Clustering, Community Partition and Disjoint Spanning Trees

CUN-QUAN ZHANG AND YONGBIN OU

West Virginia University

Abstract. Clustering method is one of the most important tools in statistics. In a graph theory model, clustering is the process of finding all dense subgraphs. A mathematically well-defined measure for graph density is introduced in this article as follows. Let $G = (V, E)$ be a graph (or multi-graph) and H be a subgraph of G. The dynamic density of H is the greatest integer k such that $\min_{\forall \mathcal{P}}\{\frac{|E(H/\mathcal{P})|}{|V(H/\mathcal{P})|-1}\} > k$ where the minimum is taken over all possible partitions $\mathcal{P}$ of the vertex set of H, and $H/\mathcal{P}$ is the graph obtained from H by contracting each part of $\mathcal{P}$ into a single vertex. A subgraph H of G is a level-k community if H is a maximal subgraph of G with dynamic density at least k. An algorithm is designed in this paper to detect all level-h communities of an input multi-graph G. The worst-case complexity of this algorithm is upper bounded by $O(|V(G)|^2h^2)$. This new method is one of few available clustering methods that are mathematically well-defined, supported by rigorous mathematical proof and able to achieve the optimization goal with polynomial complexity. As a byproduct, this algorithm also can be applied for finding edge-disjoint spanning trees of a multi-graph. The worst-case complexity is lower than all known algorithms for multi-graphs.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms, trees, network problems*; G.2.3 [**Discrete Mathematics**]: Applications; G.3 [**Probability and Statistics**]—*Statistical computing*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Spanning trees, clustering, dense subgraph, polynomial algorithm, community, dynamic density, hierarchical clustering

ACM Reference Format:

Zhang, C.-Q. and Ou, Y. 2008. Clustering, community partition and disjoint spanning trees. ACM Trans. Algor. 4, 3, Article 35 (June 2008), 26 pages. DOI = 10.1145/1367064.1367075 http://doi.acm.org/10.1145/1367064.1367075

C.-Q. Zhang was supported by a grant of the National Security Agency (MDA904-01-1-0022) and a grant of WV EPSCoR.

Y. Ou was supported by a grant of WVU Research Corporation.

Authors' address: Department of Mathematics, West Virginia University, Morgantown, WV 26506-6310, e-mail: cqzhang@math.wvu.edu; ouyb@csbl.bmb.uga.edu.

1. *Introduction*

1.1. DENSE SUBGRAPHS AND CLUSTERING. Clustering method is an important statistical tool for data mining [Anthonisse 1971; Berkhin 2002; Bezdek 1981; Bradley and Fayyad 1998; Cover and Hart 1967; D'andrade 1978; Ding and He 2004; Dunn 1973; Fraley and Raftery 1998; Freeman 1977; Girvan and Newman 2002; Han and Kamber 2000; Hartigan 1975; Hoppner et al. 1999; Jain et al. 1999; Johnson 1967; Lampinen et al. 2002; MacQueen 1967; Massart and Kaufman 1983; Moore 2001; Pelleg and Moore 1999; Radicchi et al. 2004; SAS Institute, Inc.; Seidman 1983; Steinbach et al. 2000; Stephen 1994; Wasserman and Faust 1994; Wu and Huberman 2004; Xu et al. 2002]. It has been applied in many areas such as social science, internet network, transportation, bioinformatics, image processing, etc. Clustering is the process of detecting all dense subgraphs. In this article, we introduce a new, polynomial clustering method with a mathematically well-defined optimization goal.

Note that all graphs considered in this article are multigraphs. That is, parallel edges are allowed.

Definition 1.1. Let $G = (V, E)$ be a graph. Let H be a subgraph of G. The dynamic density of H is the greatest integer k such that

$$\min_{\forall \mathcal{P}} \left\{ \frac{|E(H/\mathcal{P})|}{|V(H/\mathcal{P})| - 1} \right\} > k, \tag{1}$$

where the minimum is taken over all possible partitions $\mathcal{P}$ of the vertex set of H, and $H/\mathcal{P}$ is the graph obtained from H by contracting each part of $\mathcal{P}$ into a single vertex. And, as a default, a single vertex is of dynamic density k for any integer k.

Definition 1.2. A subgraph H of G is a level-k community if H is a maximal subgraph of G with dynamic density at least k.

The dynamic density measures the internal connectivity of a subgraph dynamically and globally, which provides a criterion for the detection of community and community classification. As a corollary of a fundamental theorem in graph theory (Lemma 1.6), dynamic density can be equivalently defined as follows:

PROPOSITION 1.3. *Let H be a subgraph of G and k be a positive integer. The dynamic density of H is at least k if and only if, for every edge e_0 of H, $H \setminus \{e_0\}$ contains k edge-disjoint spanning trees.*

Proposition 1.3 enables us to design algorithms (Algorithm 1 and Algorithm 2) to solve the following optimization problem.

PROBLEM 1.4 (THE CLUSTERING PROBLEM)
Instance: *A graph G with $w(e)$ as the multiplicity for each edge e and an integer h.*

Question: *For each $k \in \{0, \ldots, h\}$ find all level-k communities in G (and construct a hierarchical tree to describe their inclusion relations).*

THEOREM 1.5 [ZHANG AND OU 2006]. *Algorithm* 1 *and Algorithm* 2 *produce optimal solutions for Problem* 1.4. *The worst-case complexity of Algorithm* 2 *is upper bounded by $O(n^2h^2)$, $O(nmh)$ and $O(nm\Delta)$ where $n = |V(G)|$, $m = |E(G)|$ and Δ is the maximum degree of G.*

The clustering method (and its density criterion as well) introduced in this article is one of few graph clustering methods based on rigorous mathematical proofs and having well-defined optimization goals. (See Lukashin and Fuchs [2001], Massart and Kaufman [1983], Radicchi et al. [2004], SAS Institute, Inc., Xu et al. [2002] for comments about existing graph-clustering methods.)

1.2. Edge-Disjoint Spanning Trees

LEMMA 1.6 [TUTTE 1961 AND NASH-WILLIAMS 1961]. *A graph H contains k edge-disjoint spanning trees if and only if*

$$\min_{\forall \mathcal{P}} \left\{ \frac{|E(H/\mathcal{P})|}{|V(H/\mathcal{P})| - 1} \right\} \geq k \tag{2}$$

where the minimum is taken over all possible partitions $\mathcal{P}$ of the vertex set of H, and $H/\mathcal{P}$ is the graph obtained from H by contracting each part of $\mathcal{P}$ as a single vertex.

Although Algorithm 1 and Algorithm 2 are designed mainly for detecting all level-h communities, it can also be applied for determining the number of edge-disjoint spanning trees with some slight simplification of the algorithms.

PROBLEM 1.7 (THE EDGE-DISJOINT SPANNING TREE PROBLEM)
Instance: *A graph G with $w(e)$ as the multiplicity for each edge e.*

Question: (1) *Determine the maximum number of edge-disjoint spanning trees of G and* (2) *find a maximum set of edge-disjoint spanning trees of G.*

THEOREM 1.8. *Algorithm* 1 *and Algorithm* 2 *produce optimal solutions for Problem* 1.7. *The worst-case complexity of Algorithm* 2 *is upper bounded by $O(n^2h^2)$, $O(nmh)$, $O(m^2)$ and $O(nm\Delta)$ where $n = |V|$, $m = |E|$, Δ is the maximum degree of G, and h is the maximum number of edge-disjoint spanning tress.*

The optimization problem (Problem 1.7) has been extensively studied. (Some related problems, such as, network reinforcement [Barahona 2004], are also related to the problem of edge-disjoint spanning trees.) The following is a list of complexities of some earlier algorithms for Problem 1.7 (see Barahona [1995, 2004] and Gabow [1995b] for the most updated and comprehensive survey on all earlier results on this subject).

For general cases (multigraphs).

Cunningham [1984]	$O(nm^8)$;
Cunningham [1985]	$O(nm)\phi_M$ or $O(n^4m)$;
Gabow [1995a]	$O(n^4m^2 \log^2 W)$;
Gusfield [1991]	$O(n^3m)$;
Barahona [1995]	$O(n^2)\phi_M$;
Gabow [1998]	$O(n^2m \log \frac{n^2}{m})$;
Barahona [2004]	$O(n)\phi_M$;
etc.	

NOTE: $\phi_M = O(nm \log \frac{n^2}{m})$ is the complexity of the min-cut max-flow problem (by Goldberg and Tarjan [1998]), and W is the maximum edge capacity.

For simple graphs (no parallel edges).

Roskind and Tarjan [1985]	$O(m \log m + k^2 n^2)$;
Gabow and Westermann [1992]	$O(nm \log \frac{m}{n})$;
etc.	

NOTE: k is the number of edge-disjoint spanning trees in the graph.

1.3. REMARK – NEW CONTRIBUTIONS AT CONCEPT LEVEL AND AT ALGORITHM LEVEL Note that the inequality (1) in the definition of dynamic density is strict, while the inequality (2) in the Tutte-Nash-Williams theorem is not. The strict inequality guarantees an extra edge in each community. That is, in each level-k community H, for every edge $e \in E(H)$, $H - \{e\}$ contains at least k edge-disjoint spanning trees (Proposition 1.3).

This difference, though only one extra edge e, is critically important in the algorithm. The tracing and searching of replaceable edges of e play a central role in determining the community (Substep 4-1) and expanding the last forest (Substep 4-2). This extra edge enables us not only to get an algorithm (Algorithm 2) with lower worst-case complexity for the spanning tree problem (Problem 1.7), but also to provide a traceable processing of the detection of communities (Problem 1.4).

2. *Spanning Trees and Communities*

THE PROOF OF PROPOSITION 1.3. Assume that H is a graph satisfying the following inequality

$$\min_{\forall \mathcal{P}} \left\{ \frac{|E(H/\mathcal{P})|}{|V(H/\mathcal{P})| - 1} \right\} > k.$$

Let e be an arbitrary edge of H. In the graph $H - e$, we have the following inequality for every partition $\mathcal{P}$ of $V(H)$

$$\min_{\forall \mathcal{P}} \left\{ \frac{|E((H-e)/\mathcal{P})|}{|V((H-e)/\mathcal{P})| - 1} \right\} \geq k.$$

By Lemma 1.6, the graph $H - e$ contains k edge-disjoint spanning trees. This completes one direction of the proof. So, we assume that, for every edge e of H, the graph $H - e$ contains k edge-disjoint spanning trees $T_1, \ldots, T_k$. If there is a partition $\mathcal{P}$ of $V(H)$ such that

$$\min_{\forall \mathcal{P}} \left\{ \frac{|E(H/\mathcal{P})|}{|V(H/\mathcal{P})| - 1} \right\} \leq k,$$

then choose e' as a cross edge between two parts of $\mathcal{P}$. By contracting each part of $\mathcal{P}$ to a single vertex, each $T_i/\mathcal{P}$ becomes a connected, spanning subgraph of $(H - e')/\mathcal{P}$ each of which contains at least $|\mathcal{P}| - 1$ edges (the number of cross edges in $T_i/\mathcal{P}$). The sum of all those cross edges (including the deleted edge e') would contradict the above inequality and therefore, completes the proof of the other direction. □

3. *A Basic Algorithm*

Two algorithms (Algorithm 1 and Algorithm 2) will be introduced in this article. Algorithm 1, the basic one, addresses the fundamental idea in mathematics with a simple labeling system. The purpose of introducing Algorithm 1 first is that it will be easier for readers to understand the main idea of the process, and its mathematical correctness (by avoiding the complication of labeling systems in its modified/improved versions).

Algorithm 2, a modified/improved version of the basic one, is introduced later for a lower worst-case complexity and, unavoidably, has a complicated labeling system.

3.1. Brief Description and Outline of the Algorithm

In order to provide a clear overview of the algorithm for readers, we are to briefly discuss the main idea of the algorithm before its formal presentation.

Although the following lemma is well known and straightforward, it will be used frequently in this article.

LEMMA 3.1. *Let T be a spanning forest of a graph H and $e = xy \in E(H) - E(T)$.*

(1) *Suppose that x and y belong to two different components of T. Then $T + e$ is a forest larger than T.*

(2) *Suppose that x and y are in the same component of T. Let P be the unique path of T joining x and y. Then, for every $e' \in P$, $T + e - e'$ is a forest with the same size as T.*

Let H be a graph with k edge-disjoint spanning forests $T_1, \ldots, T_k$ where $T_1, \ldots, T_{k-1}$ are trees while T_k might be not, which is expected to be expanded if it is possible.

Let $e_0 \in E(H) - \bigcup_{\mu=1}^{k} E(T_\mu)$, which is an extra edge not used in any existing forest.

First, the edge e_0 is colored *red*. Then, for a spanning tree T_μ, the subgraph $T_\mu + e_0$ contains a circuit C (by Lemma 3.1). Color all these edges in C with red. Note that, for each red edge e', $H - e'$ contains k edge-disjoint forests, $T'_1, \ldots, T'_k$ that $T'_\nu = T_\nu$ if $\nu \neq \mu$, and, $T'_\mu = T_\mu + e_0 - e'$. That is why these red edges are called *replaceable edges* from the extra edge e_0.

Second, for each red edge e'' of H and for each forest T_j, color all edges of the unique circuit contained in $T_j + e''$ (if this circuit exists). This step is to be repeated until one of the following two cases occurs.

(1) A red edge $e^{(1)}$ joins two distinct components of the last forest T_k;

(2) No more edge can be further colored.

If (1) happens, $T_k + e^{(1)}$ is larger than T_k. However, the edge $e^{(1)}$ may not be that extra edge e_0. That is, it is possible that the edge $e^{(1)}$ came from a spanning tree T_{t_2}. Now, let us trace back our coloring process. The edge $e^{(1)}$ was colored red because it is contained in a circuit of $T_{t_2} + e^{(2)}$, where $e^{(2)}$ was already colored red before the edge $e^{(1)}$ was colored. Continue this trace back procedure, we find a sequence of edges $e^{(1)}, \ldots, e^{(s)}$ and a sequence of spanning forests $T_{t_2}, \ldots, T_{t_s}$

such that $e^{(\mu-1)}$ was colored red because it is contained in the circuit of $T_{t_\mu} + e^{(\mu)}$, for every $\mu = 2, \cdots, s$ and $e^{(s)}$ is the extra edge e_0 that the process started with. Now, the last forest T_k is ready to be expanded by the following adjustment,

$$T_k \leftarrow e^{(1)}$$

and

$$T_\mu \leftarrow T_\mu + e^{(\mu)} - e^{(\mu-1)}$$

for every $\mu = 2, \ldots, s$.

If (2) happens, we can show that the subgraph H' induced by all red edges is of dynamic density at least k. It is obvious that, at every stage of coloring process, the subgraph induced by all red edges (at that time) is always connected. Therefore, H' is connected. Furthermore, for every $\nu = 1, \ldots, k$, $H' \cap T_\nu$ is connected as well, for otherwise, some red edge e^* must join two components of $T_\nu \cap H'$, for some ν. Note that, the process should not be stopped since either the circuit of $T_\nu + e^*$ contains some un-colored edge, or T_k can be expanded (if e^* joins two components of $T_k = T_\nu$). So, the subgraph H' contains k edge-disjoint spanning trees $T_1 \cap H', \ldots, T_k \cap H'$ and an extra edge e_0. By the replaceability of every red edge $e \in E(H)$, $H - e$ contains k edge-disjoint spanning trees. Hence, by Proposition 1.3, H' is of dynamic density at least k.

In the formal presentation of Algorithm 1, the set B_p is the collection of all red colored edges, while the labels S and I will help us in the tracing procedure (if an expansion of T_k is needed).

3.2. NOTATION AND LABELING SYSTEM The input graph G may have parallel edges. For each edge e, the multiplicity of e is denoted by $w(e)$ (the number of parallel edges between a pair of given vertices).

For a set $\mathcal{T}$ of spanning forests, the coverage of an edge e is the number of members of $\mathcal{T}$ containing the edge e, denoted by $c_{\mathcal{T}}(e)$. (In the algorithm, it is required that $c_{\mathcal{T}}(e) \leq w(e)$; that is, one can only use an edge at most $w(e)$ times). The set of edges with $c_{\mathcal{T}}(e) < w(e)$ (unsaturated edges) is denoted by E_0.

By Definition 1.1, a graph G a level-0 community if and only if G is connected. However, level-0 community is of little interest in applications. Thus, for the sake of convenience, with an excusable abusive use of notation, we simply consider any input graph G itself as a level-0 community in the initial iteration of our algorithm. (Note that, if Proposition 1.3 is used for the definition of dynamic density for all non-negative integers, then G is a level-0 community if and only if $E(G) \neq \emptyset$.)

A basic algorithm described in Section 3.3 makes use of a **"flag"** to act as a "stop and restart" sign for the completion of the current community search and the starting point of searching another community. This flag (the label f) is not "on" until the search of the current community ends.

This basic algorithm also makes use of a **"track record"**. A track record is a pair of sequences (edge-sequence $S(e)$ and index-sequence $I(e)$) associated with each edge of G. The purpose of a "track record" is to record the track of any possible adjustment and expansion of a set of spanning trees/forests.

Integer p counts the number of potential communities. Subset B_p collects all replaceable edges (defined in Section 4.1) and will either eventually generate a potential community, or be used in an adjustment for a possible expansion of the last forest T_k.

3.3. BASIC ALGORITHM The algorithm described bellow is a basic one, and its flowchart is attached in the appendix.

ALGORITHM 1 [ZHANG AND OU 2006].

Input: *A graph G with $w(e)$ as the multiplicity for each edge e and an integer h.*
Goal: *Find all level-h communities in G.*

Step 0. *Start at $k = 0$ (the program runs until $k = h$ level is complete), $H \leftarrow G$ (a level-0 community), $\mathcal{T} \leftarrow \emptyset$ (the set of spanning trees in H) and $c_{\mathcal{T}}(e) = 0$ for all e in $E(G)$.*

Step 1. (See S1a in flowchart.)
$k \leftarrow k + 1$. (Note: if $k > 1$, $\mathcal{T} = \{T_1, \ldots, T_{k-1}\}$ is a set of edge-disjoint spanning trees of H, and $c_{\mathcal{T}}(e)$ is the coverage of edge $e \in E(H)$. These are outputs of Step 6 in the previous iteration of the main loop of the algorithm. When $k = 1$, no spanning tree preexists).
Let E_0 be the set of edges (unsaturated edges) e with $c_{\mathcal{T}}(e) < w(e)$.
Let T_k be a spanning forest in H consisting of edges of E_0.
Apply either Kruskal's algorithm, or, Prim's algorithm so that T_k is maximum in the subgraph of H induced by edges of E_0. (See Roberts and Tesman [2004]*, p.* 737–742*).* (See S1b in flowchart.)
Let $\mathcal{T} \leftarrow \mathcal{T} + \{T_k\}$ and update the coverage $c_{\mathcal{T}}(e)$ and E_0 as follows: $c_{\mathcal{T}}(e) \leftarrow c_{\mathcal{T}}(e) + 1$ if e is an edge of T_k; otherwise, $c_{\mathcal{T}}(e) = c_{\mathcal{T}}(e)$ (no change), and delete all edges e in E_0 for which $c_{\mathcal{T}}(e) = w(e)$.
Go to Step 2.

Step 2. (See S2 in flowchart.) *For each $e \in E(H)$, let $f(e) \leftarrow 0$.*
Let $p \leftarrow 1$ and go to Step 3.

Step 3. (See S3a in flowchart.) *If $E_0 = \emptyset$, go to Step* 5.
Otherwise, (See S3b in flowchart.) *pick any $e_0 \in E_0$ and set the track record*

$$S(e_0) \leftarrow e_0; \quad I(e_0) \leftarrow \emptyset.$$

Let $B_p \leftarrow \{e_0\}$, and let $e' \leftarrow e_0$. Go to Substep 4-1 *of Step* 4 *(one does not need to perform the checks in the first part of Step* 4 *because e_0 is in E_0 and T_k was constructed to be maximal relative to E_0.)*

Step 4. (See S4a in flowchart.)
If $f(e) = 1$ for every $e \in B_p$, (See S4b in flowchart)*, then $E_0 \leftarrow E_0 - B_p$, $p \leftarrow p + 1$ and go to Step* 3.
Otherwise, (See S4c in flowchart) *pick $e_1 \in B_p$ with $f(e_1) = 0$. Does e_1 join two components of T_k? If yes, go to Substep* 4-2*; if no, let $e' \leftarrow e_1$ and go to Substep* 4-1.

Substep 4-1. (See S4-2b in flowchart.)
Let x' and y' be the endvertices of the edge e'.
For each $i = 1, 2, \ldots, k$, Let P be the path of T_i between the vertices x' and y'. For each $e \in E(P) - B_p$, let

$$S(e) \leftarrow S(e')e;$$

$$I(e) \leftarrow I(e')i.$$

Let

$$B_p \leftarrow B_p \cup E(C_i).$$

Let $f(e') \leftarrow 1$.
Go to (the beginning of) Step 4.

Substep 4-2. (See S4-1 in flowchart.)
Adjust the forests as follows: Suppose

$$S(e_1) = e_{i_1}, e_{i_2}, \ldots, e_{i_t};$$

$$I(e_1) = \mu_1, \mu_2, \ldots, \mu_{t-1}.$$

For $j = 1, \ldots, t-1$, let $T_{\mu_j} \leftarrow T_{\mu_j} + e_{i_j} - e_{i_{j+1}}$ and let $T_k \leftarrow T_k + e_1$.

Also update the coverage $c_{\mathcal{T}}(e)$ as follows: $c_{\mathcal{T}}(e) \leftarrow c_{\mathcal{T}}(e) + 1$ if $e = e_0$, otherwise, $c_{\mathcal{T}}(e) = c_{\mathcal{T}}(e)$ (also delete e_0 from E_0 if $c_{\mathcal{T}}(e_0) = w(e_0)$). (See S4-1b in flowchart.) *Let $f(e) \leftarrow 0$ for every e and $B_p \leftarrow \emptyset$.*

After the adjustment, go to Step 3.

Step 5. (See S5 in flowchart.) *For $i, j = 1, 2, \ldots, p$ with $i \neq j$, if $V(B_i) \cap V(B_j) \neq \emptyset$, then merge B_i and B_j.*

Step 6. (See S6a in flowchart.) *If $k = h$* (See S6b in flowchart)*, output all B_i, each of which is a level-h community of G. If $k < h$,* (See S6c in flowchart)*, then repeat Step* 1 *for $H \leftarrow G[B_i]$ for every i, and $\mathcal{T} = \{T_1|H, T_2|H, \ldots, T_k|H\}$, the set of edge-disjoint spanning trees in H (as inputs of Step 1 for the next run of the algorithm at level $(k+1)$).*

3.4. MORE DETAILED EXPLANATION OF THE ALGORITHM Communities are detected level-by-level in a graph G. The main part of the algorithm is repeated for each $k = 1, 2, \ldots, h$. For each k, the algorithm detects all level-k communities in a previously identified level-$(k-1)$ community H. The following is a more detailed description of the main idea of the algorithm.

(a) Let $T_1, T_2, \ldots, T_{k-1}$ be $(k-1)$ edge-disjoint spanning trees of a level-k community H, and let T_k be a spanning forest in $H - E(T_1) - E(T_2) - \cdots - E(T_{k-1})$. (This is a part of Step 1.)

(b) Find an unused edge e_0 (e_0 is not in any forest $T_i : i = 1, \ldots, k$). Label all edges of the unique circuit contained in $T_i + e_0$. Repeat this procedure for each $i = 1, \ldots, k$. (Labeled edges are added into B_p – a potential level-k community. This is a part of Step 3 and Substep 4-1.)

(c) For each edge e' of B_p, repeat this procedure again until one of the following two cases occurs: (1) some new edge e' of B_p joins two components of the last forest T_k; or (2) the "flag" is turned on. (In the basic algorithm, the "flag" is "on" when every edge in B_p is of f-value 1. This is a part of Substep 4-1 and Step 4.)

(d) If Case (1) of (c) occurs, that is, an edge e' is found in (b) or (c) that joins two components of T_k. Then, Substep 4-2 will be iterated for adjustment: the forest T_k is therefore expanded as follows: adding e_0 (the unused edge given in (b)) into some tree/forest and recursively replacing a sequence of edges into and out of a corresponding sequence of trees/forests (as described in Substep 4-2). That is, a new set of spanning trees/forest is to be constructed in $\bigcup_{i=1}^{k} E(T_I) \cup \{e_0\}$.

(e) If Case (2) of (c) occurs, every edge contained in the unique circuit of $T_i + e'$ is already contained in B_p for every edge e' of B_p, and, every spanning tree/forest T_i. Therefore, the subgraph B_p cannot be expanded further and is of dynamic density at least k. At this time, Substep 4-1 stops and the algorithm returns to Step 3 to search for another potential level-k community B_{p+1}.

(f) Combine overlapping subgraphs with dynamic density at least k. (This is Step 5.)

4. *Correctness of the Algorithm*

As described in the algorithm, let H be a level-$(k-1)$ community of a large graph G. Let $T_1, \ldots, T_k$ be edge-disjoint spanning forests of H where $T_1, \ldots, T_{k-1}$ are trees.

The algorithm consists of two major parts.

Part one – Expansion. Step 4-2 expands the set of forests by adding a new edge into them. Since T_k was chosen as a "maximum" in the subgraph $H - \bigcup_{\mu=1}^{k-1} E(T_\mu)$, adjustments to other forests are necessary in order to have some expansion. Step 4-1 records the "trace" of feasible adjustments to which the expansion in Step 4-2 follows.

Part two – Record of Communities. After a series of expansions, the set of forests reaches the maximum (i.e., nothing can be added any more), Step 4-1 starts to record all edges that will be output.

The following is some discussion and some necessary lemmas to verify the correctness of the algorithm.

Section 4.1 introduces and discusses the key idea of the algorithm – *"replaceability"*. Section 4.1.1 introduces some notation and proves the correctness of adjustment. Based on the conclusions of Sections 4.1.1, Section 4.1.2 proves the correctness of expansion, Section 4.2 verifies dynamic densities of the outputs and Section 4.3 proves that the output contains all level-k communities. An immediate corollary of those theorems in Sections 4.2 and 4.3 is the following main result.

THEOREM 4.1. *Algorithm* 1 *designed in Section* 3.3 *is correct. That is, Algorithm* 1 *finds all level-k communities of a graph H*.

4.1. REPLACEABILITY AND FIXED FRAME OF REFERENCES

The key issue of the process is the concept of "replaceability", which can be roughly described as follows: Let $\{T_1^{(0)}, \ldots, T_k^{(0)}\}$ be a set of edge-disjoint spanning forests of graph H with $e' \notin \bigcup_{\mu=1}^{k} E(T_\mu^{(0)})$ and $e'' \in \bigcup_{\mu=1}^{k} E(T_\mu^{(0)})$. If (after a series of adjustments), there is a set of edge-disjoint forests $\{T_1^{(t)}, \ldots, T_k^{(t)}\}$ that contains e' but not e'', then we say that *"edges e' and e'' are replaceable with each other"* (of course, $|T_i^{(t)}| = |T_i^{(0)}|$.)

4.1.1. *The Proof of Replaceability*

The following lemma (Lemma 4.2) proves the replaceability of $e_0 \in E_0$ and any edge $e \in B_p$.

LEMMA 4.2. *For an edge $e \in B_p$ with*

$$S(e) = e_0 \cdots e_t \text{ and } I(e) = \mu_0 \cdots \mu_{t-1}.$$

Denote $T_\mu^{(0)} \leftarrow T_\mu$. And for each $h = 1, \ldots, t$, denote

$$T_\mu^{(h)} \leftarrow T_\mu^{(h-1)} \text{ if } \mu \neq \mu_h \text{ and } T_{\mu_h}^{(h)} \leftarrow T_{\mu_h}^{(h-1)} + e_{h-1} - e_h.$$

Then

(1) $\{T_1^{(h)}, \ldots, T_k^{(h)}\}$ *is a set of edge-disjoint forest of G, for each $h = 0, \ldots, t$, and*

(2)

$$\bigcup_{\mu=1}^{k} T_\mu \cup \{e_0\} = \bigcup_{\mu=1}^{k} T_\mu^{(t)} \cup \{e_t\}.$$

Note that $e_0 \in E_0$ and $e = e_t$, and the edge $e = e_t$ is added into B_p after Step 4-1 has been iterated t times without any interruption of Step 4-2 (otherwise, the record of B_p would be erased after an expansion of T_k).

It is sufficient to show part (1) of Lemma 4.2, while part (2) of lemma is guaranteed by the process of the algorithm and the correctness of part (1).

It is easy to see that $\{T_1^{(t)}, \ldots, T_k^{(t)}\}$ is a set of edge-disjoint spanning forests (by the algorithm).

Therefore, we only need to show that, for each $\mu \in \{1, \ldots, k\}$, each $T_\mu^{(t)}$ remains a forest/tree (it is proved in the following lemma).

For a forest T and an edge $e \notin T$, the unique circuit (if it exists) of $T + e$ is denoted by $C(T, e)$. For two vertices $x, y \in V(T)$, the unique path (if it exists) in T joining x and y is denoted by $P(T; x, y)$.

Applying the notation of Lemma 4.2, for each given tree $T_\mu = T_\mu^{(0)}$, let

$$\mu_{h_1} = \mu_{h_2} = \cdots = \mu_{h_q} = \mu$$

with

$$1 \le h_1 \le h_2 \le \cdots \le h_q \le t.$$

We are to show that each $T_\mu^{(h_i)} = T_\mu^{(h_{i-1})} + e_{h_i - 1} + e_{h_i}$ remains a tree/forest, for each $i = 1, \ldots, q$.

For the sake of convenience in the discussion of the next lemma, let

$$T^{(0)} \leftarrow T_\mu^{(0)}, \quad T^{(i)} \leftarrow T_\mu^{(h_i)}, \quad e_i \leftarrow e_{h_i - 1}, \quad f_i \leftarrow e_{h_i}$$

for each $i = 1, \ldots, q$.

LEMMA 4.3. *Let T be a tree and let*

$$T^{(0)}, (e_1, f_1), T^{(1)}, (e_2, f_2), T^{(2)}, \ldots, (e_q, f_q), T^{(q)} \tag{3}$$

be an alternative sequence (subgraphs and edge-pairs, alternatively) of length $2q + 1$ such that

$$T^{(0)} = T, \quad e_i \notin E(T) \bigcup \bigcup_{j=1}^{i-1} E(C(T, e_j)), \quad f_i \in E(C(T, e_i)) \setminus \bigcup_{j=1}^{i-1} E(C(T, e_j)) \tag{4}$$

and

$$T^{(i)} = T^{(i-1)} + e_i - f_i \tag{5}$$

for $i \in \{1, \ldots, q\}$.

Then, all of the following statements hold.

(α) *$T^{(q)}$ is a tree.*

(β) *$E(C(T^{(q-1)}, e_q)) \subseteq \bigcup_{i=1}^{q} E(C(T, e_i))$.*

(γ) *$\forall v_1, v_2 \in V(T)$, if $e \in E(P(T, v_1, v_2)) \setminus \bigcup_{i=1}^{q} E(C(T, e_i))$, then $e \in E(P(T^{(q)}, v_1, v_2))$.*

PROOF. Use induction on q. When $q = 1$, Claims (α) and (β) are trivial.

For Claim (γ), if $C(T, e_1)$ is disjoint from $P(T, v_1, v_2)$ it is also trivial. So assume $P_0 = v_s \cdots v_t$ is the common path of $C(T, e_1)$ and $P(T, v_1, v_2)$ and $f_1 \in E(P_0)$. Then $E(P(T^{(1)}, v_1, v_2)) = (E(P(T, v_1, v_2)) \cup E(C(T, e_1)))\backslash E(P_0)$. By assumption, $e \notin E(C(T, e_1))$. Therefore $e \in E(P(T^{(1)}, v_1, v_2))$.

Now assume the claims are true for any alternative sequence described in the lemma of length $2q' + 1$ with $1 \leq q' < q$ (i.e., they are in the form of (3) and satisfy (4) and (5)).

Claim (α). Since $f_q \notin \bigcup_{i=1}^{q-1} E(C(T, e_i))$, by the inductive hypothesis of Claim (γ), $f_q \in P(T^{(q-1)}, u_1, u_2)$ where $e_q = u_1u_2$, and so $T^{(q)} = T^{(q-1)} + e_q - f_q$ is a tree.

Claim (β). Let $T' = T^{(q-2)} + e_q - f_q$. Consider the alternative sequence

$$T^{(0)},\ (e_1, f_1),\ T^{(1)},\ (e_2, f_2),\ T^{(2)}, \ldots, T^{(q-2)},\ (e_q, f_q),\ T'$$

which is of length $2q - 1$. Note that Claim (β) doesn't involve the last tree $T^{(q)}$ at all, and it satisfies both Eqs. (5) and (4). Hence, we may apply the inductive hypothesis of Claim (β) to this sequence. Then, we have that

$$E\left(C\left(T^{(q-2)}, e_q\right)\right) \subseteq \left(\bigcup_{i=1}^{q-2} E(C(T, e_i))\right) \bigcup E(C(T, e_q)).$$

If $f_{q-1} \notin E(C(T^{(q-2)}, e_q))$, we have $C(T^{(q-1)}, e_q) = C(T^{(q-2)}, e_q)$ since $T^{(q-1)} = T^{(q-2)} + e_{q-1} - f_{q-1}$. This completes the proof for this case.

If $f_{q-1} \in E(C(T^{(q-2)}, e_q))$, then $C(T^{(q-1)}, e_q)$ is "detoured" on the other way of $C(T^{(q-2)}, e_{q-1})$. So $E(C(T^{(q-1)}, e_q)) \subseteq E(C(T^{(q-2)}, e_q)) \cup E(C(T^{(q-2)}, e_{q-1}))$. By the inductive hypothesis of Claim (β), $E(C(T^{(q-2)}, e_{q-1})) \subseteq \bigcup_{i=1}^{q-1} E(C(T, e_i))$, proving Claim (β).

Claim (γ). By inductive hypothesis of Claim (γ),

$$e \in P(T^{(q-1)}, v_1, v_2). \tag{6}$$

By Claim (β),

$$E(C(T^{(q-1)}, e_q)) \subseteq \bigcup_{i=1}^{q} E(C(T, e_i)). \tag{7}$$

Since the edge $e \notin \bigcup_{i=1}^{q} E(C(T, e_i))$, by Eq. (7), we have that

$$e \notin E(C(T^{(q-1)}, e_q)). \tag{8}$$

Now we may apply the inductive hypothesis of Claim (γ) to a shorter sequence $T^{(q-1)}, (e_q, f_q), T^{(q)}$. By Eqs. (6) and (8), $e \in P(T^{(q)}, v_1, v_2)$. This completes the proof of Claim (γ). □

4.1.2. *Expansion of the Last Forest T_k*

LEMMA 4.4. *After an expansion by Step* 4-2*, the set of edge-disjoint forests* $\{T_\mu : \mu = 1, \ldots, k\}$ *is expanded. That is, each* T_μ *for* $\mu = 1, \ldots, k - 1$*, remains as a tree while* $|T_k|$ *is larger than it was.*

This lemma is an immediate corollary of the following lemma (Lemma 4.5). We follow the notation of the previous subsection.

LEMMA 4.5. *An edge e_t joins two components of $T_k^{(t)}$ if and only if e_t joins two components of $T_k^{(0)} = T_k$*

PROOF. By Lemma 4.3 (γ) (and (α)). □

4.2. THE DYNAMIC DENSITY OF B_p

THEOREM 4.6. *The dynamic density of each B_p is at least k where B_p is an output of the algorithm.*

Note that T_ν's are changed if Step 4-2 is iterated. However, it can be seen that, after the completion of Step 4, $\{T_\nu : \nu = 1, \ldots, k\}$ is maximally expanded. Without causing any confusion, in the remaining part of this section, let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be the set of forests after the completion of Step 5, which remains the same as the input of Step 4 after the very last iteration of Step 4-2 and Step 3.

For the sake of simplification, the intersection of T_a and the induced subgraph $H[B_p]$ is denoted by $T_a \cap B_p$.

LEMMA 4.7. *$T_a \cap B_p$ is a spanning tree in B_p for every p and every $a \in \{1, \ldots, k\}$.*

Some notation introduced in Section 4.1.1 is used in the following proof.

PROOF. It is obvious that B_p induces a connected subgraph since it is built up by adding circuits (in Substep 4-1). Assume that $T_a \cap B_p$ is not connected. Since B_p is connected, let $e' = xy$ be an edge of $B_p - E(T_a)$ joining two different components of $T_a \cap B_p$.

Case 1. There is a path P joining x and y in T_a.

That is, the unique path P of T_a joining x and y contains some edge that is not in B_p. Thus, Substep 4-1 has not been iterated on the edge $e' \in B_p$ (for otherwise, after an iteration of Substep 4-1 on the edge e', all edges of P would be added into B_p). Therefore, the flag $f(e')$ remains 0 at this stage. This contradicts that $f(e) = 1$ for every $e \in B_p$ and B_p is an output after the completion of Step 4.

Case 2. There is no path joining x and y in T_a.

That is, $T_a = T_k$ which is the only possible nonconnected forest. Since $e' \in B_p$, let $S(e') = e'_0, e'_1, \ldots, e'_{t'}$, $I(e') = \mu_0, \mu_1, \ldots, \mu_{t'-1}$. Then, by Lemma 4.2, H has a set of edge-disjoint forests $\{T_\nu^{(t')} : \nu = 1, \ldots, k\}$ such that $e'_{t'} \notin$ any of those forests. Therefore, $e'_{t'}$ can be added into $T_k^{(t')}$ by an iteration of Step 4-2 and this contradicts that $\mathcal{T}$ has reached the maximum and Step 4-2 will not be iterated any more. □

We are now ready to prove Theorem 4.6.

PROOF OF THEOREM 4.6. By Proposition 1.3, it is sufficient to show that for every edge $e \in B_p$, $B_p - e$ contains k edge-disjoint spanning trees.

By Lemma 4.7, there is nothing to prove if $e \notin \bigcup_{i=1}^k E(T_i)$. So, assume that $e \in \bigcup_{i=1}^k E(T_i)$. Suppose $S(e) = e_0, e_1, \ldots, e_t$, $I(e) = \mu_0, \mu_1, \ldots, \mu_{t-1}$. Then,

by Lemma 4.2, the subgraph of G induced by $\bigcup_{i=1}^{k} E(T_i) + \{e_0\}$ contains a set of edge-disjoint spanning trees $\{T_i^{(t)} : i = 1, \ldots, k\}$ such that

$$\bigcup_{i=1}^{k} E(T_i) + \{e_0\} = \bigcup_{i=1}^{k} E(T_i^{(t)}) + \{e_t\}.$$

That is, $\{T_i^{(t)} \cap B_p : i = 1, \ldots, k\}$ is also a set of edge-disjoint spanning trees in B_p. □

4.3. The Inclusion of All Level-k Communities

THEOREM 4.8. *If Q is a subgraph of H with dynamic density at least k then $Q \subseteq B_p$ for some p, where B_p is the output of Step 5 (after the merge).*

PROOF OF THEOREM 4.8. Let $\{B_p : p \in \mathcal{O}\}$ be the output of the algorithm and let $\mathcal{T} = \{T_v^{(f)} : v = 1, \ldots, k\}$ be the set of forests after the completion of Step 5.

Let $\mathcal{P}$ be a partition of $V(H)$ with $\mathcal{P} = \{V(B_p) : p \in \mathcal{O}\} \cup (V(H) - \bigcup V(B_p))$. That is, each nontrivial member of $\mathcal{P}$ is the vertex set of some B_p. By Lemma 4.7, $T_v \cap B_p$ is a spanning tree in B_p for every $i \in \{1, \ldots, k\}$ and every $p \in \mathcal{O}$. Hence, in the graph $H/\mathcal{P}$, each $T_v/\mathcal{P}$ remains acyclic and maintains the same number of components.

Now, consider the subgraph Q and its corresponding contraction $Q/\mathcal{P}$. Assume that $Q \not\subseteq$ any B_p. Hence, $|V(Q/\mathcal{P})| \geq 2$. Note that

$$\frac{|E(Q/\mathcal{P})|}{|V(Q/\mathcal{P})| - 1} > k, \tag{9}$$

since the dynamic density of Q is at least k. Since each $T_v^{(f)}/\mathcal{P}$ remains acyclic in $Q/\mathcal{P}$ and is of size at most $(|V(Q/\mathcal{P})| - 1)$, $E(Q/\mathcal{P})$ must have some edge e that is not in any $T_v^{(f)}/\mathcal{P}$ (by (9)).

Since $e \notin$ any T_v. By Steps 2 and 3 of the algorithm, e must be in some B_p. But this indicates that the edge e is not in Q/P. This is a contradiction. □

5. *An Improved Algorithm (Working Zone Version)*

Algorithm 1 shows the most basic and fundamental ideas of the process. Following the same ideas, it can be further modified to improve the complexity. The algorithm introduced in this section is one of several modified versions of Algorithm 1.

5.1. Notation and Labeling System.

In order to have a lower worst-case complexity, the improved version has a more complicated data structure and corresponding labeling system than the basic one.

The following is a list of brief descriptions of data structures and corresponding labeling system in the sample algorithm (Algorithm 2).

(1) The set B_p in Algorithm 1 collects edges that are replaceable with an unsaturated edge e_0. In the following algorithm, the set B_p collects vertices so that all edges of $G[B_p]$ are replaceable with e_0.

Furthermore, for the purpose of reducing the complexity, the set B_p is created as a sequence (in Substep 4-1) so that the ordering of elements of B_p clearly indicates when and how those vertices were selected into the set B_p.

(2) The adjustment sequences $S(e)$ and $I(e)$ will be created when they are needed (in Substep 4-2) based on the information generated in Substep 4-1.

(3) For the purpose of reducing the complexity, the forests of $\mathcal{T}$ are all considered as rooted forests. An end-vertex of an unsaturated edge e_0 is the root.

5.2. MAIN PROGRAM OF THE ALGORITHM. The following is the main program of the algorithm.

ALGORITHM 2 [ZHANG AND OU 2006].

Input: *A graph G with $w(e)$ as the multiplicity for each edge e and an integer h.*
Goal: *Find all level-h communities in G.*

Step 0. *Start at $k = 0$ (the program runs until $k = h$ level is complete), $H \leftarrow G$ (a level-0 community), $\mathcal{T} \leftarrow \emptyset$ (the set of spanning trees in H) and $c_{\mathcal{T}}(e) = 0$ for all e in $E(G)$.*

Step 1.
$k \leftarrow k + 1$.
(Note: If $k > 1$, $\mathcal{T} = \{T_1, \ldots, T_{k-1}\}$ is a set of edge-disjoint spanning trees of H, and $c_{\mathcal{T}}(e)$ is the coverage of each edge. These are outputs of Step 6 in the previous iteration of the main loop of the algorithm. When $k = 1$, no spanning tree preexists).
Let E_0 be the set of edges (unsaturated edges) e with $c_{\mathcal{T}}(e) < w(e)$.
Find a maximum spanning forest T_0 in H consisting of edges of E_0.
Let $\mathcal{T} \leftarrow \mathcal{T} + \{T_0\}$ and update the coverage $c_{\mathcal{T}}(e)$ and E_0 as follows: $c_{\mathcal{T}}(e) \leftarrow c_{\mathcal{T}}(e) + 1$ if e is an edge of T_0; otherwise, $c_{\mathcal{T}}(e) = c_{\mathcal{T}}(e)$ (no change), and delete all edges e in E_0 such that $c_{\mathcal{T}}(e) = w(e)$.
Go to Step 2.

Step 2.
Let $p \leftarrow 1$ and go to Step 3.

Step 3. *If $E_0 = \emptyset$ then go to Step 5.*
Otherwise, go to Step 4.

Step 4.
Pick any $e_0 = xy \in E_0$. Let $B_p \leftarrow b_1 b_2$ where $b_1 \leftarrow x$ and $b_2 \leftarrow y$.
Let Q be the component of T_0 containing xy.
Let x be the root of each T_i ($i > 0$) and Q.
Go to Substep 4-1.

Substep 4-1. *(The Substep for searching of replaceable edges.) (See Subprogram 4-1 for details.)*
Outline and brief description. *For each $i \in \{0, 1, \ldots, k-1\}$, and for each $b_j \in B_p$, add all vertices of the directed path in T_i from $b_1 = x$ to b_j into B_p.*
For every new vertex b_j added into B_p, always check whether or not $b_j \in Q$. If NOT, then stop the iteration of Substep 4-1 and go to Substep 4-2. Repeat this Substep until no new vertex can be added into B_p, then

$$H \leftarrow H/B_p, \quad E_0 \leftarrow E_0|H, \quad p \leftarrow p + 1$$

and go to Step 3 (starting a new search for another level-k community). (Note, H/B_p is the graph obtained from H by identifying all vertices of B_p as a single vertex; $E_0|H$ is the set of edges of E_0 contained in the newly contracted graph H.) The new vertex of H created by contracting B_p is denoted by z_p.
Note, during the iteration, each new vertex b_i added into B_p is labeled with $m(b_i)$, $\epsilon(b_i)$ and $\lambda(b_i)$, which are to be used in creating the adjustment sequences S and I in Substep 4-2 in case that the spanning forest T_0 can be expanded. (For definitions of labels m, ϵ and λ, see Subprogram 4-2 for details.)

Substep 4-2 *(The Substep for expanding T_0.) (See Subprogram* 4-2 *for details.)*

Outline and brief description. *Create the adjustment sequences $S(e)$ and $I(e)$ based on the labels $m(b_i)$, $\epsilon(b_i)$ and $\lambda(b_i)$ generated in Substep* 4-1 *(See Subprogram* 4-1*).*

Follow the adjustment sequences S and I to adjust and expand the forests of $\mathcal{T}$.

And update the coverage $c_{\mathcal{T}}$ for the edge e_0.

Let $B_p \leftarrow \emptyset$ and erase labels $m(b_i)$, $\epsilon(b_i)$ and $\lambda(b_i)$ for all vertices of H.

Go to Step 3.

Step 5.

Let $\{v_1, \ldots, v_r\}$ be the vertex set of the resulting graph H (which has gone through a series of contractions in Step 4-1-2*). Each vertex v_i is a child of H in the hierarchical tree, some of which are single vertices, while others represent non-trivial level-k communities.*

If v_i is not a contracted vertex, then it is a child of H in the hierarchical tree, and no further action is needed for this vertex.

For a contracted vertex $v_i = z_p$, replace v_i with the corresponding community B_p and go to Step 6 *for further iteration. Note that it is possible that some vertex of B_p is also a contracted vertex $z_{p'}$. In this case, all vertices of $B_{p'}$ should be added into B_p. This procedure should be repeated for all possible contracted vertices in B_p.*

Step 6.

If $k = h$, output all B_i, each of which induces a level-h community of G.

If $k < h$, then repeat Step 1 *for $H \leftarrow G[B_i]$ for every i, $T_k \leftarrow T_0$, and $\mathcal{T} = \{T_1|H, T_2|H, \ldots, T_k|H\}$ which is a set of edge-disjoint spanning trees in H (as inputs of Step* 1 *for the next iteration of the algorithm at level $(k+1)$).*

5.3. SUBPROGRAM 4-1: THE SEARCH OF REPLACEABLE EDGES (SUBSTEP 4-1). The subprogram in this Section is the detailed Substep 4-1 of Algorithm 2.

5.3.1. *Notation and Description.* Some notation and labels used in this subprogram are introduced together with a brief discussion of the algorithm. The main frame of Algorithm 2 is basically the same as that of Algorithm 1. The major difference is the labeling system in Substep 4-1.

How is B_p constructed?

A vertex sequence $B_p = b_1 b_2 \cdots b_{D_e}$: B_p is the sequence consisting of vertices of H that are already selected for the p-th potential community. Note that, in Algorithm 1, it was already proved that the subgraph induced by all replaceable edges (at any stage) is connected. Hence, we only need to record the set of vertices incident with replaceable edges. That is why, in Algorithm 2, the subgraph induced by replaceable edges is recorded as a vertex set B_p. Furthermore, the ordering of the sequence indicates the order when a vertex is added in that subgraph, which will speed up the search of new members for B_p, and a possible expansion for the non-connected forest T_0.

The process starts with an extra edge $xy \in E(H) - \bigcup_{\mu=0}^{k-1} E(T_\mu)$, where each T_μ is a tree if $\mu \geq 1$ and T_0 is the non-connected spanning forest. And let Q be the component of T_0 containing the edge xy. Each tree of $\{T_0 \cap Q, T_1, \ldots, T_{k-1}\}$ is considered as a rooted tree with the root at x. It is evident that, for every vertex $v_j \in B_p$, all vertices along the unique directed path from the root x to v_j in a rooted tree T_β (or $T_0 \cap Q$) must be added into B_p.

For the sake of convenience in discussion, with an excusable abusive use of notation, whenever we mention a rooted tree T_β, it means T_β if $\beta \neq 0$, or, $T_0 \cap Q$ if $\beta = 0$ though T_0 is not connected.

Two auxiliary markers D_s and D_e are used along the sequence B_p: In order to avoid unnecessary searching along the sequence B_p, a "working zone" is defined

which starts at the vertex b_{D_s} and ends at b_{D_e}. The execution of Subsubstep 4-1-3 always starts at the vertex b_{D_s} instead of the first member of the sequence B_p.

An integer system M_k. In order to record the current working stage of the expansion (Substep 4-1) of the sequence B_p, and the "membership id" for each member of B_p, an integer system M_k is used for convenience. A non-negative integer μ are represented by an ordered integer pair (α, β) where $\mu = \alpha k + \beta$ with $0 \le \beta \le k-1$ and $\alpha \ge 0$. That is, $M_k = \{0, 1, 2, \dots\} \times \{0, 1, \dots, k-1\} = Z^+ \times Z_k$.

Label $m.B_p \mapsto M$, the "membership id" of each vertex of B_p. The label $m(b_i) = (\alpha, \beta)$ records when and how this vertex b_i is added into B_p. The second component β of $m(b_i)$ is the index of a tree T_β and indicates that the vertex b_i is added into B_p because it is contained in the circuit of $T_\beta + e$ where $e = b_{j_1}b_{j_2}$ is a replaceable edge ($b_{j_1}, b_{j_2} \in B_p$ with $j_1, j_2 < i$); The first component α of $m(b_i)$ indicates that, when b_i is added into B_p, the search (Substep 4-1) has already run α complete circles of the forest set $\{T_0, T_1, \dots, T_{k-1}\}$.

Current status m_C ($\in M_k$) is an indicator that indicates the current working status. At the initial situation, $m_C = (0, 0)$. When $m_C = (\alpha, \beta)$, the second component β indicates that the tree T_β is currently in the iteration of Subsubsteps 4-1-3, 4-1-4, 4-1-5 and 4-1-6.

"Stop signs" D_e and M_C. In Subsubstep 4-1-2, the inequality $m(b_{D_e}) \le m_C - (1, 0)$ indicates that the expansion of B_p has exhausted: during last k executions of Subsubstep 4-1-3 (one complete circle of the entire set $\mathcal{T}$ of spanning forests), the sequence B_p got no new expansion.

How is the forest T_0 expanded?

Although this is the purpose of Substep 4-2, the track records used in expansion was constructed based on the labelings from Substep 4-1. In order to fully understand the labeling system generated in this substep, we'd better to present the expansion process in Substep 4-2 before any explanation.

In a possible expansion (Substep 4-2) of the nonconnected forest, the adjustment follows a pair of edge sequence S and index sequence I as track records in Algorithm 1. In this improved version, we also have similar track records S and I. It is different from Algorithm 1, the sequences S and I of Algorithm 2 are not constructed until they are needed, and they are constructed based on some information (labels λ, ϵ, m, etc.) generated in Substep 4-1.

In Substep 4-2, an edge-sequence S and an index-sequence I are constructed as follows:

$$S = (b_{i_1}b_{i_1^*}), (b_{i_2}b_{i_2^*}), \dots, (b_{i_t}b_{i_t^*}), \quad \text{and} \quad I = \beta_{i_1}, \beta_{i_2}, \dots, \beta_{i_{t-1}},$$

where $i_\ell^* = \epsilon(b_{i_\ell})$, $i_{\ell+1} = \lambda(b_{i_\ell})$ and $m(b_{i_\ell}) = (\alpha_{i_\ell}, \beta_{i_\ell})$, for each $\ell = 1, \dots, t-1$. Here b_{i_1} is the very first vertex found by Substep 4-1 that is not in Q, and therefore, the edge $(b_{i_1}b_{i_1^*})$ is to be used to connect two components of T_0 (an expansion).

The nonconnected forest T_0 is expanded as follows: For each $\mu = (t-1), (t-2), \dots, 3, 2, 1$ (note, in reversed order):

$$T_{\beta_{i_\mu}} \leftarrow T_{\beta_{i_\mu}} + \left(b_{i_{\mu+1}}b_{i_{\mu+1}^*}\right) - \left(b_{i_\mu}b_{i_\mu^*}\right), \quad \text{and} \quad T_0 \leftarrow T_0 + \left(b_{i_1}b_{i_1^*}\right).$$

Note that, for each μ, the edge $(b_{i_\mu}b_{i_\mu^*})$ is contained in the circuit of $T_{\beta_{i_\mu}} + (b_{i_{\mu+1}}b_{i_{\mu+1}^*})$.

Now, we can see that the label ϵ indicates the another endvertex of a replaceable edge $b_i b_{\epsilon(b_i)}$, while the label λ indicates another edge $b_j b_{\epsilon(b_j)}$ that $b_i b_{\epsilon(b_i)}$ *is contained in the circuit of* $T_\beta + b_j b_{\epsilon(b_j)}$ *where* $\lambda(b_i) = j$. (Here $b_i b_{\epsilon(b_i)}$ and $b_j b_{\epsilon(b_j)}$ are two consecutive members in the sequence S in this order.)

Label $\epsilon : B_p \mapsto Z^+$: $\epsilon(b_i) = j$ indicates that the vertex b_i is added into B_p because b_j was already in B_p and b_j is one of the children of b_i in the rooted tree T_β (where $m(b_i) = (\alpha, \beta)$).

Label $\lambda : B_p \mapsto Z^+$: $\lambda(b_i) = h$ indicates that the vertex $b_h \in B_p$ is one of the descendants of b_i in T_β (where $m(b_i) = (\alpha, \beta)$) such that $b_i b_{\epsilon(b_i)}$ *is contained in the circuit of* $T_\beta + b_h b_{\epsilon(b_h)}$. Note that there is a directed path $P = b_i b_{\epsilon(b_i)} \cdots b_g$ in the rooted tree T_β with $b_g \in B_p$. Choose $\lambda(b_i) = h$ the minimum of g's for all such directed paths P in T_β. Then $b_{\epsilon(b_h)}$ is not a descendant of b_i in T_β and therefore, $b_h b_{\epsilon(b_h)}$ must be a replaceable edge not contained T_β.

$c : B_p \mapsto Z^+$, *a (temporary) carry-on label*: The label c is used for generating λ, which carries hierarchical information along the process so that

$$\lambda(b_i) = \min\{g : b_g \in D(T_\beta, b_i) \cap B_p\}$$

where $D(T_\beta, b_i)$ is the set of all descendants of b_i in the rooted tree T_β. It is only used in the current iteration of Subsubstep 4-1-3 and will be erased for next iteration.

5.3.2. *Subprogram 4-1* (The expansion of B_p).

Subsubstep 4-1-1.

$$D_s \leftarrow 2, \quad D_e \leftarrow 2,$$

$$m(x) = m(y) \leftarrow (0, 0),$$

and

$$m_C \leftarrow (0, 1).$$

Subsubstep 4-1-2. *(Check whether the expansion of* B_p *has ended.)*
If $m(b_{D_e}) \leq m_C - (1, 0)$ *then*

$$H \leftarrow H/B_p, \quad E_0 \leftarrow E_0|H, \quad p \leftarrow p + 1$$

and go to Step 3 *of Algorithm* 2 *(starting a new search for another level-k community). (Note,* H/B_p *is the graph obtained from* H *by identifying all vertices of* B_p *as a single vertex;* $E_0|H$ *is the set of edges of* E_0 *contained in the newly contracted graph* H*.) The new vertex of* H *created by contracting* B_p *is denoted by* z_p.
Otherwise, go to Substep 4-1-3 *and continue.*

Subsubstep 4-1-3.

$$i \leftarrow D_s,$$

and let $m_C = (\alpha_C, \beta_C)$.
(In the rooted tree T_{β_C}*, all ancestors of vertices in the working zone will be added into the sequence* B_p *in Subsubstep* 4-1-6.*)*

Subsubstep 4-1-4. *(Update* D_s *for the next iteration over the tree* $T_{D_{\beta_C+1}}$.*)*
If $m(b_i) < m_C - (0, k-2)$*, then* $D_s \leftarrow i$*, otherwise,* D_s *remains the same.*
Continue.

Subsubstep 4-1-5. *(Update $c(b_i)$ if it does not exist.)*
If $c(b_i)$ does not exist, then

$$c(b_i) \leftarrow i.$$

Otherwise, do nothing.
Continue.

Subsubstep 4-1-6. *(Adding vertices into B_p and labeling new vertices with λ)*
Find the parent v of b_i in the rooted tree $T_{\beta C}$.

Case 1. *$v \notin B_p$. (This vertex v is to be added into B_p.)*
Subcase 1-1. *If $v \notin Q$, then the spanning forest T_0 is now ready for expansion (and the expansion of B_p stops): go to Substep* 4-2 *of Algorithm* 2.
Subcase 1-2. *If $v \in Q$, then this new vertex v is to be added at the end of the sequence B_p and all labels are to be updated for this new vertex as follows:*

$$D_e \leftarrow D_e + 1, \quad b_{D_e} \leftarrow v, \quad \lambda(b_{D_e}) \leftarrow c(b_i), \quad c(b_{D_e}) \leftarrow c(b_i), \quad \epsilon(b_{D_e}) \leftarrow i.$$

And

$$i \leftarrow i + 1$$

and go to Substep 4-1-4 *(repeating for the next b_i in the sequence).*

Case 2. *$v \in B_p$, say $v = b_j$, and $j > i$.*

$$c(b_j) \leftarrow \min\{c(b_j), c(b_i)\}$$

if $c(b_j)$ exists; or

$$c(b_j) \leftarrow c(b_i)$$

if $c(b_j)$ does not exist.
And

$$i \leftarrow i + 1$$

and go to Subsubstep 4-1-4.

Case 3. *$v \in B_p$, say $v = b_j$, and $j < i$.*
Check whether b_i has reached the end of the working zone as follows.
If $i = D_e$, then

$$m_C \leftarrow m_C + (0, 1),$$

and erase all "carry-on" labels c, and go to Subsubstep 4-1-2.
If $i < D_e$, then

$$i \leftarrow i + 1$$

and go to Subsubstep 4-1-4.

5.3.3. *Remarks about Subprogram 4-1*
Fact. The labels m of vertices in the sequence B_p form an nondecreasing sequence. That is,

$$m(b_1) \leq m(b_2) \leq \cdots \leq m(b_{D_e}).$$

Fact. Whenever the Subsubstep 4-1-3 starts, the induced subgraph $G[B_p]$ is connected, and, furthermore, B_p induces a connected subtree of T_{β_C-1}.

Fact. During Substep 4-1-6, those vertices b_i with $m(b_i) \leq m_C - (1, 0)$ induces a connected subtree of T_{β_C}.

Fact. During Substep 4-1-6 Case 1, each new vertex added into B_p is along a directed path in T_{β_C} from the root $b_1 = x$ to a pre-existing vertex b_j of B_p, where,

$$m_C - (0, k-1) \leq m(b_j) \leq m_C - (0, 1).$$

Fact. A vertex $b_i \in B_p$ with $i \geq 3$ is added into B_p because it is in the circuit of $T_\beta + b_h b_{h'}$ where $h = \lambda(b_i)$, b_h is a descendant of b_i in the rooted tree T_β with the smallest subscript h, and $b_{h'} \in B_p$ is not a descendant of b_i in the tree T_β. Furthermore, $h' = \epsilon(b_h)$ and $b_h b_{h'}$ is an edge contained in a tree $T_{\beta'}$ where $m(b_h) = (\alpha', \beta')$.

5.4. SUBPROGRAM 4-2. EXPANSION OF $\mathcal{T}$ (FOR SUBSTEP 4-2)

Subprogram 4-2 *(Expansion for T_0).* At this stage, the inputs are (outputs of Substep 4-1-6, Subcase 1-1):
the current status $m_C = (\alpha_C, \beta_C)$,
and a vertex $v \notin Q$ that is the parent of $b_i \in B_p$ in the rooted tree T_{β_C}.

Subsubstep 4-2-1. Let

$$b_{D_e+1} \leftarrow v, \quad \lambda\big(b_{D_e+1}\big) \leftarrow c(b_i), \quad \epsilon\big(b_{D_e+1}\big) \leftarrow i.$$

Subsubstep 4-2-2. Set an edge-sequence S and an index-sequence I as follows:

$$S = \big(b_{i_1} b_{i_1^*}\big), \big(b_{i_2} b_{i_2^*}\big), \ldots, \big(b_{i_t} b_{i_t^*}\big);$$

$$I = \beta_{i_1}, \beta_{i_2}, \ldots, \beta_{i_{t-1}},$$

where

$$i_\ell^* = \epsilon\big(b_{i_\ell}\big), \quad i_{\ell+1} = \lambda\big(b_{i_\ell}\big)$$

$$m\big(b_{i_\ell}\big) = \big(\alpha_{i_\ell}, \beta_{i_\ell}\big)$$

for each $\ell = 1, \ldots, t-1$, and

$$b_{i_1} = b_{D_e+1}, \quad b_{i_1^*} = b_i,$$

and

$$\big(b_{i_t} b_{i_t^*}\big) = (b_2, b_1).$$

(That is, for each $\ell \leq t-1$, each $b_{i_\ell^*}$ is a child of b_{i_ℓ} in the rooted tree $T_{\beta_{i_\ell}}$, each $b_{i_{\ell+1}}$ is a descendant of b_{i_ℓ} in the rooted tree $T_{\beta_{i_\ell}}$ with the smallest subscript $i_{\ell+1}$ in B_p.)

Subsubstep 4-2-3. For each $\mu = (t-1), (t-2), \ldots, 3, 2, 1$ (note, in reversed order):

$$T_{\beta_{i_\mu}} \leftarrow T_{\beta_{i_\mu}} + \big(b_{i_{\mu+1}} b_{i_{\mu+1}^*}\big) - \big(b_{i_\mu} b_{i_\mu^*}\big)$$

and

$$T_0 \leftarrow T_0 + \left(b_{i_1} b_{i_1^*}\right).$$

Subsubstep 4-2-4. Update the coverage: $c_{\mathcal{F}} \leftarrow c_{\mathcal{F}}(e) + 1$ if $e = e_0 = xy$ (delete e_0 from E_0 if $c_{\mathcal{F}}(e_0) = w(e_0)$), and $c_{\mathcal{F}} \leftarrow c_{\mathcal{F}}(e)$ otherwise. Erase labels B_p, D_s, D_e, m, m_C, λ, ϵ, c and go back to Step 3.

5.4.1. *Remarks about Subprogram* 4-2 In Subsubstep 4-2-2 (the construction of adjustment sequences S and I),

$$i_1 > i_1^* \geq i_2 > i_2^* \geq \cdots \geq i_t = 2 > i_t^* = 1.$$

And $(b_{i_\mu} b_{i_\mu^*}) \in T_{\beta_{i_\mu}}$ for each $\mu = 1, \ldots, t-1$, and is an edge contained in $T_{\beta_{i_\mu}} + (b_{i_{\mu+1}} b_{i_{\mu+1}^*})$.

Similar to Algorithm 1, we also need to show that each $T_{\beta_{i_\mu}}$ in Subsubstep 4-2-3 remains a tree/forest.

Let

$$S = \left(b_{i_1} b_{i_1^*}\right), \left(b_{i_2} b_{i_2^*}\right), \ldots, \left(b_{i_t} b_{i_t^*}\right);$$
$$I = \beta_{i_1}, \beta_{i_2}, \ldots, \beta_{i_{t-1}}$$

as obtained in Subsubstep 4-2-2.

For a given $\mu \in \{0, \ldots, k-1\}$, let $T^{(0)} \leftarrow T_\mu$ and let

$$\beta_{i_{v_1}} = \beta_{i_{v_2}} = \cdots = \beta_{i_{v_q}} = \mu$$

with

$$i_{v_1} > i_{v_2} > \cdots > i_{v_q}.$$

Follow the algorithm (Subsubstep 4-2-3), let

$$e_j \leftarrow \left(b_{i_{v_j+1}} b_{i_{v_j+1}^*}\right), \quad f_j \leftarrow \left(b_{i_{v_j}} b_{i_{v_j}^*}\right),$$

and

$$T^{(j)} \leftarrow T^{(j-1)} + e_j - f_j$$

for each $j = 1, \ldots, q$. Then, applying Lemma 4.3 to the following alternative sequence

$$T^{(0)},\ (e_1, f_1),\ T^{(1)},\ (e_2, f_2),\ T^{(2)}, \ldots, (e_q, f_q),\ T^{(q)},$$

we can see that the output $T^{(q)} = T_\mu$ of Subsubstep 4-2-3 is a tree/forest.

5.5. COMPLEXITY Let $|V(G)| = n$ and $|E(G)| = m$. At the kth level, Step 1 is iterated for all nontrivial outputs, $H_1, \ldots, H_{a_k}$, from Step 6 of the previous iteration (at the $(k-1)$-st level). Let $m_k = \sum_{i=1}^{a_k} |E(H_i)|$ and $n_k = \sum_{i=1}^{a_k} |V(H_i)|$.

1. *The cost of Step 3 at a given level k for a given input H_i ($i \in \{1, \ldots, a_k\}$):* The process starts with a selected edge $e_0 = xy \in E_0$ and ends when either the forest T_0 is expanded (at Subsubstep 4-2-4) or a level-k community is produced (at Subsubstep 4-1-2).

1-1. *The cost of one execution of Substep 4-1 (Subprogram 4-1):* During the iteration of Subprogram 4-1, each b_i is processed only when it appears in the

working zone. Hence, each b_i is processed at most $(k-1)$ times (see Subsubstep 4-1-4). The processing is along the sequence $B_p = b_1 b_2 \cdots b_{D_e}$. So, the cost is at most

$$O(k|B_p|) \leq O(k|V(H_i)|).$$

1-2. *The cost of one execution of Substep 4-2 (Subprogram 4-2):* Subsubstep 4-2-2 creates the adjustment sequences S and I from the sequence B_p based on the existing labels λ and ϵ. Since

$$i_1 > i_1^* \geq i_2 > i_2^* \geq \cdots \geq i_t = 2 > i_t^* = 1,$$

the construction of S and I costs at most $O(|B_p|) \leq O(|V(H_i)|)$. And the expansion of $\mathcal{T}$ in Subsubstep 4-2-3 costs at most

$$O(t) \leq O(|B_p|) \leq O(|V(H_i)|).$$

Since Substep 4-2 starts after Substep 4-1 (into Substep 4-2 from the Subcase 1-1 of Subsubstep 4-1-6), together with the cost of iterations of Substep 4-1, the total cost is

$$O(|B_p|) + O(t) + O(k|B_p|) = O(k|B_p|) \leq O(k|V(H_i)|).$$

1-3. Each iteration of Step 3 starts by selecting an edge e_0 from E_0. Those unsaturated edges e_0 can be classified as one of two types. An edge $e_0 \in E_0$ is of Type 1 if the iteration ends with an expansion of T_0, or Type 2 if the iteration ends with an output of a potential level-k community B_p. The number of Type 1 edges is less than $|V(H_i)| - 1$ since the spanning forest T_0 (in H_i) cannot have more than $|V(H_i)| - 1$ edges. The number of Type 2 edges is at most $|V(H_i)| - 1$ as well since it is easy to show that the subgraph of H_i induced by Type 2 edges must be acyclic because of the contraction of B_p in H_i (See Subsubstep 4-1-2.)

Therefore, the total cost is at most

$$O(|V(H_i)|)O(|V(H_i)|k) = O(|V(H_i)|^2 k).$$

2. The cost of Step 1 at a given level k, which is also the total cost of Step 3 for all H_i's $(1 \leq i \leq a_k)$, is at most

$$\sum_{i=1}^{a_k} O(|V(H_i)|^2 k) \leq O\left(n_k^2 k\right). \tag{10}$$

3. The total cost for all k's $(1 \leq k \leq h)$ is at most

$$\sum_{k=1}^{h} \sum_{i=1}^{a_k} O(|V(H_i)|^2 k) \leq \sum_{k=1}^{h} O\left(n_k^2 k\right) \leq O\left(n^2 h^2\right). \tag{11}$$

This is the worst-case complexity for Problem 1.4.

4. Since each H_i is a level-$(k-1)$ community, by the definition of dynamic density, we have that

$$k - 1 < \frac{|E(H_i)|}{|V(H_i)| - 1} \leq \frac{2|E(H_i)|}{|V(H_i)|} \tag{12}$$

(note that each input H_i is nontrivial). That is,

$$O(|V(H_i)|k) \leq O(|E(H_i)|). \tag{13}$$

Hence, the estimation of the worst-case complexity in (11) can be restated as follows:

$$\begin{aligned}\sum_{k=1}^{h}\sum_{i=1}^{a_k} O(|V(H_i)|^2 k) &\leq \sum_{k=1}^{h}\sum_{i=1}^{a_k} O(|V(H_i)||E(H_i)|) \\ &\leq \sum_{k=1}^{h} O(n_k m_k) \leq O(nmh). \end{aligned}\tag{14}$$

5. If one would like to construct a complete hierarchical tree for G (with all vertices of G as leaves of the tree), then one must go all the way down by exhausting all nontrivial communities at every possible level (without a fixed input h). Let h_m be a maximum level in which the algorithm produces a nontrivial community. It is easy to see that $h_m \leq \Delta$ where Δ is the maximum degree of G. By (14), the worst-case complexity is upper bounded by $O(nm\Delta)$.

6. *Other Applications*

6.1. EDGE-DISJOINT SPANNING TREES For the problem of finding a maximum set of edge-disjoint spanning trees in a graph G (Problem 1.7), the algorithm can be slightly modified as follows,

The process is to be stopped if either (in Step 3) $E_0 = \emptyset$ or (in Step 5) more than one B_p are produced.

The worst-case complexity of Problem 1.7 is also $O(n^2h^2)$ where h is the dynamic density of G.

Similar to (12), $h \leq \frac{|E(G)|}{|V(G)|-1} \leq \frac{2m}{n}$ and therefore, $hn \leq 2m$. So, the worst-case complexity of determining the dynamic density of G is also upper bounded by $O(m^2)$.

6.2. MOTIF Motif, roughly speaking, is a subgraph H very close to a clique. That is, its minimum or average degree is close to $|V(H)| - 1$.

Definition 6.1. Let σ be a positive real number (choice by users). A graph H is called a σ-dynamic-motif if

$$\frac{D(H)}{\overline{w}(H)} \geq \sigma|V(H)|,$$

where $D(H)$ is the dynamic density of H and $\overline{w}(H)$ is the average weight of edges in H.

The following is a modified version of Step 6, Algorithm 2 for searching σ-dynamic-motifs.

Step 6. *Does $V(H)$ contain some contracted vertex z_p? If yes, go to Substep* 6-1; *Otherwise, go to Substep* 6-2.

Note that each contracted vertex in the resulted graph H corresponds to a contracted level-k community.

Substep 6-1. *For each output B_p of Step* 5*, if*

$$|B_p| \leq \frac{D(H)}{\sigma \overline{w}(H)},$$

store the data of B_p for final output; otherwise, repeat Step 1 *for the induced subgraph $G[B_p]$ as follows:*

$$H \leftarrow G[B_p], \quad T_k \leftarrow T_0,$$

and

$$\mathcal{T} = \{T_1|H, T_2|H, \ldots, T_k|H\}$$

which is a set of edge-disjoint spanning trees in H (as inputs of Step 1 *for the next iteration of the algorithm at level $(k+1)$).*

Substep 6-2. *Store the data of H (with some special label) for final output. In this case, although H is not a level-$(k-1)$ community, it requires special attention for further data analysis (that is why a label is needed for this kind of non-motif output).*

Note that this case happens only when E_0 becomes empty in Step 3 *while there is no level-k community B_p produced by Substep 4-1 yet (i.e., p remains* $= 1$*).*

7. *Remarks*

In this section, we are to present some results in graph theory, each of which is an immediate corollary of Algorithm 1 and its corresponding theorems/propositions. These results are expected to be useful in graph theory and related subjects.

A partial ordering among all spanning forests of G is defined as follows:

Definition 7.1. Let T_α and T_β be two spanning forests of G. $T_\alpha \succeq_{\mathcal{T}} T_\beta$ if, for every component C_β of T_β, there is a component C_α of T_α such that $V(C_\beta) \subseteq V(C_\alpha)$.

For a graph G, let

$$\gamma_k(G) = \max_{\forall \mathcal{T}} \left\{ \sum_{T_\mu \in \mathcal{T}} |E(T_\mu)| \right\}$$

where the maximum is taken over all possible sets of k edge-disjoint spanning forests $\mathcal{T} = \{T_1, \ldots, T_k\}$ of G.

COROLLARY 7.2. *Let G be a graph with dynamic density $d(G)$ and k be a positive integer. Then G has a set of edge-disjoint spanning forests $\{T_1, \ldots, T_k\}$ such that*

(1)

$$|\sum_{\mu=1}^{k} |E(T_\mu)| = \gamma_k(G);$$

(2)

$$T_1 \succeq_{\mathcal{T}} T_2 \succeq_{\mathcal{T}} \cdots \succeq_{\mathcal{T}} T_k;$$

(3) *T_μ is connected (therefore, a spanning tree) for each μ:* $1 \leq \mu \leq \min\{d(G), k\}$*;*

(4) *For every edge* $e \in E(G) - \bigcup_{\mu=1}^{k} E(T_\mu)$*, the edge* e *is contained in a level-*k *community* C*. Furthermore,* $V(C)$ *is a proper subset of* $V(G)$ *if* T_k *is not a spanning tree.*

Algorithm 1 and Algorithm 2 provides a practical approach in the searching of edge-disjoint spanning forests in a graph G with maximum total size. That is, a level-k community serves as a *contractible configuration* for this graph theory property.

COROLLARY 7.3. *Let* H *be a level-*k *community. Then, for any supgraph* G *of* H*,*

(1) *the graph* G *contains* k *edge-disjoint spanning trees if and only if* G/H *contains* k *edge-disjoint spanning trees;*
(2) *a set of edge-disjoint forests in* G/H *of total size* $\gamma_k(G/H)$ *can be extended to a set of edge-disjoint forests in* G *of total size* $\gamma_k(G)$*.*

Note that a set of edge-disjoint forests $\{T'_1, \ldots, T'_k\}$ in G/H is extendable to a set of edge-disjoint forests $\{T_1, \ldots, T_k\}$ in G if $T_i/H = T'_i$ and $T_i \cap H$ is a spanning tree in H.

COROLLARY 7.4. *Let* $\{T_1, \ldots, T_k\}$ *be a set of edge-disjoint foresters in* G *of total size* $\gamma_k(G)$*. If* $E(G) - \cup_{i=1}^{k} E(T_i) \neq \emptyset$*, then* G *contains a nontrivial level-*k *community* C*. Furthermore, if one of* $\{T_1, \ldots, T_k\}$ *is not a spanning tree, then* $V(C)$ *is a proper subset of* $V(G)$*.*

REFERENCES

ANTHONISSE, J. M. 1971. Technical Report BN 9-71. Stichting Mathematisch Centrum, Amsterdam, The Netherlands.

BARAHONA, F. 1995. Packing spanning trees. *Math. Operat. Res. 20*, 1, 104–115.

BARAHONA, F. 2004. Network reinforcement. *IBM Research,* preprint. http://www.research.ibm.com/people/b/barahon/publications.html.

BERKHIN, P. 2002. Survey of Clustering Data Mining Techniques. Accrue Software.

BEZDEK, J. C. 1981. *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York.

BRADLEY, P. S., AND FAYYAD, U. M. 1998. Refining initial points for K-means clustering. In *Proceedings of the 15th International Conference on Machine Learning* (San Francisco, CA). pp. 91–99.

COVER, T., AND HART, P. 1967. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory 13*, 21–27.

CUNNINGHAM, W. H. 1984. Testing membership in matroid polyhedra. *J. Combin. Theory, Ser. B 36*, 2, 161–188.

CUNNINGHAM, W. H. 1985. Optimal attack and reinforcement of a network. *J. ACM 32*, 3, 549–561.

D'ANDRADE, R. 1978. U-statistic hierarchical clustering. *Psychometrika 4*, 58–67.

DING, C., AND HE, X. 2004. K-nearest-neighbor consistency in data clustering: Incorporating local information into global optimization. In *Proceedings of the 2004 ACM Symposium on Applied Computing*. ACM, New York, pp. 584–589.

DUNN, J. C. 1973. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *J. Cybernet. 3*, 32–57.

FRALEY, C., AND RAFTERY, A. E. 1998. How many clusters? Which clustering method? Answers via model-based cluster analysis. *Comput. J. 41*, 8, 578–588.

FREEMAN, L. C. 1977. A set of measures of centrality based on betweenness. *Sociometry 40*, 35–41.

GABOW, H. N. 1995a. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci. 50*, 2, 259–273.

GABOW, H. N. 1995b. Algorithms for graphic polymatroids and parametric s-sets. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms.* ACM, New York, 88–97.

GABOW, H. N., AND WESTERMANN, H. H. 1992. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica 7*, 5–6, 465–497.

GABOW, H. N. 1998. Algorithms for graphic polymatroids and parametric $\overline{s}$-sets. *J. Algor. 26*, 1, 48–86.

GIRVAN, M., AND NEWMAN, M. E. J. 2002. Community structure in social and biological networks, *Proc. Natl. Acad. Sci. 99*, 8271–8276.

GOLDBERG, A. V., AND TARJAN, R. E. 1998. A new approach to the maximum-flow problem. *J. ACM 35*, 4, 921–940.

GUSFIELD, D. 1991. Computing the strength of a graph. *SIAM J. Comput. 20*, 4, 639–654.

HAN, J., AND KAMBER, M. 2000. *Data Mining: Concepts and Techniques,* The Morgan-Kaufmann Series in Data Management Systems. Jim Gray, Series Ed. Morgan-Kaufmann, San Francisco, CA.

HARTIGAN, J. A. 1975. *Clustering Algorithms*, Wiley, New York.

HOPPNER, F., KLAWONN, F., KRUSE, R., AND RUNKLER. T. 1999. Fuzzy cluster analysis, methods for classification, data analysis and image recognition. Wiley, New York.

JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. 1999. Data clustering: A review. *ACM Comput. Surv. 31*, 3, 264–323.

JOHNSON, S. C. 1967. Hierarchical Clustering Schemes. *Psychometrika 2*, 241–254.

LAMPINEN, T., KOIVISTO, H., AND HONKANEN, T. 2002. Profiling network applications with fuzzy C-means clustering and self-organising map. In *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery* (Nov.).

LUKASHIN, A.V., AND FUCHS, R. 2001. Analysis of temporal gene expression profiles: Clustering by simulated annealing and determining the optimal number of clusters. *Bioinformatics 17*, 5, 405–414.

MACQUEEN, J. B. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability,* vol. 1. University of California Press, Berkeley, CA, 281–297.

MASSART, D. L., AND KAUFMAN, L. 1983. *Interpretation of Analytical Chemical Data by the Use of Cluster Analysis* . Wiley Interscience, New York, ISBN: 0471078611.

MOORE, A. W. 2001. K-Means and Hierarchical Clustering, Carnegie Mellon University, Nov.

NASH-WILLIAMS, C. ST. J. A. 1961. Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc. 36*, 445–450.

PELLEG, D., AND MOORE, A. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Conference on Knowledge Discovery in Data, Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, 277–281.

RADICCHI, F., CASTELLANO, C., CECCONI, F., LORETO, V., AND PARISI, D. 2004. Defining and identifying communities in networks. *Proc. Nat. Acad. Sci. 101*, 9, 2658–2663.

ROBERTS, F. S., AND TESMAN, B. 2004. *Applied Combinatorics* (2nd Ed.). Prentice-Hall, Englewood Cliffs, NJ.

ROSKIND, J., AND TARJAN, R. E. 1985. A note on finding minimum-cost edge-disjoint spanning trees. *Math. Oper. Res. 10*, 4, 701–708.

SAS INSTITUTE. INC. Introduction to Clustering Procedures. Chap. 8 of *SAS/STAT User's Guide*. (SAS OnlineDoc™: Version 8).

SEIDMAN, S. B. 1983. Network structure and minimum degree. *Social Netw. 5*, 269–287.

STEINBACH, M., KARYPIS, G., AND KUMAR, V. 2000. A comparison of document clustering techniques. *TextMining Workshop, KDD*.

STEPHEN, S. P. 1994. How to explain hierarchical clustering. *Connections 17*, 2, 78–80.

TUTTE, W. T. 1961. On the problem of decompositing a graph into *n* connected factors. *J. London Math. Soc. 36*, 221–230.

WASSERMAN, S., AND FAUST, K. 1994. *Social Network Analysis*. Cambridge University Press, Cambridge, UK.

WU, F., AND HUBERMAN, B. A. 2004. Finding communities in linear time: A physics approach. *Europ. Phys. J. B - Condensed Matter 38*, 2, 331–338.

XU, Y., OLMAN, V., AND XU, D. 2002. Clustering gene expression data using graph-theoretic approach: An application of minimum spanning trees. *Bioinformatics 18*, 536–545.

ZHANG, C. Q. 1997. *Integer Flows and Cycle Covers of Graphs*. Marcel Dekker Inc., New York.

ZHANG, C. Q., AND OU, Y. 2006. *Method for Data Clustering and Classification by a Graph Theory Model – Network Partition into High Density Subgraphs.* US Patent (Pending 2006: # 11/416,766; Provisional 2005: # 60/677,655)).

RECEIVED JANUARY 2006; REVISED AUGUST 2006; ACCEPTED APRIL 2007