HANP produce regular power law distributions, while CNM prefers more large communities so that the distributions are more plain and long-tailed. For SCP, when κ =3, the distribution on Amazon is reasonable while the distribution on DBLP is uneven due to an undivided large-scale community whose size is about 10^5. It may result from the small value of κ which makes different κ-1-cliques form together more easily. The results of Radicchi and M-KMF are quite homogeneous due to the similar ϕ. As discussed in Sec. 6.8, undesirable excessive outliers will be found by M-KMF if we increase k. For MB-DSGE, even with an expectation for community density of 0.1, the results also show remarkable sparsity. Unlike M-KMF, MB-DSGE only takes those dense groups away, leaving others as poor outliers. The community distribution generated by gCluSkeleton is a little messy which may be caused by the lower differentiation of the values of ϕ, i.e. the values of the ϵ-queue, on large-scale networks. At last, we find that a predefined *ld* makes the detection much more purposeful, and thus TopLeaders presents quite dissimilar distributions.



(a) LPA (b) HANP (c) CNM

(d) Radicchi (e) SCP (f) TopLeaders

(g) M-KMF (h) MB-DSGE (i) gCluSkeleton

Figure 14: Distributions of community size of each algorithm

Remarks. The distributions produced by CNM, LPA and HANP are closer to regular power law distribution. TopLeaders has a different community distribution with other algorithms due to the predefined leader number. On large-scale networks, for many algorithms such as Radicchi, M-KMF, SCP as well as gCluSkeleton, the existence of an undivided large component, which means a lower separation degree, is actually an intractable problem to be solved.

6.10 Diagnosis Effects

Based on the two targeted diagnoses discussed in Sec. 5.1 and 5.2, we compare the modified algorithms with their original ones respectively, i.e. MB-DSGE* vs. MB-DSGE and LPA* vs. LPA, to verify the effects of improvement.

Table 6 shows the superiority of LPA* on both real-world and synthetic networks. Similarly, MB-DSGE* also leads obvious improvements of the performance on real-world networks, as shown in Table 7. For lack of space, here we only present the result on part of the datasets we used. Based on the verification, we find LPA* shows significant improvements over LPA which has already outperformed other algorithms on many datasets. MB-DSGE* also leads to better performance on most metrics, in both accuracy and effectiveness. In particular, we find MB-DSGE* can significantly reduce the proportion of the excessive produced outliers, as shown in Table 8. In sum, the diagnoses based on our framework can improve the performance in accuracy and effectiveness, and both modifications will not increase the processing time.

Table 8: Reduction of outlier proportion

Alg.	CA-HepPH	Gowalla	BrightKit	Amazon	DBLP
MB-DSGE	0.3218	0.7692	0.8841	0.4244	0.4105
MB-DSGE*	**0.1220**	**0.4027**	**0.4882**	**0.1446**	**0.1665**

6.11 Summary

According to the above comprehensive evaluations covering multi-aspects of community detection, we can find that each approach has its own merits and faults. They vary in the efficiency of processing, have various adaptabilities to different metrics of performance as well as different characteristics of datasets. To sum up, Table 9 shows an intuitive summary on the performance, adaptability and competitiveness of the studied approaches. We give general ratings from multiple critical aspects in this table, including process efficiency (Effi.), accuracy (Accu.), effectiveness (Effe., in terms of *CC*, *S* and *Q*), the stability w.r.t. density sensitivity (Dens.) or mixture sensitivity (Mixt.), as well as to what extent an algorithm can avoid producing excessive outliers (Outl.).

From the perspective of *efficiency*, we see that M-KMF is the most efficient approach, while CNM, MB-DSGE and gCluSkeleton cost much more time than others.

Table 9: A visualized summary of studied approaches (more stars refer to better performances in the corresponding aspect)

Alg.	Effi.	Accu.	Effe.			Dens.	Mixt.	Outl.
			CC	*S*	*Q*			
CNM	⋆⋆	⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆
Radicchi	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆
LPA	⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆
HANP	⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆
TopLeaders	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆⋆
SCP	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆⋆⋆
M-KMF	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆	⋆⋆	⋆⋆	⋆⋆⋆
MB-DSGE	⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆
gCluSkeleton	⋆⋆	⋆	⋆⋆⋆⋆	⋆	⋆	⋆⋆⋆	⋆	⋆⋆

Among the most important aspects, the *accuracy* evaluates the performance of algorithms using datasets with ground-truth. In this respect, LPA and HANP perform best in our study, showing the strongest capability of identifying authentic communities.

As for the *effectiveness* which measures the inner- and intra-structures of the detected communities with various metrics, we find different algorithms with different characteristics show different performances w.r.t. different metrics. CNM, Radicchi, LPA, HANP and M-KMF prefer to produce communities with high clustering coefficients (*CC*), Radicchi and M-KMF tend to form communities with high strength (*S*), while CNM is the best one at finding communities with high modularity (*Q*).

These algorithms have different *sensitivities* with the density and the mixture degree of the datasets. Comparably speaking, LPA and HANP are the most stable approaches which keep excellent performance even when networks become much denser or the communities become highly mixed. Moreover, Radicchi and CNM are also less sensitive with the varying network density.

Considering the tendency of producing *outliers*, MB-DSGE tends to pick out excessive outliers and performs better on networks with more outliers. While LPA and HANP hardly identify nodes as singletons and are more applicable on networks with few outliers. Besides, the *community distributions* produced by LPA, HANP and CNM are closest to the power law distribution.

All in all, with an acceptable time cost, the label propagation algorithms are more excellent and stable approaches which could produce more accurate and valid communities with reasonable community distributions and fewer outliers under most circumstances.

7. CONCLUSIONS

In this paper, we conduct a comprehensive benchmarking study on approaches to community detection in social networks. Within the proposed benchmark, we formulate a generalized procedure-oriented framework, with high abstraction and nice modularization of the fundamental factors and critical steps of this problem. We have re-implemented ten state-of-the-art representative algorithms by mapping them to the proposed framework, and make in-depth evaluations on them based on our benchmark using both real-world and synthetic datasets. We discuss their merits and faults thoroughly, draw a set of interesting take-away conclusions, and provide intuitive ratings. In addition, we present how to make diagnoses for these algorithms based on our framework, and report significant improvements in the experimental study.

8. ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No. 61170064, No. 61373023) and the National High Technology Research and Development Program of China (No. 2013AA013204).

9. REFERENCES

[1] C. C. Aggarwal and H. Wang. A survey of clustering algorithms for graph data. *Managing and Mining Graph Data*, pages 275–301. Springer, 2010.

[2] B. Bollobás. *Modern Graph Theory*, volume 184. Springer, 1998.

[3] D. Bortner and J. Han. Progressive clustering of networks using structure-connected order of traversal. *ICDE*, pages 653–656, 2010.

[4] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *TKDE*, 24(7):1216–1230, 2012.

[5] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004.

[6] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005.

[7] R. Diestel. *Graph Theory*, volume 173. Springer, 2000.

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

[9] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. *VLDB*, pages 721–732, 2005.

[10] L. Hamers, Y. Hemeryck, G. Herweyers, M. Janssen, H. Keters, R. Rousseau, and A. Vanhoutte. Similarity measures in scientometric research: the jaccard index versus salton's cosine formula. *Information Processing & Management*, 25(3):315–318, 1989.

[11] J. Huang, H. Sun, Q. Song, H. Deng, and J. Han. Revealing density-based clustering structure from the core-connected tree of a network. *TKDE*, 25(8):1876–1889, 2013.

[12] R. R. Khorasgani, J. Chen, and O. R. Zaïane. Top leaders community detection approach in information networks. *SNA-KDD*, 2010.

[13] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. *Link Mining: Models, Algorithms, and Applications*, pages 337–357. Springer, 2010.

[14] J. M. Kumpula, M. Kivelä, K. Kaski, and J. Saramäki. Sequential algorithm for fast clique percolation. *Physical Review E*, 78(2):026109, 2008.

[15] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[16] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[17] I. X. Leung, P. Hui, P. Lio, and J. Crowcroft. Towards real-time community detection in large networks. *Physical Review E*, 79(6):066107, 2009.

[18] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.

[19] M. E. Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, 2006.

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

[21] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. *CIKM*, pages 1677–1681, 2012.

[22] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. *WWW*, pages 225–236, 2014.

[23] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *PNAS*, 101(9):2658–2663, 2004.

[24] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[25] V. Satuluri and S. Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. *SIGKDD*, pages 737–746, 2009.

[26] L. Tang and H. Liu. Community detection and mining in social media. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2(1), pages 1-137, 2010.

[27] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys*, 45(4):43, 2013.

[28] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *ICDM*, pages 745–754, 2012.

[29] F. Zhao and A. K. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *VLDB*, pages 85–96, 2012.