The VLDB Journal (2020) 29:353–392
https://doi.org/10.1007/s00778-019-00556-x

SPECIAL ISSUE PAPER

A survey of community search over big graphs

Yixiang Fang[1,2] · Xin Huang[3] · Lu Qin[4] · Ying Zhang[4] · Wenjie Zhang[1] · Reynold Cheng[5] · Xuemin Lin[1,2]

Received: 31 December 2018 / Revised: 15 June 2019 / Accepted: 5 July 2019 / Published online: 20 July 2019
© Springer-Verlag GmbH Germany, part of Springer Nature 2019

Abstract
With the rapid development of information technologies, various big graphs are prevalent in many real applications (e.g., social media and knowledge bases). An important component of these graphs is the network community. Essentially, a community is a group of vertices which are densely connected internally. Community retrieval can be used in many real applications, such as event organization, friend recommendation, and so on. Consequently, how to efficiently find high-quality communities from big graphs is an important research topic in the era of big data. Recently, a large group of research works, called community search, have been proposed. They aim to provide efficient solutions for searching high-quality communities from large networks in real time. Nevertheless, these works focus on different types of graphs and formulate communities in different manners, and thus, it is desirable to have a comprehensive review of these works. In this survey, we conduct a thorough review of existing community search works. Moreover, we analyze and compare the quality of communities under their models, and the performance of different solutions. Furthermore, we point out new research directions. This survey does not only help researchers to have better understanding of existing community search solutions, but also provides practitioners a better judgment on choosing the proper solutions.

Keywords Community search · Community retrieval · Big graph · Graph queries · Online queries

1 Introduction

With the rapid development of information technologies, various big graphs are prevalent in many real applications (e.g., social media and knowledge bases). An important component of these graphs is the network community. Essentially, a community is a group of vertices which are densely connected internally. For example, in Facebook, communities consist of users that are with strong friendship [3]; on the World Wide Web, communities contain web sites which share similar topics [22]; in protein–protein interaction networks [151] and metabolic networks [82], communities correspond to functionality modules. Retrieving communities from a network is a fundamental problem in network science, and it can be applied to many real-life applications. Here are some typical applications, to name a few:

– *Event organization* A social event (e.g., a party or a conference) often involves a group of users and its organization can benefit from communities. For example, to hold a cocktail part, a user can find his community, i.e., a group of researchers, each of which is well acquainted.

✉ Yixiang Fang
yixiang.fang@unsw.edu.au

Xin Huang
xinhuang@comp.hkbu.edu.hk

Lu Qin
lu.qin@uts.edu.au

Ying Zhang
ying.zhang@uts.eud.au

Wenjie Zhang
zhangw@cse.unsw.edu.au

Reynold Cheng
ckcheng@cs.hku.hk

Xuemin Lin
lxue@cse.unsw.edu.au

1 University of New South Wales, Sydney, Australia

2 Zhejiang Lab, Hangzhou, China

3 Hong Kong Baptist University, Kowloon Tong, Hong Kong

4 The University of Technology Sydney, Sydney, Australia

5 The University of Hong Kong, Pok Fu Lam, Hong Kong

– *Friend recommendation* Many social media platforms (e.g., Facebook) often maintain a friendship network. To suggest candidate friends to a specific user u, intuitively we can recommend u those who are in u's community but are not yet u's friends.
– *Protein complex identification* In biology, proteins interact with each other and a gene is often regulated by a set of proteins. To study a gene, a biologist may focus on a set of proteins that highly interact with each other, which is a community of proteins.
– *Advertisement in e-commence* Users of the same community often share similar interests. To push advertisements for a user u, we may find her community first and then select advertisements that are checked by members of her community.

Owing to the importance of communities, how to effectively and efficiently find communities from large graphs is an important research topic in the era of big data. With a careful observation on these applications, we identify a list of factors that the community retrieval solutions should satisfy:

– *High efficiency* For many real applications (e.g., event organization), the communities often need to be retrieved in real time, based on query requests. Thus, the community retrieval solutions should be able to respond in real time.
– *High scalability* Nowadays, many real networks contain millions or billions of vertices. As a result, the solutions should be scalable to real big graphs.
– *High personalization* In practice, for large networks, people usually are interested in communities of some specific users, rather than all the users. Thus, the solutions should allow users to specify query vertices. Moreover, some personalized requirements on structures (and attributes) could be imposed.
– *High quality* The vertices in the communities retrieved should be cohesively linked. Moreover, the communities should be easy for interpretation.
– *Support for dynamic graphs* Since real networks often involve as the time goes on, the solutions should be able to adapt for the dynamic changes easily.

Toward the goals above, recently a large group of research works, called community search (CS), have been proposed [103]. Generally, the goal of CS is to search high-quality communities in an online manner, based on a query request. Specifically, given a vertex q of a graph G, it aims to find a community, or a dense subgraph, which contains q and satisfies the properties: (1) *connectivity*, i.e., vertices in the community are connected; and (2) *cohesiveness*, i.e., vertices in the community are intensively linked to each other w.r.t. a particular goodness metric [15,45,46,175,175]. The metric is often defined by using some classical subgraph cohesiveness metrics such as:

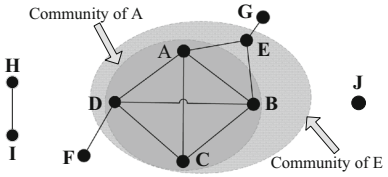


Fig. 1 An example of community search

– k-core The k-core [17,170] is the largest subgraph of G, in which each vertex's degree is at least k within the subgraph.
– k-truss The k-truss [41,98] is the largest subgraph of G in which every edge is contained in at least $(k-2)$ triangles within the subgraph.
– k-clique A k-clique [2] is a set of k vertices of G such that each pair of vertices has an edge.
– k-ECC A k-ECC (k-edge-connected component) [76] is a subgraph of G such that after removing any $k-1$ edges, it is still connected.

Let us illustrate CS by an example. Consider the graph with ten vertices in Fig. 1, and CS solutions [15,46,175], which are based on the k-core model. Let $q = A$. Then, the induced subgraph of vertices $\{A, B, C, D\}$ will be returned as the community. Note that the subgraph forms a k-core with $k = 3$, since each vertex's degree is 3 within the subgraph, and it is also the core attaining the maximum value of k.

In the literature, there is a highly related group of research works, called community detection (CD) [44,110,154,156,158]. Generally, it has similar goals with CS, but there are three key differences: (1) The problem definitions are different. CS aims to search communities regarding a set of query vertices and some query parameters, while CD often detects all communities in the graph. (2) The criteria of defining communities are different. In CS, the criteria of defining communities are based on query parameters given by the users. In other words, communities are retrieved depending on user-defined parameters. In contrast, CD methods often use the same global criterion to detect communities by partitioning the entire graph. For example, in Fig. 1, if $q = A$, CS solutions [46,175] will find the community $\{A, B, C, D\}$, and if $q = E$, they will find the community $\{A, B, C, D, E\}$. In contrast, if using a CD method (e.g., the spectral clustering [182]) with setting the number of communities to 3, we will obtain three communities, each of which forms a connectivity component, where B and E are in the same community. (3) The algorithms are different. As shown in existing studies, CS solutions can search communities efficiently in an online

manner, while CD solutions are often time-consuming and unscalable to big graphs. Moreover, CS queries can often be supported by indexes and handle dynamic graphs easily. Thus, compared to CD solutions, CS solutions can better satisfy factors aforementioned.

Although there are many CS solutions, they deal with different types of graphs and formulate communities in different manners. Meanwhile, there is a lack of systematic survey of CS solutions. Thus, it is desirable to organize these works and understand how well they perform in terms of efficiency and quality. To this end, in this paper we will provide a thorough review of these works. We will also compare different CS solutions so that readers can better understand the state of the art and point out directions for future study.

As shown in Table 1, we classify CS solutions into five categories such that solutions in each category (except the last category) adopt the same structure cohesiveness metric. Moreover, for works in each category, we further partition them into two groups, where the first group focuses on simple graphs while the second group targets attributed graphs. Note that the IDs of CS problems are also included in the brackets of Table 1. For simple graphs, CS solutions search communities purely based on link information, while for attributed graphs, CS solutions often consider both links and attributes. We remark that these cohesiveness metrics are orthogonal to graph types. This implies that if a metric has not been studied for a particular type of graphs, then it is a possible future research direction to study CS by applying the metric on this type of graphs.

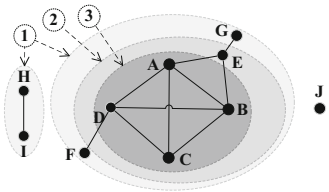
In summary, our main contributions are as follows:

- First, we provide a systematic classification of studies on CS. Specifically, we classify these studies according to the community cohesiveness metrics. For each class of works, we review the representative studies on different types of graphs.
- Second, we perform a thorough analysis and comparison of different community cohesiveness metrics. Moreover, we analyze and compare CS solutions on simple graphs and attributed graphs.
- Third, we offer insightful suggestions for future study on CS. This may give researchers new to CS an understanding of the recent development of CS, as well as a good starting point to work in this field.

The rest of this paper is organized as follows: In Sect. 2, we introduce and discuss community cohesiveness metrics. In Sects. 3, 4, 5, 6, and 7 , we extensively discuss CS solutions in each category. We also present two CS systems in Sect. 8. We review the related work in Sect. 10. Finally, we present a list of future topics in Sect. 11 and conclude in Sect. 12.

Table 1 Classification of works of community search ("P." means Problem)

Metric	Simple graphs	Attributed graphs				
		Keyword	Location	Temporal	Influence (weight)	Profile
k-core	[15,46,66,175] (P. 1, 2,3, 4,5)	[58,61] (P. 6)	[60,65,185,221,221] (P.7, 8,9)	[129] (P.10)	[21,30,126–128,215] (P. 12, 13)	[31] (P. 14)
k-truss	[6,98,101] (P.15, 16)	[102] (P. 17)	–	–	[216] (P. 18)	–
k-clique	[45,187,195,205] (P. 19, 20, 21, 22)	–	–	[125] (P. 23)	–	–
k-ECC	[6,98,101] (P.24, 25, 26)	–	–	–	–	–
Others	Local modularity: [40,136]; query biased density: [190]; PageRank: [9,114] (P. 27); neighbors: [142]					

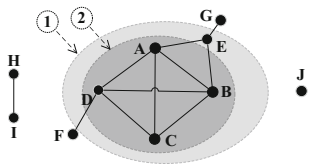


(a) k-cores

Core number	Vertices
0	J
1	F, G, H, I
2	E
3	A, B, C, D

(b) Core numbers

Fig. 2 Illustration of k-core



(a) k-trusses

Truss number	Edges
4	(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)
3	(A, E), (B, E)
2	(D, F), (E, G), (H, I)

(b) Trusses

Fig. 3 Illustration of k-truss

2 Preliminaries

In this section, we first formally introduce the commonly used community cohesiveness metrics and then compare their cohesiveness and computational efficiency.

2.1 Cohesiveness metrics

For ease of exposition, we consider a simple undirected graph $G(V, E)$, with vertex set V and edge set E. Let n and m be the corresponding sizes of V and E. The degree of a vertex v of G is denoted by $\deg_G(v)$.

•*k-core*. We introduce its formal definition as follows:

Definition 1 (*k-core* [17,170]) Given an integer k ($k \geq 0$), the k-core of G, denoted by H_k, is the largest subgraph of G, such that $\forall v \in H_k$, $\deg_{H_k}(v) \geq k$.

We say that H_k has an order of k. Notice that H_k may not be a connected graph [17]. Observe that k-cores are "nested" [17]: given two positive integers i and j, if $i < j$, then $H_j \subseteq H_i$.

Example 1 In Fig. 2a, the subgraph of $\{A, B, C, D\}$ is the 3-core. The 1-core has vertices $\{A, B, C, D, E, F, G, H, I\}$ and is composed of two connected components: $\{A, B, C, D, E, F, G\}$ and $\{H, I\}$. The number k in each circle represents the k-core contained in that ellipse. Clearly, $H_3 \subset H_2 \subset H_1$.

Definition 2 (*core number*) Given a vertex $v \in V$, its core number, denoted by $\text{core}_G[v]$, is the highest order of a k-core that contains v.

A list of core numbers and their respective vertices for Example 1 are shown in Fig. 2b. Equivalently, the k-core is the induced subgraph of vertices, whose core numbers are at least k.

•*k-truss* The k-truss is defined based on triangles. Specifically, a triangle in G is a cycle of length 3. Let $u, v, w \in V$ be the three vertices on the cycle. Then, we denote this triangle by $\triangle_{uvw}$.

Definition 3 (*support*) Given a graph $G(V, E)$, the support of an edge $(u, v) \in E$, denoted by $\sup(e, G)$, is defined as $|\{\triangle_{uvw} : u, v, w \in V\}|$.

Definition 4 (*k-truss* [41,166,212]) Given a graph G, the k-truss of G, denoted by J_k, is the largest subgraph of G, such that $\forall e \in J_k$, $\sup(e, J_k) \geq (k-2)$.

Example 2 Let us reconsider the graph G in Fig. 2a. The induced subgraph of G by vertex set $\{A, B, C, D\}$ is the 4-truss. The 3-truss has vertices $\{A, B, C, D, E\}$. The number k in each circle represents the k-truss contained in that ellipse.

Definition 5 (*truss number* [184]) Given a graph G, the truss number (trussness) of an edge $e \in G$, denoted by $\tau(e)$, is the largest k such that there is a k-truss containing e.

A list of truss numbers and their respective edges for Example 2 are shown in Fig. 3b. Equivalently, the k-truss is the induced subgraph of edges, whose truss numbers are at least k. Similar to k-core, a k-truss may contain multiple connected components.

• k-clique It is defined as follows:

Definition 6 (*k-clique* [2,151]) A k-clique is a complete graph with k vertices where there is an edge between every pair of vertices.

Example 3 In the graph in Fig. 2a, the subgraph of $\{A, B, C, D\}$ is a 4-clique and any three vertices of them form a 3-clique (i.e., triangle). The subgraph of $\{A, B, E\}$ is also a 3-clique. Any edge is a 2-clique.

•*k-ECC*. We first introduce some related concepts.

Definition 7 (*edge connectivity* [76,95]) Given a graph $G(V, E)$ and two vertices $u, v \in V$, the connectivity $\lambda(u, v)$ between u and v is the minimum number of edges whose removal disconnects u and v.

Definition 8 (*graph connectivity* [76,95]) Given a graph $G(V, E)$, the connectivity of the graph G, $\lambda(G) = \min_{u,v \in V} \lambda(u, v)$, is the minimum connectivity between any two distinct vertices in G, i.e., the minimum number of edges whose removal disconnects G.

Definition 9 (k-*ECC* [76,95]) Given a graph $G(V, E)$, a subgraph G' of G is a k-edge-connected component, or k-ECC, if $\lambda(G') \geq k$ and the connectivity of any super-graph of G' in G is less than k.

Example 4 In the graph in Fig. 2a, the subgraph of $\{A, B, C, D\}$ is the 3-ECC, because for any pair of vertices in it, to disconnect them, we need to remove at least 3 edges. The 2-ECC has vertices $\{A, B, C, D, E\}$. There are two 1-ECCs, which contain vertices $\{H, I\}$ and $\{A, \ldots, G\}$, respectively.

2.2 Cohesiveness and computational efficiency

Generally, in terms of structure cohesiveness, k-clique is the most cohesive one, since each vertex of a k-clique is linked to all the other $(k-1)$ vertices. For each connected component of the k-truss, it is more cohesive than a k-ECC. This is because k-truss is more restrictive as it is defined based on triangle, which is a local concept, whereas k-ECC is more global [7].

Obviously, the k-truss is more cohesive than the k-core, since in a k-truss, each pair of vertices within an edge must have $(k-2)$ common neighbors, while in a k-core, any pair of vertices within an edge may have no common neighbors. Also, the k-ECC is more cohesive than k-core, since it is a connected subgraph and requires that each vertex has at least k neighbors, while a k-core may contain multiple connected components. We further analyze their inclusion ship as follows: Let $G(V, E)$ be a graph and k be an integer ($k \geq 0$). We have:

1. a k-clique must be a subgraph of the k-truss;
2. each connected component of the k-truss must be a subgraph of a particular k-ECC;
3. the k-truss must be a subgraph of the $(k-1)$-core;
4. a k-ECC must be a subgraph of the k-core.

In summary, in terms of structure cohesiveness, the four metrics above can be roughly ranked as: k-core $\preceq$ k-ECC $\preceq$ k-truss $\preceq$ k-clique.

Next, we discuss their computational efficiency.[1] Note that for each metric, there may exist multiple algorithms for enumerating its subgraphs, but here we only discuss complexities of the most efficient ones.

In [17], a linear k-core decomposition algorithm, which computes all the k-cores in the graph G, takes $O(m+n)$ time and $O(m+n)$ space. In [26], Chang et al. proposed an algorithm, which computes all the k-ECCs for a specific k, and it takes $O(h \cdot l \cdot m)$ time and $O(m+n)$ space, where h and l are usually bounded by smaller constants for real graphs [26]. In [184], an efficient algorithm for computing the k-truss, for all $k \geq 3$, takes $O(m^{1.5})$ time and $O(m+n)$ space. In [47], an algorithm, which enumerates all the k-cliques for a specific k, completes in $O(c(G) \cdot \Sigma_{l=2}^{k-1} N^l + k \cdot N^k)$ time and $O(m+n)$ space, where $c(G)$ denotes the maximum core number of vertices in G and N^l is the number of l-cliques. Notice that N^l could be exponentially large. As a result, considering their computational efficiency, we can rank these metrics as: k-core $\succeq$ k-ECC $\succeq$ k-truss $\succeq$ k-clique.

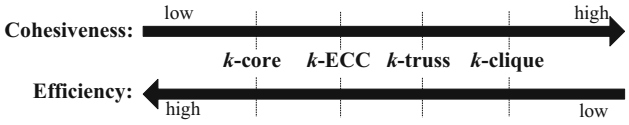


Fig. 4 Comparison of cohesiveness models

Table 2 Efficiency comparison for different metrics

Datasets	k-core (s)	k-ECC (s)	k-truss	k-clique
Email-Enron	0.2	0.8	5	201 s
Google	8.9	40.8	65	> 24 h
Livejournal	85	854	1726	> 24 h
Wise	553 s	5764	32,221	> 24 h

In summary, there is a trade-off between the structure cohesiveness and computational efficiency, as shown in Fig. 4. That is, a more cohesive metric often takes more computational cost. In addition, we have performed a comparison study of the efficiency for these metrics on four real graphs,[2] namely Email-Enron ($|V| = 36.7\,\mathbf{K}$, $|E| = 183.8\,\mathbf{K}$), Google ($|V| = 876\,\mathbf{K}$, $|E| = 5.1\,\mathbf{M}$), Livejournal ($|V| = 4.8\,\mathbf{M}$, $|E| = 69\,\mathbf{M}$), and Wise ($|V| = 58.6\ \mathbf{M}$, $|E| = 265.1\,\mathbf{M}$), where $\mathbf{K} = 10^3$ and $\mathbf{M} = 10^6$. Clearly, as shown in Table 2, the efficiency results well confirm the analysis above.

Based on the comparison analysis above, we would like to make some suggestions: (1) For small or moderate-size graphs, k-clique and k-truss not only achieve higher cohesiveness but also reasonable efficiency. (2) For large graphs, k-core and k-ECC should be better choices since they can be computed more efficiently. (3) For graphs with higher clustering coefficient which can be decomposed into more triangles, k-truss is preferable. (4) For some special graphs (e.g., bipartite graphs), there may not exist any triangles and thus, the k-truss model may not work.

3 *K*-core-based community search

In this section, we review CS works that use the k-core as structure cohesiveness metric. We classify these works into several groups according to the types of graphs, namely undirected graphs, directed graphs, and attributed graphs, including keyword-based, location-based, temporal, influence value-based, and profile-based graphs, and then discuss them, respectively.

[1] Here, we only consider algorithms that assume the graph can be kept in the memory of a single machine.

[2] Email-Enron, Google, Livejournal are downloaded from https://snap.stanford.edu/data/index.html, and Wise is downloaded from http://www.wise2012.cs.ucy.ac.cy/challenge.html.

3.1 Undirected graphs

An undirected graph, denoted by $G(V, E)$, contains a set V of vertices and a set E of edges. Existing CS works on simple undirected graphs can be classified as *size-unbounded* and *size-bounded* CS, where the former one has no constraint on the size of the community and the latter one imposes constraint on the community size.

3.1.1 Size-unbounded community search

In [175], Sozio et al. proposed and studied the problem of community search, defined as follows:

Problem 1 Given an undirected simple graph $G(V, E)$, a set of query vertices $Q \subseteq V$, and a goodness function f, return a subgraph $H(V_H, E_H)$ of G, such that

1. V_H contains Q;
2. H is connected;
3. $f(H)$ is maximized among all feasible choices for H.

Here, $f(H)$ is a general goodness function for measuring cohesiveness of the community H. Intuitively, the value of $f(H)$ should be larger, if H is densely connected. There are many possible choices for f, and an outstanding one is defined based on the *minimum degree*, i.e., $f(H) = \min_{\forall v \in H} \deg_H(v)$. The reasons why the minimum degree is a good metric for the community are threefold: First, minimum degree is one of the most fundamental characteristics of a graph. For instance, it is adopted for describing the evolution of random graphs and graph visualization [46]. Second, it is often used to measure the cohesiveness of user groups in social media. In [170], Seidman et al. compared the minimum degree with many other metrics of cohesiveness (e.g., connectedness and diameter) and found that the minimum degree is indeed a good metric for social network analysis. Third, for community search tasks, Sozio et al. [175] also showed that it is better than some other metrics, including the average degree and density. In the following, we assume that the minimum degree metric is adopted in f.

To solve Problem 1, there are two online algorithms, which are based on global and local search [46,175], respectively, and one index-based algorithm [15].

• **A global search algorithm** Sozio et al. [175] proposed a greedy algorithm, which follows the peeling framework [27] of computing the densest subgraphs [78] and removes vertices iteratively. Specifically, let $G_0 = G$ and G_t be the graph in t-th iteration ($1 \leq t < n$). At the t-th ($1 \leq t < n$) step, it removes the vertex which has the minimum degree in G_{t-1} and obtain an updated graph G_t. The above operation iterates and stops at the T-th step, if either (1) at least one of the query vertices Q has minimum degree in the graph G_{T-1}, or (2) the query vertices Q are no longer connected. Let G'_t be the connected component containing Q in G_t. Then, the subgraph $G_O = \arg\max\{f(G'_t)\}$ satisfies all the constraints in Problem 1.

We denote the algorithm above by `Global`, as it finds the community in a global manner. By using some special optimization techniques [27,175], `Global` is able to achieve linear time and space complexities, i.e., $O(n+m)$. Note that the function $f(H)$ above can be generalized to any monotone function, and the corresponding problem can also be solved by `Global` [175].

It is easy to observe that since `Global` peels all the vertices with low degrees, the subgraph returned is the largest connected subgraph, in which each vertex has at least k neighbors. As a result, the returned subgraph is a connected k-core containing Q, where k equals the minimum core number of vertices in Q.

• **A local search algorithm** According to Problem 1, there may exist some subgraphs of G_O, which satisfy all the constraints and achieve the same value on the function f, but have smaller sizes. Thus, they can be considered the communities as well.

Example 5 Let the graph be the one in Fig. 2a, $Q = \{E\}$. `Global` will return the subgraph of vertices $\{A, B, C, D, E\}$ as the community, and the value of function f is 2. However, there are other three subgraphs, whose vertex sets are $\{A, B, C, E\}$, $\{A, B, D, E\}$, and $\{A, B, E\}$, which also satisfy the constraints of Problem 1, and their values on f are 2. Thus, they can be considered as communities.

In [46], Cui et al. proposed a local CS method, denoted by `Local`, which works in a local expansion manner and finds a community that may have smaller size than that of `Global`. Specifically, it assumes that there is only one query vertex q (i.e., $Q = \{q\}$). `Local` consists of three steps: First, it expands the search space from q. Second, it generates a candidate vertex set C in the search space. Third, it finds the community from C.

The key step is the second step, which works in an iterative manner. In each iteration, it selects the vertex that is the local optimal and adds it into the candidate set C. To decide the local optimal vertex, some heuristic criteria are adopted. One typical criterion is to select the vertex that leads to the largest increment of the function f; another one is to select the vertex which has the largest number of connections to vertices of the candidate set. The iterations stop when the candidate set C theoretically guarantees that it contains a community satisfying the constraints of Problem 1.

Let H and H' denote the communities returned by `Global` and `Local`, respectively. Then, we have $f(H') = f(H)$ and $H' \subseteq H$. Besides, since in the worst case the candidate set C could be the same as vertex set V, the time

complexity of `Local` is the same as that of `Global`, but in practice for large graphs, the candidate set is often much smaller than the entire graph, and thus, `Local` achieves higher efficiency.

• **An index-based algorithm** In [15], Barbieri et al. proposed an index structure, called `ShellStruct`, which organizes all the connected k-cores in an offline manner. Based on `ShellStruct`, Problem 1 can be answered in optimal time cost, i.e., $O(|H_V|)$, where H_V is the set of vertices in the returned community and it is the same as that of `Global`.

The index is built based on the key observation that cores are nested. That is, for any integer $0 < k \leq k_{\max}$, the k-core is contained by the $(k-1)$-core, where $k_{\max}$ is the maximum core number. `ShellStruct` is a tree-like structure with $k_{\max}$ levels. The root of the tree corresponds to the 1-core, and the k-th level keeps track of the information about the k-th core. In k-th level, each tree node, p_k, corresponds to a connected component C_k of the k-core, and it keeps:

1. the set of "children" nodes, each of which corresponds to a connected component that is in the $(k+1)$-core and contained by C_k;
2. the set of vertices in C_k but not in $(k+1)$-core.

It is easy to observe that in `ShellStruct`, all the connected k-cores are well organized. The space cost is exactly $O(n)$ because each vertex appears only once. To build the index, Barbieri et al. proposed an index construction algorithm, which builds the tree level by level, starting from the root level. As a result, its time complexity is $O(n \cdot k_{\max} + m)$. We remark that a more efficient algorithm for building the same index is proposed in [61], which takes $O(m \cdot \alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function and it is less than 5 for all remotely practical values of n.

Based on `ShellStruct`, a query algorithm is proposed. Specifically, it starts from the l-th level where l is the maximum core number of vertices in Q and checks its upper levels, until there is a connected component containing all the query vertices. By using the lowest-common-ancestor (LCA) data structure [72], the time cost of the query algorithm can be reduced to $O(|H_V|)$.

In Problem 1, the cohesiveness function is required to be maximized. However, for some applications, such as infectious disease control discussed in Sect. 1, this constraint may need to be relaxed so that vertices which have less connections with the query vertices can also be involved. Motivated by this, a variant of Problem 1 is also studied in the literature [46]:

Problem 2 Given an undirected simple graph $G(V, E)$, a query vertex $q \in V$, and a nonnegative integer k, return a subgraph $H(V_H, E_H)$ of G, such that

1. V_H contains q;
2. H is connected;
3. for each vertex $v \in H$, $\deg_H(v) \geq k$.

In Fig. 2a, let $q = A$ and $k = 2$. Then, the subgraph of $\{A, B, C, D, E\}$ satisfies all the constraints, and thus is a community for Problem 2. Note that if we maximize the minimum degree as required by Problem 1, we will return a smaller subgraph, i.e., $\{A, B, C, D\}$, since the minimum degree is 3. The algorithms `Global` and `Local` can be easily adapted for answering the query of Problem 2. For details, please refer to [46].

3.1.2 Size-bounded community search

One drawback of Problem 1 is that the returned subgraph may contain a large number of vertices. Notice that although `Local` may find communities which are smaller than those of `Global`, it does not have any guarantee on the sizes of the returned communities, which implies that the returned communities may still have very large sizes.

For many real applications, such as holding a cocktail part, they often require the size of the output community is less than a pre-specified upper bound. Thus, it is desirable to search communities with bounded size. By imposing the size constraint, we obtain another problem:

Problem 3 Given an undirected simple graph $G(V, E)$, a set of query vertices $Q \subseteq V$, a size constraint k, and a goodness function f, return a subgraph $H(V_H, E_H)$ of G, such that

1. V_H contains Q;
2. H is connected;
3. $|V_H| \leq k$ (H has at most k vertices);
4. $f(H)$ is maximized among all feasible choices for H.

Unfortunately, due to the size constraint, Problem 3 is NP-hard [175]. This implies that an exact algorithm for solving Problem 3 will take exponential time cost, and thus, it is impractical for large graphs. To alleviate the computational issue, some heuristic algorithms are developed [175], and they are able to achieve reasonable efficiency, although they do not have any provable quality guarantee.

To further reduce the size of the returned community, Barbieri et al. [15] proposed the *minimum community search problem*, which aims to find a community that satisfies all the constraints of Problem 1 and has the minimum number of vertices.

Problem 4 Given an undirected simple graph $G(V, E)$, a set of query vertices $Q \subseteq V$, and a minimum degree-based function f, let H^* be the subgraph returned by `Global`. Find a subgraph H of G, such that

1. V_H contains Q;
2. H is connected;
3. $f(H) = f(H^*)$;
4. the size of H is the smallest.

Similar to Problem 3, Problem 4 is also NP-hard. It can be proved by a reduction from the STEINER TREE problem: Given a graph $G(V, E)$ and a set of terminal vertices $T \subseteq V$, find a connected subgraph G' of G such that it contains all the terminal vertices and has the minimum number of edges. Note that the most efficient algorithm [115] of STEINER TREE problem achieves an approximation ratio of $(2\text{-}2/|Q|)$ and takes linear time cost by the Mehlhorn's implementation [143].

To answer the query in Problem 4, Barbieri et al. [15] proposed an algorithm, and it consists of two steps: First, it reduces the size of H^* as much as possible using some local greedy search. Note that after the reduction, the subgraph H^* is still a qualified community of Problem 1, but may have much smaller size. Second, it finds a subgraph from H^* by adopting the above approximation algorithm for the STEINER TREE problem.

Remark Some other factors, such as distances among vertices [175] and local distance dynamics [24,144], have also been considered for CS on simple graphs. Due to the space limitation, we skip the details.

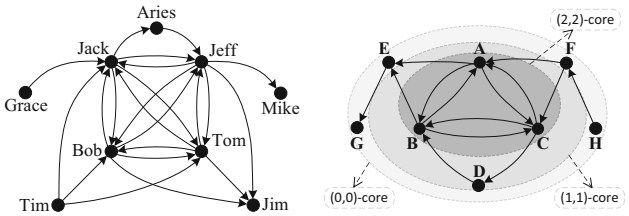
3.2 Directed graphs

A directed graph is a graph $G(V, E)$, which contains a set of vertices V and a set of directed edges E. The in-degree and out-degree of a vertex v in G, denoted by $\deg_G^{\text{in}}(v)$ and $\deg_G^{\text{out}}(v)$, are the number of its in-neighbors and out-neighbors, respectively. The minimum in-degree and out-degree of the graph G are denoted by $\delta_{\text{in}}(G)$ and $\delta_{\text{out}}(G)$, respectively. Figure 5a depicts a directed graph with nine users.

A straightforward method of performing CS on directed graph is to ignore the directions and then use the method `Global` in Sect. 3.1.1 to find the community. In Fig. 5a, if we let q = Jack, then we will find a community with members {`Jack`, `Jeff`, `Bob`, `Tom`, `Tim`, `Jim`}. However, `Tim` has no in-neighbors and `Jim` has no out-neighbors in the community, which implies their interactions with other members are quite weak.

In [66], Fang et al. extended the minimum degree measure for directed graphs, and studied the problem of community search on directed graph (or CSD problem), based on the D-core, also called (k, l)-core [75].

Definition 10 (*(k, l)-core* [75]) Given a directed graph $G(V, E)$ and two nonnegative integers k and l, the (k, l)-core is the maximum subgraph C of G such that $\delta_{\text{in}}(C) \geq k$ and $\delta_{\text{out}}(C) \geq l$.



(a) a directed graph **(b)** illustrating D-cores

Fig. 5 Two directed graphs [66]

Problem 5 (*CSD*) Given a directed graph $G(V, E)$, two positive integers k and l, and a query vertex q, return a connected subgraph $G_q \subseteq G$, such that it contains q and $\forall v \in G_q$, $\delta_{\text{in}}(G_q) \geq k$ and $\delta_{\text{out}}(G_q) \geq l$.

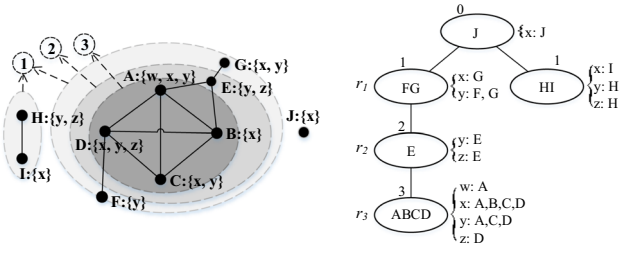
Figure 5b shows a directed graph with its D-cores. Let $q = B$, $k = 2$, and $l = 2$. Then, the subgraph of $\{A, B, C\}$ is the returned community for B.

Similar to `Global`, a simple solution to the CSD problem is to peel vertices iteratively until each remaining vertex satisfies the in-degree and out-degree constraints. As a result, its time complexity is $O(m + n)$, which may be inefficient for large graphs. To improve efficiency, Fang et al. [66] proposed an index-based method. Specifically, it first performs D-core decomposition (i.e., computing all the (k, l)-cores), then organizes these cores in an index with a two-dimensional table, and finally answers queries using the index.

To keep all D-cores, a simple method takes $O(n^3)$ space since $k, l \leq n - 1$ and each D-core takes $O(n)$ space. To alleviate this issue, three methods are proposed. For ease of exposition, let $V_{i,j}$ denote the set of vertices in (i, j)-core. The first one exploits the nested property of D-cores, i.e., for any $l \geq 0$, we have $(k, l + 1)$-core $\subseteq (k, l)$-core, so if $(k, l + 1)$-core has been kept, we only need to keep vertices $V_{k,l} \backslash V_{k,l-1}$ for the (k, l)-core. As a result, for any k, it takes $O(n)$ space to keep all (k, l)-cores $(0 \leq l \leq n)$, so the overall space cost is $O(m)$.

The second method relies on a key observation that for any $k, l \geq 0$, we have both $(k + 1, l)$-core $\subseteq (k, l)$-core and $(k, l+1)$-core $\subseteq (k, l)$-core. After keeping $(k, l+1)$-core and $(k+1, l)$-core, for (k, l)-core, if $|V_{k+1,l}| \geq |V_{k,l+1}|$, we only keep $V_{k,l} \backslash V_{k+1,l}$; otherwise, we keep $V_{k,l} \backslash V_{k,l+1}$. Thus, it takes less space than the first method. For the third method, after keeping $(k, l+1)$-core and $(k+1, l)$-core, it only keeps vertices $V_{k,l} \backslash (V_{k+1,l} \cup V_{k,l+1})$ for the (k, l)-core and takes the least space cost.

In addition, although the community G_q of a CSD query is a connected subgraph, it may not be a strongly connected component (SCC) [92] (i.e., each vertex of the SCC is reachable from each other vertex). To tackle this issue, a variant of



(a) a graph G **(b)** CL-tree index of G

Fig. 6 An example for illustrating ACQ [61]

the CSD problem is to find a community, which not only satisfies the minimum degree constraints, but also is an SCC. The CSD algorithms can be extended for solving this variant [66].

3.3 Keyword-based attributed graphs

A keyword-based attributed graph is an undirected graph $G(V, E)$, with vertex set V and edge set E. Each vertex $v \in V$ is associated with a set of keywords, $W(v)$. The keyword-based attributed graphs are prevalent in social media, bibliographical networks, and knowledge bases. In Fig. 6a, a keyword-based attributed graph is depicted. For example, vertex A has a set of keywords $\{w, x, y\}$. In [57,58,61,173], CS on keyword-based attributed graphs has been studied extensively.

Problem 6 (*ACQ* [61]) Given a keyword-based attributed graph $G(V, E)$, a positive integer k, a vertex $q \in V$, and a set of keywords $S \subseteq W(q)$, return a set $\mathcal{G}$ of subgraphs of G, such that $\forall G_q \in \mathcal{G}$, and the following properties hold:

1. *Connectivity* G_q is connected and contains q;
2. *Structure cohesiveness* $\forall v \in G_q, \deg_{G_q}(v) \geq k$;
3. *Keyword cohesiveness* The size of $L(G_q, S)$ is maximal, where $L(G_q, S) = \cap_{v \in G_q}(W(v) \cap S)$ is the set of keywords shared in S by all vertices of G_q.

For example, in Fig. 6a, if $q = A$, $k = 2$ and $S = \{w, x, y\}$, then the output of Problem 6 is the subgraph of $\{A, C, D\}$, with a shared keyword set $\{x, y\}$, meaning that these vertices share the keywords x and y.

The subgraph G_q is called an *attributed community* (or AC) of q, and $L(G_q, S)$ is the *AC-label* of G_q. In Problem 6, the first two properties ensure the structure cohesiveness. Property 3 enables the retrieval of communities whose vertices have common keywords in S. It requires $L(G_q, S)$ to be maximal, because it aims to find the AC(s) only containing the most related vertices, in terms of the number of common keywords. In Fig. 6a, if we use the same query ($q = A$, $k = 2$, $S = \{w, x, y\}$), without the "maximal" requirement, we can obtain communities such as $\{A, B, E\}$ (which share no keywords), $\{A, B, D\}$, or $\{A, B, C\}$ (which share 1 keyword). Note that there does not exist an AC with AC-label being exactly $\{w, x, y\}$.

Two outstanding features of ACQ are as follows: (1) *Ease of interpretation.* An AC contains tightly connected vertices with similar contexts or backgrounds. Thus, an ACQ user can focus on the common keywords or features of these vertices, i.e., the AC-labels facilitate understanding of the vertices that form the AC. (2) *Personalization.* The user of an ACQ can control the semantics of the AC, by specifying a set of S of keywords. Intuitively, S decides the meaning of the AC based on the user's need.

The ACQ problem is challenging. A simple method to answer an ACQ runs three steps. First, all non-empty subsets of S, $S_1, S_2, \ldots, S_{2^l-1}$ ($l = |S|$), are enumerated. Then, for each subset $S_i (1 \leq i \leq 2^l - 1)$, it checks whether there is a subgraph which satisfies the first two properties. Finally, it outputs the subgraphs having the most shared keywords. However, since there are exponential number of subsets, it is impractical for large graphs. To alleviate this issue, the authors observed the *anti-monotonicity* property, which states that given a set S of keywords, if it appears in every vertex of an AC, then for every subset S' of S, there exists an AC in which every vertex contains S'. Based on this property, many subsets of S can be pruned, and thus, faster online query algorithms can be developed.

An index, called `CL-tree`, is proposed for organizing the vertex keyword data in a hierarchical structure. The CL-tree has the same tree structure as `ShellStruct` (see Sect. 3.1.1), but for each node p, it maintains an additional inverted list such that for each keyword e that appears in the vertices of p, a list of IDs of vertices which contain e is stored. Since each graph vertex and each keyword appear only once, the space cost of keeping such an index is $O(\widehat{l} \cdot n)$, where $\widehat{l}$ denotes the average size of $W(v)$ over V. As a result, the space cost is linear to the size of G. As shown in [61], the `CL-tree` structure can be built level by level in a bottom-up manner and it takes linear time cost, i.e., $O(m \cdot \alpha(n))$. In addition, index maintenance algorithms for the `CL-tree` are developed [58]. Figure 6b presents the `CL-tree` index for the graph in Fig. 6a.

Based on the `CL-tree`, two incremental algorithms (from examining smaller candidate keyword sets to larger ones) and one decremental algorithm (from examining larger candidate keyword sets to smaller ones) are developed. For each candidate keyword set, they check whether there is a connected k-core containing q and finally return the one with largest keyword set.

3.4 Location-based attributed graphs

A location-based attributed graph, also called geo-social network, is an undirected graph $G(V, E)$ with vertex set V and

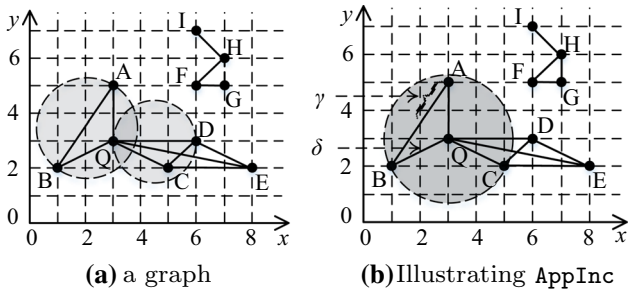


Fig. 7 Illustration of SAC search [60]

Table 3 CS works on geo-social networks

CS query	Spatial cohesiveness
SAC search [60,65]	Smallest minimum covering circle
RB-k-core search [185]	Radius-fixed covering circle
GSGQ [221]	Rectangle, center-fixed circles

edge set E. For each vertex $v \in V$, it has a location position $(v.x, v.y)$, where $v.x$ and $v.y$ denote its positions along x- and y-axis in a two-dimensional space. Geo-social networks widely exist in many location-based services, including Twitter, Facebook, and Foursquare [12,63,68]. In Fig. 7a, a geo-social network with ten vertices is depicted.

Three kinds of CS queries have been studied on geo-social networks, namely *spatial-aware community (SAC) search* [60], *radius-bounded k-core (RB-k-core) search* [185], and *geo-social group queries with minimum acquaintance constraint (GSGQ)* [221]. Generally, they all require that the communities are structurally and spatially cohesive. For structure cohesiveness, they all adopt the k-core model, but for spatial cohesiveness, they use different constraints, as outlined in Table 3. In SAC search, the community is in the smallest minimum covering circle (MCC); in RB-k-core search, the community is in a circle with radius less than an input threshold; in GSGQ, the community is in a given rectangle or circle centered at the query vertex.

3.4.1 Spatial-aware community (SAC) search

The MCC and SAC search are defined as follows. Note that the notion of MCC has been widely adopted to describe a set of spatially compact objects [53,85].

Definition 11 (*MCC*) Given a set S of vertices with locations, the MCC of S is the spatial circle, which contains all the vertices in S with the smallest radius.

Problem 7 (*SAC search*) Given a geo-social network $G(V, E)$, a positive integer k, and a vertex $q \in V$, return a subgraph $G_q \subseteq G$, and the following properties hold:

1. *Connectivity* G_q is connected and contains q;
2. *Structure cohesiveness* $\forall v \in G_q, \deg_{G_q}(v) \geq k$;
3. *Spatial cohesiveness* The MCC of vertices in G_q satisfying Properties 1 and 2 has the smallest radius.

A subgraph satisfying properties 1 and 2 is a *feasible* solution, and the subgraph satisfying all the three properties is the *optimal* solution (denoted by Ψ). The radius of the MCC containing Ψ is denoted by r_{opt}. In Fig. 7a, the two circles denote the MCCs of $C_1 = \{Q, C, D\}$ and $C_2 = \{Q, A, B\}$. Let $q = Q$ and $k = 2$. Then, Ψ contains vertex set C_1 with $r_{\text{opt}} = 1.5$.

The SAC search problem is challenging. A basic exact approach takes $O(m \times n^3)$ time, which relies on an observation that a spatial circle can be determined by three points on its boundary [53]. This implies that we can enumerate all the three-vertex combinations, and for each combination we find a connected k-core in its circle, and finally get Ψ. This approach, however, is impractical for large graphs due to its high complexity.

To improve efficiency, the authors resorted to approximation algorithms. The first one, called `AppInc`, returns the feasible solution in a circle $O(q, \delta)$ which centers at q and has the smallest radius δ, and it has an approximation ratio of 2. Here, the approximation ratio is defined as the ratio of the radius of MCC returned over r_{opt}. In Fig. 7b, let $q = Q$ and $k = 2$. Then, `AppInc` returns the subgraph of $\{A, B, Q\}$.

The circle $O(q, \delta)$ can also be approximated by performing binary search on the radius δ. As a result, we can get another approximation solution with ratio of $(2+\epsilon_F)$, where $\epsilon_F \geq 0$ is an input parameter. To achieve an approximation ratio of $(1+\epsilon_A)$ where $0 < \epsilon_A < 1$, the authors developed another algorithm, called `AppAcc`. It first locates the area containing the center of the circle of Ψ, then approximates the center by splitting the area into small grids, and finally finds an approximation solution by using these grids. Overall, these approximation algorithms guarantee that the radius of the MCC of Ψ has an arbitrary expected approximation ratio. Based on `AppAcc`, an advanced exact algorithm is developed. An interesting observation is that there is a trade-off between the quality of results and efficiency, i.e., algorithms with lower approximation ratios tend to have higher complexities. In addition, the SACs can be returned in a continuous manner, as shown in [65].

3.4.2 Radius-bounded k-core search

Problem 8 defines the problem of radius-bounded k-core search, or RB-k-core search.

Problem 8 (*RB-k-core search*) Given a geo-social network $G(V, E)$, a positive integer k, a radius r, and a vertex $q \in V$, return all the subgraphs $G_q \subseteq G$, and the following properties hold:

1. *Connectivity* G_q is connected and contains q;
2. *Structure cohesiveness* $\forall v \in G_q, \deg_{G_q}(v) \geq k$;
3. *Spatial cohesiveness* The MCC of vertices in G_q has a radius $r' \leq r$;
4. *Maximality constraint* There exists no other subgraph G'_q satisfying properties above and $G_q \subset G'_q$.

Similar to SAC search, it adopts the MCC, but imposes a constraint on its radius. To solve Problem 8, Wang et al. proposed three algorithms. The first one, denoted by `TriV`, is a triple-vertex-based algorithm, which is also based on the observation that a spatial circle can be determined by three points on its boundary [53]. It proposes to generate all the candidate circles containing q at first and then compute the maximum k-core for the subgraphs contained in the candidate circles with radius $r' \leq r$. The time complexity of `TriV` is $O(mn^3)$, since there are $O(n^3)$ candidate circles in the worst case and each circle needs $O(m)$ time to verify.

To reduce the number of candidate circles, a binary-vertex-based algorithm `BinV` is proposed. In `BinV`, only the circles with radius $r' = r$ are generated and for each candidate circle, its arc passes a pair of vertices in G. In this manner, for each pair of vertices, at most two circles are generated. As a result, it reduces the number of candidate circles from $O(n^3)$ to $O(n^2)$.

To further improve the efficiency, a rotating-circle-based algorithm `RotC` is proposed to reuse the intermediate computation results in the process of finding RB-k-cores. Fixing each vertex $v \in V$ as a pole, `RotC` generates the candidate circles in a rotating way so that the computation cost can be shared among the adjacent circles. In addition, the authors also proposed several pruning techniques to early terminate the processing of invalid candidate circles.

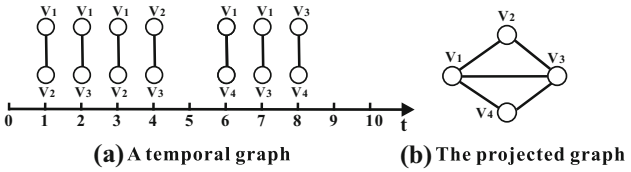
3.4.3 Geo-social group queries with minimum acquaintance constraint (GSGQs)

The GSGQ is defined formally as follows:

Problem 9 (*GSGQ*) Given a geo-social network $G(V, E)$, a vertex $q \in V$, a positive integer k, and a spatial constraint Λ, return a subgraph $G_q \subseteq G$, and the following properties hold:

1. *Connectivity* G_q is connected and contains q;
2. *Structure cohesiveness* $\forall v \in G_q, \deg_{G_q}(v) \geq k$;
3. *Spatial cohesiveness* G_q satisfies constraint Λ.
4. *Maximality constraint* There exists no other subgraph G'_q satisfying properties above and $G_q \subset G'_q$.

In Problem 9, for spatial constraint Λ, Zhu et al. [221] considered three kinds of constraints:



(a) A temporal graph **(b)** The projected graph

Fig. 8 A temporal graph and the projected graph [129]

1. Λ is a spatial rectangle for containing G_q;
2. Λ is a circle centered at q with radius less than the distance from q to its k-th nearest vertex in G_q (G_q may contain more than $k + 1$ vertices);
3. Λ is a circle satisfying Constraint 2 and G_q contains exactly $k + 1$ vertices.

By using an R-tree index [86], a GSGQ with the first constraint can be answered in $O(n + m)$ time; when the second constraint is imposed, a GSGQ can be solved in $O(n(n+m))$ time; when the third constraint is applied, a GSGQ takes $O(C_k^{n-1}(m + n))$ time.

To improve efficiency, they proposed the social-aware R-trees (or SaR-tree) index, which incorporates both vertices' spatial locations and social relations. It is built based on the concept of core bounding rectangle (CBR), which projects the minimum degree constraint on the spatial layer. Specifically, the CBR of a vertex v is a rectangle containing v, inside which any vertex group with v does not satisfy the minimum degree constraint.

Unlink classical R-tree, each entry of an SaR-tree refers to two pieces of information, i.e., a set of CBRs and a minimum bounding rectangle (MBR). Perceptually, a CBR bounds a group of vertices from the social perspective, while an MBR bounds vertices from the spatial perspective. As such, SaR-tree gains power for both social-based and spatial-based pruning. In addition, they developed a variant of SaR-tree, called SaR*-tree, which optimizes the group of spatial objects to minimize the disk I/O cost. Based on these indexes, they developed efficient algorithms for answering GSGQs with different spatial constraints.

3.5 Temporal graphs

Li et al. [129] studied the persistent community search problem in a temporal graph. A temporal graph is an undirected graph $G(V, E)$ with vertex set V and edge set E. Each edge $e \in E$ is a triplet (u, v, t), where u, v are vertices in V and t is the interaction time between u and v. For a temporal graph G, the projected graph denoted by G_p over the time interval $[t_s, t_e]$ is defined as $G_p = (V, E, [t_s, t_e])$, where $V = V(G)$ and $E = \{(u, v)|(u, v, t) \in E(G), t \in [t_s, t_e]\}$. Figure 8b illustrates the projected graph of the temporal graph in Fig. 8a over the interval [1, 8].

Definition 12 (*Maximal* (θ, k)*-persistent-core interval*) Given a temporal graph $G = (V, E)$ and parameters $\theta > 0$ and $k > 0$, an interval $[t_s, t_e]$ with $t_e - t_s \geq \theta$ is called a maximal (θ, k)-persistent-core interval for G if and only if the following two conditions hold. (1) For any $t \in [t_s, t_e - \theta]$, the projected graph of G over the interval $[t, t+\theta]$ is a connected k-core subgraph. (2) There is no super-interval of $[t_s, t_e]$ such that (1) holds.

Definition 13 (*Core persistence*) Let $T = \{[t_{s_1}, t_{e_1}], \ldots, [t_{s_r}, t_{e_r}]\}$ be the set of all maximal (θ, k)-persistent-core intervals of G. Then, the core persistence of G with parameters θ and k, denoted by $F(\theta, k, G)$, is defined as

$$F(\theta, k, G) = \begin{cases} \sum_{i=1}^{r} (t_{e_i} - t_{s_i}) - (r-1)\theta, & \text{if } T \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Definition 14 ((θ, τ)*-persistent k-core*) Given a temporal graph G and parameters θ, τ, and k, a (θ, τ)-persistent k-core is an induced temporal subgraph $C = (V_C, E_C)$ that meets the following properties.

1. *Persistent-core property* $F(\theta, k, C) \geq \tau$;
2. *Maximal property* There does not exist an induced temporal subgraph C' that contains C and also satisfies the persistent-core property.

Problem 10 (*The persistent community search problem*) Given a temporal graph G and parameters θ, τ and k, the persistent community search problem aims to find the largest (θ, τ)-persistent k-core in G.

Consider the temporal graph G in Fig. 8a. Assume that $\theta = 3$ and $k = 2$. We can see that there is no maximal $(3, 2)$-persistent-core interval for the entire graph G. There is a maximal $(3, 2)$-persistent-core interval $[1, 5]$ for the subgraph C induced by vertices $\{v_1, v_2, v_3\}$. This is because $[1, 5]$ is the maximal interval such that in any 3-length subinterval of $[1, 5]$, the vertices $\{v_1, v_2, v_3\}$ form a connected 2-core. Let $\tau = 4$, we can see that the subgraph C induced by vertices $\{v_1, v_2, v_3\}$ is a $(3, 4)$-persistent 2-core. Because $F(3, 2, C) = 4$, which is no less than τ; and C is the maximal subgraph that meets such a persistent-core property.

As shown in [129], the persistent community search problem is NP-hard. Therefore, a prune-and-search approach is proposed in [129]. In the pruning phase, a temporal graph reduction algorithm is designed by decomposing the whole time span of the temporal graph into several meta-intervals, each of which has some properties to prune vertices. In the search phase, a branch-and-bound algorithm with several pruning rules is proposed to find the maximum (θ, τ)-persistent k-core.

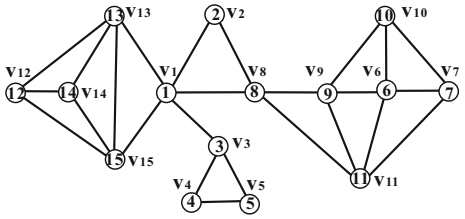


Fig. 9 An example of influential CS (the numbers denote the weights) [127]

3.6 Influence value-based attributed graphs

3.6.1 Single-dimensional influential CS

Li et al. [127] proposed the influential CS problem. They considered an undirected graph $G(V, E)$ with vertex set V and edge set E. Each vertex $v \in V$ is associated with a weight w_u indicating the influence (or importance) of u. Without loss of generality, they assumed that the weight vector $W = (w_1, w_2, \ldots, w_n)$ forms a total order, i.e., for any two vertices v_i and v_j, if $i \neq j$, then $w_i \neq w_j$.

Definition 15 (*Influence value of a subgraph*) Given an undirected graph $G(V, E)$ and an induced subgraph $H(V_H, E_H)$ of G, the influence value of H denoted by $f(H)$ is defined as the minimum weight of the vertices in H, i.e., $f(H) = \min_{u \in V_H}\{w_u\}$.

Definition 16 (k*-influential community*) Given an undirected graph $G = (V, E)$ and an integer k, a k-influential community is an induced subgraph $H^k = (V_H^k, E_H^k)$ of G that meets all the following constraints.

1. *Connectivity* H^k is connected;
2. *Cohesiveness* Each vertex u in H^k has degree at least k;
3. *Maximal structure* There is no other induced subgraph $\tilde{H}$ such that (1) $\tilde{H}$ satisfies connectivity and cohesiveness constraints, (2) $\tilde{H}$ contains H^k, and (3) $f(\tilde{H}) = f(H^k)$.

Consider the graph shown in Fig. 9. Suppose, for instance, that $k = 2$, then by definition the subgraph induced by vertex set $\{v_{12}, v_{13}, v_{14}, v_{15}\}$ is a 2-influential community with influence value 12, as it meets all the constraints in Definition 16. Note that the subgraph induced by vertex set $\{v_{12}, v_{14}, v_{15}\}$ is not a 2-influential community. This is because it is contained in a 2-influential community induced by vertex set $\{v_{12}, v_{13}, v_{14}, v_{15}\}$ whose influence value equals its influence value, and thus fail to satisfy the maximal structure constraint.

Problem 11 (*Top-r k-influential CS problem* (*TIC*)) Given a graph $G(V, E)$ and two parameters k and r, the problem is to find the top-r k-influential communities with the highest influence value.

Definition 17 (*Non-contained k-influential community*) Given a graph $G(V, E)$ and an integer k, a non-contained k-influential community $H^k = (V_H^k, E_H^k)$ is a k-influential community that meets the following constraint.

– *Non-containment* H^k cannot contain a k-influential community $\bar{H}^k$ such that $f(\bar{H}^k) > f(H^k)$.

Consider the graph shown in Fig. 9. Assume that $k = 2$. By Definition 17, we can see that the subgraphs induced by $\{v_3, v_4, v_5\}$, $\{v_8, v_9, v_{11}\}$ and $\{v_{13}, v_{14}, v_{15}\}$ are non-contained 2-influential communities. However, the subgraph induced by $\{v_{12}, v_{13}, v_{14}, v_{15}\}$ is not a non-contained 2-influential community, because it includes a 2-influential community (the subgraph induced by $\{v_{13}, v_{14}, v_{15}\}$) with a larger influence value.

Problem 12 (*Top-r non-contained k-influential CS problem*) Given a graph $G(V, E)$ and parameters k and r, find the top-r non-contained k-influential communities with the highest influence value.

• **Online search algorithms** An online search algorithm is proposed in [127] to compute the top-r (non-contained) k-influential communities given graph G and parameters r and k. The algorithm first computes the k-core C of G and then iteratively updates C by removing vertices from C until C becomes empty. In each iteration, a vertex u with the smallest influence value is removed from C. After u is removed, the algorithm further removes those vertices that do not belong to the k-core from C by invoking a DFS procedure. For each iteration, the connected component that vertex u belongs to forms a k-influential community. The k-influential communities obtained by the last r iterations are the top-r k-influential communities. If after deleting a certain u, the vertices in the whole connected component that u belongs to are removed in the DFS procedure, then the corresponding connected component is a non-contained k-influential community. In this way, we can obtain the top-r non-contained k-influential communities. The algorithm runs in $O(m + n)$ time using $O(m + n)$ space.

The above algorithm needs to compute all (non-contained) k-influential communities before obtaining the top-r (non-contained) k-influential communities which is costly when the graph is large and r is small. Therefore, Chen et al. [30] proposed a backward search algorithm to obtain the top-r (non-contained) k-influential communities. The general idea is as follows: Instead of deleting the vertex with the smallest influence value each time, the backward search algorithm initializes an empty vertex set C and inserts into C the vertex with the largest influence value in each iteration. After a vertex u with the largest influence value is inserted, if the core number of u in the subgraph induced by C is no smaller than k, the connected component containing u in the subgraph induced by C represents a k-influential community. The algorithm can terminate once r k-influential communities are reported. The top-r non-contained k-influential communities can be computed in a similar way by checking whether each k-influential community is a non-contained k-influential community before reporting the community.

The online search algorithms in [127] and [30] need to access the whole graph to obtain the top-r (non-contained) k-influential communities. To solve this issue, Bi et al. [21] proposed a local search algorithm. Let $G_{\geq\tau}$ be the subgraph of G induced by all vertices with weights at least τ, the authors proved that *if the subgraph* $G_{\geq\tau}$ of G contains at least r k-influential communities, then the top-r k-influential communities in $G_{\geq\tau}$ are the query result. The goal is to find the smallest subgraph $G_{\geq\tau^*}$ of G containing at least r k-influential communities. The general idea is as follows: The algorithm starts with a large τ and iteratively decreases the value of τ until reaching the target value. For each τ, only the vertices with weights no smaller than τ need to be accessed. The authors proved that the time complexity of the algorithm is linear to the size of the smallest subgraph $G_{\geq\tau^*}$ that an online search algorithm without indexes needs to access to correctly compute the top-r k-influential communities. Thus, the algorithm is instance optimal. Their algorithm can be easily extended to solve Problem 12.

• **An index-based algorithm** In [127], an index, called ICP-Index, is presented for solving Problem 12. The index is designed based on the observation that *for each k, the k-influential communities form an inclusion relationship*. Based on such an inclusion relationship, all the k-influential communities can be organized by a tree-shape structure. The index includes such tree structures for all possible k values. In addition, instead of keeping the whole community for each tree node, a compression method is proposed to make the ICP-Index compact. Specifically, for each non-leaf node in the tree which corresponds to a k-influential community, the index only stores the vertices of the k-influential communities that are not included in their sub-k-influential communities. The same idea is recursively applied to all the non-leaf nodes of the tree following a top-down manner. For each leaf node which corresponds to a non-contained k-influential community, the index stores all the vertices of that non-contained k-influential community. Using the ICP-Index, the query can be answered efficiently because each node in the tree corresponds to a k-influential community and each leaf node in the tree corresponds to a non-contained k-influential community. In [127], the authors proved that the ICP-Index can be constructed in $O(m^{1.5})$ time using $O(m + n)$ space.

Consider the graph shown in Fig. 9. Let us consider the case of $k = 2$. Clearly, the entire graph is a connected 2-core, so it is a 2-influential community. Therefore, the root node

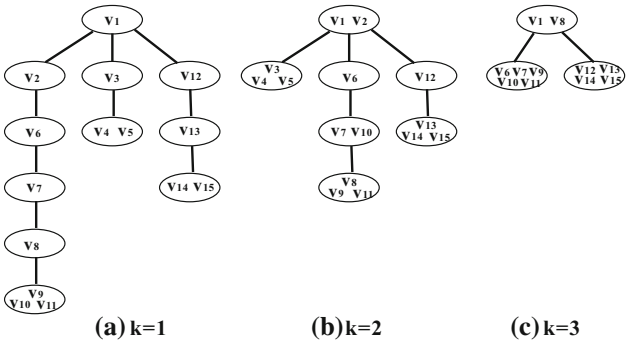


Fig. 10 Tree organization of all the k-influential communities (the ICP-Index) [127]

of the tree corresponds to the entire graph. After deleting the smallest-weight vertex v_1, we get three 2-influential communities which are the subgraphs induced by the vertex sets $\{v_3, v_4, v_5\}$, $\{v_6, \ldots, v_{11}\}$, and $\{v_{12}, \ldots, v_{15}\}$, respectively. Thus, we create three child nodes for the root node which correspond to the three 2-influential communities, respectively. Since v_1 and v_2 are not included in these three 2-influential communities, we store them in the root node. The same idea is recursively applied in all the three 2-influential communities.

Figure 10 shows the tree organization for all k for the graph shown in Fig. 9.

• **An I/O-efficient algorithm** An I/O-efficient algorithm to compute the top-r (non-contained) k-influential communities is presented in [128]. It assumes that all vertices of the graph can be stored in the main memory. The key idea of the algorithm is that it computes the k-influential communities following the decreasing order of their weights, and the communities (as well as the edges in community) with large weights can be safely deleted without affecting the correctness of the algorithm to compute the tree vertices with small weights. Specifically, let $w(e) = \min\{w_u, w_v\}$ be the weight of an edge $e = (u, v)$. The algorithm first sorts the edges in a non-increasing order of their weights using the standard external-memory sort algorithm. (We can use the vertex ID to break ties.) Then, following this order, the algorithm loads the edges into the main memory up to the memory limit. Subsequently, the algorithm invokes an in-memory algorithm to compute the influential communities in the main memory. After that, the algorithm deletes the computed influential communities as well as the associated edges from the main memory and then sequentially loads new edges into the main memory until reaches the memory limit. The algorithm iteratively performs this procedure until all the edges are scanned. Note that in each iteration, the algorithm only works on a partial graph, which is loaded in the main memory.

As an example, consider the graph shown in Fig. 9. Suppose $k = 2$ and the memory can hold at most 10 edges. The

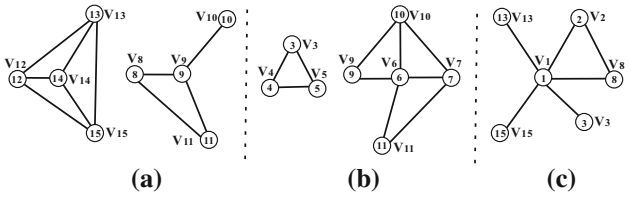


Fig. 11 Partial graphs in the memory ($k = 2$, memory can hold at most 10 edges) [128]

partial graph loaded into memory in the first three iterations for the algorithm is shown in Fig. 11

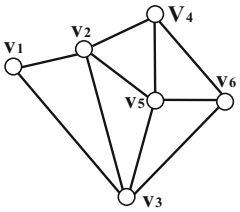
• **Center-core CS** Another model to capture the influence of vertices is called the center-core community search, which is studied by Ding et al. [50]. The model uses k-core to qualify the dense structure for the community and uses coreness to evaluate the vertex influence. Given a query vertex q and an integer k, the center-core community is a connected component of the maximal k-core containing the query vertex q and the coreness of vertices in the community is no less than q. In addition, the community excludes those vertices with coreness equal to q but cannot be reached from q via vertices with the same coreness with q. An online search algorithm and an index-based algorithm are proposed in [50] to compute the center-core community.

3.6.2 Multi-dimensional influential CS

In [126], Li et al. studied the multi-dimensional influential CS. It deals with a multi-valued graph $G(V, E, X)$ where V and E denote the set of vertices and edges, respectively, and X ($|X| = n$) is a set of d-dimensional vectors. In a multi-valued graph, each vertex $v \in V$ is associated with a d-dimensional real-valued vector denoted by $X_v = (x_1^v, \ldots, x_d^v)$, where $X_v \in X$ and $x_i^v \in \mathbb{R}$. Suppose without loss of generality that on the x_i dimension, x_i^v for all $v \in V$ form a strict total order, i.e., $x_i^v \neq x_i^u$ for any $u \neq v$. It is important to note that if this assumption does not hold, we can easily construct a strict total order by using the vertex identity to break ties for any $x_i^v = x_i^u$. The d-dimensional vector X_v represents the values of the vertex v w.r.t. d different numerical attributes. The model studied in [126], called the skyline community search, is based on the one-dimensional influential community model proposed in [127]. The authors defined the value of H on the x_i dimension (for $i = 1, 2, \ldots, d$) as $f_i(H) \triangleq \min_{v \in V(H)}\{x_i^v\}$.

Definition 18 Let $H(V_H, E_H)$ and $H'(V_{H'}, E_{H'})$ be two subgraphs of a multi-valued graph G. If $f_i(H) \leq f_i(H')$ for all $i = 1, \ldots, d$, and there exists $f_i(H) < f_i(H')$ for a certain i, we call that H' dominates H, denoted by $H \prec H'$.

Definition 19 Given a multi-valued graph $G(V, E, X)$ and an integer k. A skyline community with a parameter k is an



	x_1	x_2	x_3
v_1	8	14	7
v_2	9	15	12
v_3	11	17	3
v_4	6	16	4
v_5	7	8	5
v_6	10	9	10

Fig. 12 An example of a multi-valued graph [126]

induced subgraph $H(V_H, E_H, X_H)$ of G such that it satisfies the following properties.

1. *Cohesive property* H is a connected k-core;
2. *Skyline property* There does not exist an induced subgraph H' of G such that H' is a connected k-core subgraph and $H \prec H'$;
3. *Maximal property* There does not exist an induced subgraph H' of G such that (1) H' is a connected k-core subgraph, (2) H' contains H, and (3) $f_i(H') = f_i(H)$ for all $i = 1, \ldots, d$.

Problem 13 (*Skyline CS problem*) Given a multi-valued graph $G(V, E, X)$ and an integer k, the problem is to find all the skyline communities from G with the parameter k. More formally, let $\mathcal{H}$ be the set of all connected k-core subgraphs in G. We aim to compute a subset $\mathcal{R}$ of $\mathcal{H}$ which is defined as:

$$\mathcal{R} \triangleq \{H \in \mathcal{H} | \neg \exists H', H'' \in \mathcal{H} : H \prec H', \\ H \subset H'' \wedge f(H) = f(H'')\},$$

where $H \subset H''$ denotes that H is a subgraph of H'' and $H \neq H''$, and $f(H) = f(H'')$ means that $f_i(H) = f_i(H'')$ for $i = 1, \ldots, d$.

Consider the graph shown in Fig. 12. The left panel is a graph with 6 vertices, and the right panel shows the values of these vertices in three different dimensions. Suppose, for instance, that $k = 2$. Then, by Definition 19, $H_1 = \{v_1, v_2, v_3\}$ is a skyline community with values $f(H_1) = (8, 14, 3)$, because there does not exist a connected 2-core subgraph that can dominate it, and it is also the maximal subgraph that satisfies the cohesive and skyline properties. Similarly, $H_2 = \{v_2, v_4, v_5, v_6\}$ is a skyline community with $f(H_2) = (6, 8, 4)$. The subgraph $H_3 = \{v_4, v_5, v_6\}$ is not a skyline community, because it is contained in $H_2 = \{v_2, v_4, v_5, v_6\}$ which has the same f values as H_3. The subgraph $H_4 = \{v_2, v_3, v_4, v_5, v_6\}$ is not a skyline community, as $f(H_4) = (6, 8, 3)$ is dominated by H_1 and H_2.

In [126], the authors first developed an efficient algorithm, called SkylineComm2D, to find all the skyline communities in the 2D case, i.e., $d = 2$. The time complexity of SkylineComm2D is $O(s(m + n))$ where s denotes the number of 2D skyline communities (i.e., the answer size), and the space complexity of SkylineComm2D is $O(m + n + s)$, which is linear w.r.t. the graph and answer size. To handle the high-dimensional case (i.e., $d \geq 3$), the authors proposed a space-partition algorithm to find the skyline communities efficiently. Two novel features of the space-partition algorithm are that (1) its worst-case time complexity is dependent mainly on the answer size; thus, it is very efficient when the answer size is not very large; and (2) it is able to progressively output the skyline communities during the execution of the algorithm, and thus, it is useful for applications that only require part of skyline communities.

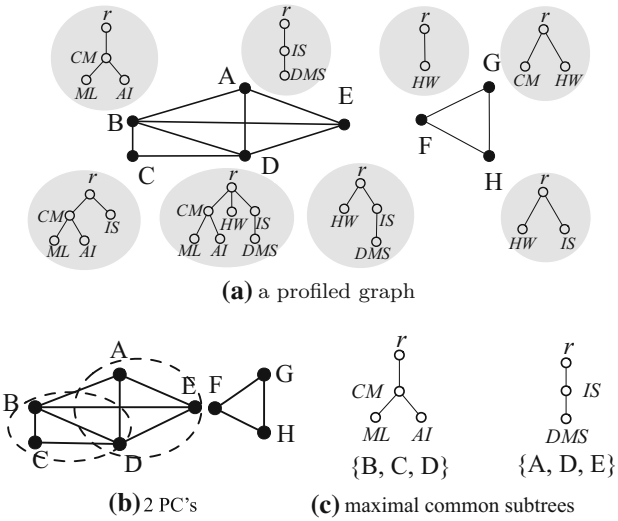


Fig. 13 A profiled graph and two PC's [31]

3.7 Profile-based attributed graphs

A profiled-based attributed graph, or profiled graph, is an undirected graph $G(V, E)$ with vertex set V and edge set E, in which each vertex is associated with *profile*. The profile of a vertex $v \in V$ is a set of keywords $T(v)$ that are arranged in a hierarchical manner, called a P-tree. Typical such attributes are users' affiliation, expertise, locations, etc. Profiled graphs are prevalent in knowledge bases and social media.

Figure 13a depicts a profiled graph. For instance, vertex D has a hierarchically organized profile that describes his expertise in computer science (e.g., abbreviation AI means artificial intelligence) by following the *ACM Computing Classification System (CCS)*.[3]

Chen et al. [31] investigated the problem of *profiled community search* (or PCS) on profiled graphs. To capture the profile-based cohesiveness, they introduced the concept of

[3] ACM CCS: http://www.acm.org/publications/class-2012.

"maximal common subtree," which describes the commonality of vertices' profile.

Definition 20 (*Maximal common subtree*) Given a profiled graph G, the maximal common subtree of G, denoted by $\mathcal{M}(G)$, holds the properties: (1) $\forall v \in G$, $\mathcal{M}(G) \subseteq T(v)$; (2) there exists no other common subtree $\mathcal{M}'(G)$ such that $\mathcal{M}(G) \subseteq \mathcal{M}'(G)$.

Problem 14 (*PCS*) Given a profiled graph $G(V, E)$, a positive integer k, a query node $q \in G$, find a set $\mathcal{G}$ of graphs, such that $\forall G_q \in \mathcal{G}$, and following properties hold:

1. *Connectivity* G_q is connected and contains q;
2. *Structure cohesiveness* $\forall v \in G_q$, $\deg_{G_q}(v) \geq k$;
3. *Profile cohesiveness* There exists no other $G'_q \subseteq G$ satisfying the above two constraints, such that $\mathcal{M}(G_q) \subseteq \mathcal{M}(G'_q)$.
4. *Maximal structure* There exists no other subgraph G'_q satisfying the above properties, such that $G_q \subset G'_q$ and $\mathcal{M}(G_q) = \mathcal{M}(G'_q)$;

The subgraph G_q is called a *profiled community* (or PC). In Problem 14, the first two properties guarantee the structure cohesiveness. The *profile cohesiveness* captures the maximal shared profile among all vertices in G_q. The *maximal structure* property aims to retrieve all qualified vertices in the community. For instance, in Fig. 13a, if $q = D$, $k = 2$, then two PC's and their maximal common subtrees are, respectively, shown in Fig. 13b, c. These two common subtree sufficiently reflects the "theme" of the community. For example, in the PC grouped by vertices {B, C, D}, all the researchers involved share interest in ML (i.e., machine learning) and artificial intelligence, whereas for the other PC, the researchers are all interested in other research domains.

The PCS problem is technically challenging, because the number of subtrees of a P-tree could be exponentially large, and thus, enumerating all of them is impractical. To answer the PCS query efficiently, Chen et al. [31] introduced the anti-monotonicity property, based on which the query can be performed much faster. To further improve efficiency, they developed the *CP-tree* index, which systematically organizes all the graph vertices and their P-trees into a compact tree structure. The CP-tree index enables the development of two fast PC discovery algorithms.

3.8 Discussions

In this section, we review CS studies that use the k-core model. For simple graphs, we can divide them into two groups, where the first group [15,46,175] focuses on undirected graphs while the second group [66] only considers directed graphs. In particular, for the first group, the first work [175] returns the maximal k-core containing the query vertex, while communities of the other two studies [15,46] may not be the maximal k-core or with size constraints.

For attributed graphs, all the corresponding CS studies take both link relationship and attributes into consideration, because the attributes often make communities more meaningful and easy for interpretation. As a result, the solutions for different attributed graphs are different. Generally, both online and index-based algorithms are developed for CS on these graphs. The index-based algorithms run faster, but incur an offline computational cost.

In practice, the query users can select the CS solutions based on the graph models since the community models are formulated based on the graph models. For example, for keyword-based attributed graphs, ACQ can be considered. Meanwhile, if the CS queries are executed with high frequency, the index-based algorithms should be better choices as they are faster, although they have to build the index in an offline manner.

4 *K*-truss-based community search

In this section, we review CS works that use the k-truss as structure cohesiveness metrics, including triangle-connected truss community [6,98], closest truss community [101], attribute-driven truss community [102], and weighted truss community [216]. In the following, we will introduce the community models and compare their algorithms and applications.

4.1 Simple graphs

In a simple and undirected graph $G(V, E)$, triangle-connected k-truss community model proposed by Huang et al. [98] finds all communities containing a query vertex. We first introduce the definitions of k-truss and triangle connectivity and then present the model below.

A k-truss is the largest subgraph H of G such that every edge is contained in at least $k-2$ triangles in H, i.e., $\forall e \in E$, its support $\text{sup}(e, H) \geq k - 2$ by Definition 4. However, k-truss may be disconnected with several components in a graph, which is similar with k-core. Consider the graph G in Fig. 14. There exist two components in the shaded regions to form the 4-truss of G, which are obviously disconnected. Disconnected subgraphs are insufficient to define a cohesive and meaningful community.

To address the disconnectivity problem of k-truss, *triangle connectivity* is imposed on top of the k-truss in [98]. Given two triangles $\triangle_1$ and $\triangle_2$ in G, $\triangle_1$ and $\triangle_2$ are said to be adjacent if they share a common edge. Then, for two edges $e_1, e_2 \in E$, e_1 and e_2 are *triangle-connected* if they either belong to the same triangle, or are reachable from each

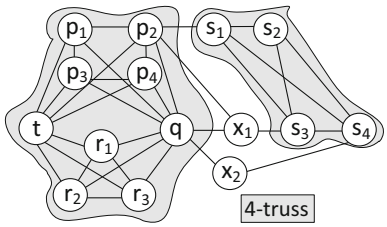
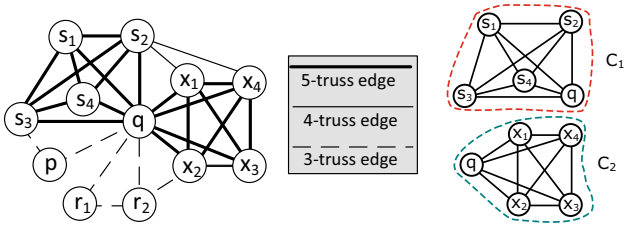


Fig. 14 Example of 4-truss with 2 disconnected components



(a) Graph G **(b)** TTCs

Fig. 15 An example of TTC search. Here, $k = 5$

other through a series of adjacent triangles. In other words, $\exists \triangle_1, \triangle_2$ such that $e_1 \in \triangle_1$, $e_2 \in \triangle_2$, then either $\triangle_1 = \triangle_2$, or $\triangle_1$ is triangle-connected with $\triangle_2$. Based on the k-truss and triangle connectivity, the problem of triangle-connected truss community (TTC) search is formulated as follows:

Problem 15 (*TTC search*) Given an undirected simple graph $G(V, E)$, a query vertex $q \in V$, and an integer $k \geq 2$, return all subgraphs $H \subseteq G$ which satisfies the following three properties:

1. *Structure Cohesiveness* H contains the query vertex q such that $\forall e \in E(H)$, $\text{sup}(e, H) \geq (k-2)$;
2. *Triangle Connectivity* $\forall e_1, e_2 \in E(H)$, e_1 and e_2 are triangle-connected;
3. *Maximal Subgraph* H is the maximal subgraph of G satisfying Properties 1 and 2.

TTC model imposes the triangle connectivity requirement in Property 2 to ensure the discovered communities are connected. This requirement also allows the query vertex to participate in multiple overlapping communities. For example, consider the graph G in Fig. 15a, a query vertex q, and parameter $k = 5$. Two triangle-connected 5-truss communities C_1 and C_2 containing vertex q are shown in Fig. 15b. As the edges in C_1 cannot reach the edges in C_2 through adjacent triangles, C_1 and C_2 cannot merge as one large community. This is reasonable, as there are few connections between the two vertex sets $\{s_1, s_2, s_3, s_4\}$ and $\{x_1, x_2, x_3, x_4\}$.

Thanks to k-trusses, truss-based community model inherits several good structural properties of k-trusses [98], such as $(k-1)$-edge-connected, bounded diameter, and hierarchical structure. Specifically, the diameter of a k-truss community H with $|V(H)|$ vertices is no larger than $\lfloor \frac{2|V(H)|-2}{k} \rfloor$ [41]. Small diameter has been considered as an important feature of a good community in [52]. Second, a k-truss community is $(k-1)$-edge-connected [41], i.e., the community keeps connected whenever fewer than $k-1$ edges are deleted [74]. Third, truss-based communities have a strong decomposability for analyzing large-scale networks at different levels of granularity.

To tackle the problem of TTC search, there exists one online search algorithm [98], and two index-based search algorithms, which are, respectively, based on TCP-index [98] and EquiTruss [6]. In the following, we briefly introduce the key ideas of these algorithms one by one.

• **Online search algorithm** [98] Huang et al. [98] proposed an online query algorithm to process a TTC query on a graph G. The algorithm firstly applies the truss decomposition [184] on graph G to compute the trussness of all edges in G. By the community definition, it starts from the query vertex q and checks an incident edge of $(q, v) \in E$ with trussness $\tau((q, v)) \geq k$ to search triangle-connected truss communities. It explores all edges that are triangle-connected to (q, v) and having trussness no less than k in a BFS manner. This process iterates until all incident edges of q have been processed. Finally, a set of k-truss communities containing q are returned.

However, this online search algorithm may incur a large number of wasteful edge accesses on checking disqualified edges, which is inefficient.

• **TCP-index-based search algorithm** [98] To avoid the computational issues mentioned above, Huang et al. [98] designed a triangle connectivity-preserving index (TCP-index). TCP-index preserves the truss number and triangle adjacency relationship in a compact tree-shape index and supports the query of k-truss community in linear time with respect to the community size, which is optimal. Given a graph G, it needs to construct a TCP-index for each vertex in G, which is denoted as $\mathcal{T}_x$. Take a vertex x as an example for TCP-index construction. Essentially, T_x is the maximum spanning forest of G_x, where G_x is the induced subgraph of G by vertex set of x's neighbors as $N(x)$. For each edge $(y, z) \in E(G_x)$, a weight $w(y, z) = \min\{\tau((x, y)), \tau((x, z)), \tau((y, z))\}$ is assigned to it, which indicates that $\triangle_{xyz}$ can appear only in k-truss communities where $k \leq w(y, z)$. Figure 16 presents a TCP-index T_q for vertex q in graph G shown in Fig. 15a. Vertices x_1, x_2, x_3, and x_4 are connected via the weighted edges of 5, indicating these vertices present in a triangle-connected 5-truss community.

Based on the TCP-index, an efficient query processing algorithm is developed for CTC search. Assume that we want to query 5-truss communities containing a query vertex q in G in Fig. 15a, we first visit an incident edge on q, say

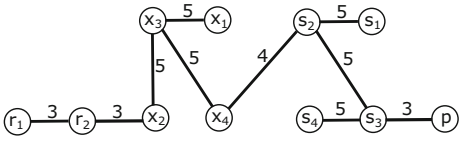


Fig. 16 TCP-index T_q for vertex q of G in Figure 15a

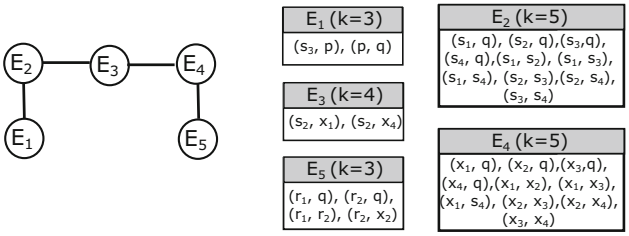


Fig. 17 EquiTruss index for graph G in Fig. 15a

(q, x_1), where $\tau((q, x_1)) = 5$. From TCP-index $\mathcal{T}_q$ in Fig. 16, we retrieve the vertex set $\{x_1, x_2, x_3, x_4\}$ belong to the same 5-truss community. Since $\mathcal{T}_q$ is a spanning forest, which does not keep all the edges between the vertices, the query processing algorithm then performs the reverse operations on the TCP-index for each vertex x_1, x_2, x_3, x_4 and gets the complete 5-truss community.

Remarkably, the TCP-index supports the k-truss community query in the optimal time, which accesses each edge in the answer community exactly twice [98]. Meanwhile, the TCP-index can be constructed in $O(\sum_{(u,v)\in E} \min\{deg_G(u), deg_G(v)\})$ time and stored in $O(m)$ space.

• **EquiTruss-index-based search algorithm** [6] To further improve efficiency, Akbas and Zhao [6] proposed a novel indexing technique of k-truss equivalence, to represent the triangle connectivity and k-truss cohesiveness in the triangle-connected truss communities.

We introduce the definition of k-truss equivalence as follows: Given two edges $e_1, e_2 \in E$, e_1 and e_2 are k-truss equivalence, if and only if (1) $\tau(e_1) = \tau(e_2) = k$, and (2) e_1 and e_2 are triangle-connected via a series of triangles in a k-truss.

The index of EquiTruss, a summarized graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is constructed based on k-truss equivalence. According to k-truss equivalence, all edges of a given graph G are partitioned into a series of mutually exclusive equivalence classes. Each class represents a TTC. A super-node $E_i \in \mathcal{V}$ represents a distinct equivalence class C_i where $e \in G$, and a super-edge $(E_i, E_j) \in \mathcal{E}$, where $E_i, E_j \in \mathcal{V}$, indicates that the two equivalence classes are triangle-connected; that is, there exists two edges $e_1 \in E_i$ and $e_2 \in E_j$, s.t., e_1 and e_2 are k-truss triangle adjacent. Note that EquiTruss is a community-preserving graph summary, where all triangle-connected k-truss communities are comprehensively recorded in the super-nodes, and the triangle connectivity across different communities is exactly encoded in super-edges. In this way, EquiTruss keeps records of all the information critical to community search. Moreover, each edge e is recorded in exactly one super-node, which represents its k-truss equivalence class, C_e. Compared with TCP-index, which may redundantly maintain an edge in multiple maximum spanning forests, EquiTruss is significantly more succinct and space efficient [6].

For example, Fig. 17 shows an EquiTruss index for graph G in Fig. 15a. It has 5 super-nodes representing the k-truss equivalence classes for edges in G, as tabulated in Fig. 17. The super-node E_2 represents a 5-truss community with 10 edges: All these 10 edges are triangle-connected and belong to the 5-truss. In addition, there exist 5 super-edges in EquiTruss, which represents the triangle connectivity between super-nodes (triangle-connected k-truss communities).

The EquiTruss-index-based community search algorithm is described as follows: Finding triangle-connected communities containing vertex q can be carried out directly on EquiTruss, without the access to graph G. First, the algorithm finds all super-nodes containing q. A hash structure can help quick identification of such super-nodes. Next, starting from these super-nodes, we can traverse $\mathcal{G}$ in a BFS manner. For each unvisited neighboring super-nodes E^* with $\tau(E^*) \geq k$, the edges within E^* will be included into the k-truss community. The algorithm outputs all the discovered communities containing q. Consider the graph G in Fig. 15a, $k = 5$ and query vertex q. Based on the EquiTruss index in Fig. 15a, we first find two super-nodes E_2 and E_4 containing q with trussness no less than 5. Super-nodes E_2 and E_4 are disconnected via any super-edges. Then, E_2 and E_4 can be, respectively, output as two communities. Compared to TCP-index, EquiTruss-index-based query processing only needs to access each edge exactly once, which is more efficient [6].

4.2 Closest truss community search

In this section, we introduce a new truss-based community model for multiple query vertices. Although the triangle-connected k-truss community model works well to find all overlapping communities containing a single query vertex q, it may fail to discover any community for multiple query vertices, due to the strict requirement of triangle connectivity constraint. For example, for the graph G in Fig. 18a and query vertices $Q = \{v_4, q_3, p_1\}$, the above k-truss community model cannot find a qualified community for any k, since the edges (v_4, q_3) and (q_3, p_1) are not triangle-connected in any k-truss. To address this limitation, Huang et al. [101] studied the problem of closest truss community (CTC) search for multiple query vertices as follows:

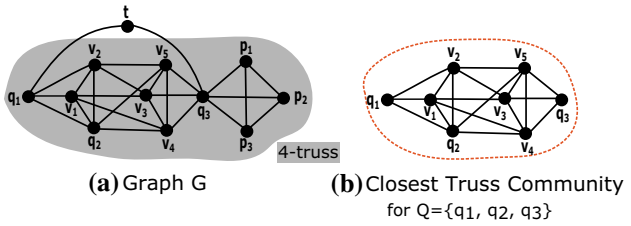


Fig. 18 Closest truss community example

Problem 16 (*CTC search*) Given a graph G and a set of query vertices Q, return a subgraph $H \subseteq G$ as a closest truss community (CTC), satisfying the following two properties:

1. Connected k-truss. H is containing Q and a connected k-truss with the largest k, i.e., $Q \subseteq H$ and $\forall e \in E(H)$, $\text{sup}(e, H) \geq k - 2$;
2. Smallest Diameter. H is a subgraph of smallest diameter satisfying Property 1.

Property 1 requires that the closest community contains the query vertices Q which are densely connected. In addition, to ensure every vertex included in the community is close to query vertices and other vertices in the community, Property 2 uses graph diameter to measure the closeness of all vertices in the community. Moreover, the CTC model can avoid the free rider effect issue, that is, vertices far away from query vertices and irrelevant to them are included in the detected community [101].

Consider the graph G in Fig. 18a, and $Q = \{q_1, q_2, q_3\}$; the subgraph in the region shaded gray is a 4-truss containing Q with the largest trussness and has a diameter of 4. Figure 18b shows another 4-truss containing Q but not p_1, p_2, p_3, and its diameter is 3. It can be verified that this is indeed the CTC, which is the 4-truss containing the query vertices Q with the smallest diameter.

The problem of CTC search is very challenging. A connected k-truss with the largest k containing query vertices can be found in polynomial time. However, finding such a k-truss with the minimum diameter is NP-hard [101]. Moreover, it is even hard to approximate the CTC-Problem within a factor better than 2. Here, the approximation is with regard to the minimum diameter.

To find the closest truss community, a simple but effective greedy algorithm is proposed in [101]. The method uses a greedy strategy for finding a CTC that delivers a 2-approximation to the optimal solution, thus essentially matching the lower bound. Here is an overview of this algorithm. First, given a graph G and query vertices Q, we find a maximal connected k-truss, denoted G_0, containing Q and having the largest trussness. As G_0 may have a large diameter, we iteratively remove vertices far away from the query vertices, while maintaining the trussness of the remainder subgraph at k. Actually, this algorithm can find a connected k-truss with the largest k containing query vertices, which achieves the smallest query distance in optimal. According to the inequality of query distance and graph diameter, this answer is a 2-approximation to CTC [101].

In order to improve the efficiency of CTC search, Huang et. al proposed two new techniques of bulk deletion and local exploration. One of them is based on bulk deletion of vertices far away from query vertices. This speeds up the pruning process, by deleting at least k vertices in batch, to achieve quick termination while sacrificing some approximation ratio. Second, they also propose a heuristic strategy of local exploration to quickly find the closest truss community in the local neighborhood of query vertices. The key idea is as follows: It first forms a Steiner tree to connect all query vertices and then expand the Steiner tree to a k-truss with the largest k by involving the local neighborhood of the query vertices. Finally, to reduce the diameter, it iteratively removes the furthest vertices from this k-truss using the bulk deletion.

4.3 Keyword-based attributed graphs

In this section, we introduce a k-truss-based community search model on a keyword-based attributed graph where vertices are associated with a set of keywords. Huang and Lakshmanan [102] proposed an attribute-driven truss community model, denoted by ATC, which finds the densely interconnected communities containing query vertices with similar query attributes. ATC is equipped with two key components of (k, d)-truss and an attribute score function.

To capture dense cohesiveness and low communication cost, ATC builds upon a notion of dense and tight substructure called (k, d)-truss. A (k, d)-truss requires that every edge is contained at least $(k - 2)$ triangles, and the communication cost between the vertices of H and the query vertices is no greater than d. By definition, the cohesiveness of a (k, d)-truss increases with k, and its proximity to query vertices increases with decreasing d. For instance, H in Fig. 19b for $V_q = \{q_1, q_2\}$ is a (k, d)-truss with $k = 4$ and $d = 2$.

To measure the goodness of an attributed community w.r.t. attribute coverage and correlation, an attribute score function is developed for ATC. Let $\mathsf{f}(H, W_q)$ be the attribute score of

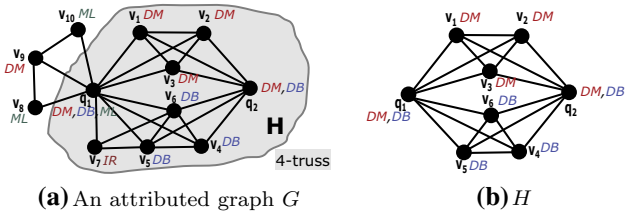


Fig. 19 An example of attributed truss community search with query vertices $V_q = \{q_1, q_2\}$ and query attributes $W_q = \{$ 'DB', 'DM'$\}$. Here, $k = 4$

community H w.r.t. query attributes W_q. Then, $\mathsf{f}(H, W_q) = \sum_{w \in W_q} \frac{\mathsf{score}(H,w)^2}{|V(H)|}$, where $\mathsf{score}(H, w) = |V_w \cap V(H)|$ is the number of vertices covering query attribute w. The function $\mathsf{f}(H, W_q)$ satisfies three important properties as follows: *Property* 1 The more query attributes that are covered by some vertices of H, the higher score of $\mathsf{f}(H, W_q)$. The rationale is obvious; *Property* 2 The more vertices that contain an attribute $w \in W_q$, the higher the contribution of w should be toward the overall score $\mathsf{f}(H, W_q)$. The intuition is that attributes that are covered by more vertices of H signify homogeneity within the community w.r.t. shared query attributes; *Property* 3 The more vertices of H that are irrelevant to the query, the lower the score $\mathsf{f}(H, W_q)$. The more query attributes a community has that are shared by more of its vertices, the higher its attribute score. For example, consider the query $Q = (\{q_1\}, \{\text{'DB', 'DM'}\})$ on the running example graph of Fig. 19a. Intuitively, we can see that H has 5 vertices covering 'DB' and 'DM' each and also has the highest attribute score (i.e., $\mathsf{f}(H, W_q) = \frac{5^2}{8} + \frac{5^2}{8} = 6.25$), which is the attributed truss community. On the other hand, the induced subgraph of G by vertices $\{q_1, q_2, v_1, v_2, v_3\}$ and $\{q_1, q_2, v_4, v_5, v_6\}$ is mainly focused in one area ('DB' or 'DM'), achieving the score of 5.8.

Based on the (k, d)-truss and $\mathsf{f}(H, W_q)$, Huang et al. [102] studied the ATC problem.

Problem 17 (*ATC search*) Given a graph G, a query $Q = (V_q, W_q)$, and two numbers k and d, return an attributed truss community (ATC) H, satisfying the following properties:

1. H is a (k, d)-truss containing V_q.
2. H has the maximum attribute score $\mathsf{f}(H, W_q)$ among all subgraphs satisfying property 1.

Theoretical proofs show that ATC search is NP-hard [102], which shows the challenge for computation. To help efficiently processing of ATC queries, [102] presents a greedy algorithmic framework for finding an ATC in a top-down search manner. The general ideas of this algorithm have three steps. First, it finds the maximal (k, d)-truss of original graph G as a candidate. Second, it iteratively removes vertices with the smallest "attribute marginal gains" from the candidate graph and maintains the remaining graph as a (k, d)-truss, until no longer possible. The removed vertices have the smallest contribution to attribute score function $\mathsf{f}(H, W_q)$. Finally, it returns a (k, d)-truss with the maximum attribute score among all generated candidate graphs as the answer. If there exists more than one (k, d)-truss with the maximum attribute scores, the algorithms just outputs one answer.

To further improve the search efficiency while ensuring high quality, a novel index called attributed truss index (ATindex) is developed. The ATindex consists of two components: structural trussness and attribute trussness, which maintain known graph structure and attribute information. ATindex can quickly identify a good candidate of (k, d)-truss to the answer. In addition, another technique of local exploration is applied for efficiently detecting a small neighborhood subgraph around query vertices, which tends to be densely and closely connected with the query attributes.

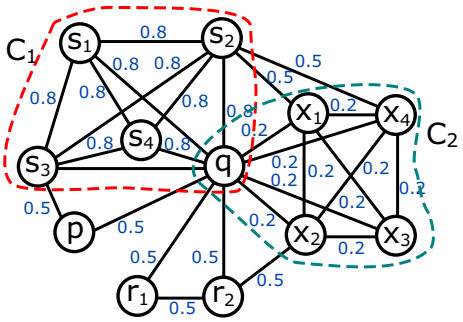


Fig. 20 An example of weighted truss community search

4.4 Weight-based attributed graphs

In this section, we consider an undirected weighted graph $G = (V, E, W)$, where the weight of e is denoted by $w(e) \in W$, representing the importance between vertices u and v. Weighted graphs naturally exist in the real-world applications. For instance, in the collaboration network, the edge weights may represent the number of co-authored articles between two authors. Figure 20 depicts an undirected weighted graph G, e.g., edge (q, s_1) has a weight of 0.8. Taking the edge weights into consideration, community search on weighted graphs can find communities capturing more semantics. Zheng et al. [216] proposed a model of weighted truss community (WTC):

Definition 21 (*Weighted Truss Community*) Given an undirected weighted graph $G = (V, E, W)$, and a positive integer k, a weighted k-truss community is an induced subgraph $H \subseteq G$, and the following properties hold:

1. *Connectivity* $\forall e_1, e_2 \in E(H)$, e_1 and e_2 are triangle-connected in H;
2. *Cohesiveness* $\forall e \in E(H)$, $\sup_H(e) \geq k - 2$;
3. *Maximal Structure* H is a maximal induced subgraph that satisfies Properties 1 and 2.

In the weighted k-truss community model, Property 1 adopts the same constraint of triangle connectivity as other k-truss community models [98]; Property 2 requires the community to satisfy the structure of k-truss; Property 3 can guarantee the property of maximal structure in the weighted k-truss community. Given a weighted truss community H, the community weight of H is defined as $w(H) =$

$\min_{e \in E(H)} w(e)$. To discover the communities with large weights, Zheng et al. [216] investigated the problem of weighted truss community (WTC) search.

Problem 18 (*WTC search*) Given an undirected weighted graph $G(V, E, W)$, and parameters k and r, find the top-r weighted k-truss communities H with the largest weights $w(H)$.

Consider a weighted graph G in Fig. 20, $k = 5$, and $r = 1$. The community C_1 shown in Fig. 20 has the weight $w(C_1) = 0.8$, which is larger than the weight of community C_2 as $w(C_2) = 0.2$. Thus, C_1 is the answer of WTC search with the largest weight.

Straightforward to enumerate all weighted k-truss communities to find the r communities with the largest community weights is impractical in large graphs. To speed up the search efficiency, an index structure called KEP-Index is designed. KEP-Index is built upon the observation that all the communities can be organized into a tree-shaped structure. This is because all the weighted k-truss communities form a partial order relationship for each value of k. By indexing all the pre-computed weighted k-truss communities in a tree-shaped structure, WTC search can be done in the linear time w.r.t. the answer size, which is optimal.

4.5 Discussions

Generally, the k-truss-based CS solutions on simple graphs can be divided into two groups, where the first group [6,98] computes the k-truss community, while the second group [101] aims to find closest communities. In the first group, Akbas et al. [6] improved the efficiency of [98] by developing a novel index. For attributed graphs, there are two CS solutions, which consider keywords [102] and influence values [216], respectively. For all these studies above, both online and index-based algorithm are developed.

For practitioners, to perform CS, we would like to offer some suggestions: (1) We should figure out the type of graph (e.g., simple graphs and attributed graphs) in the application. (2) For simple graphs, there are two community models, i.e., triangle-connected model and closest model. Generally, the triangle-connected model [6,98] is suitable for one single query vertex to discover all overlapping communities containing it, while the closest model [101] is suitable to discover one closest community containing multiple query vertices, which is not strict to one query vertex. Moreover, triangle connectivity is weaker than the optimization metric of minimum diameter. According to our experience in the real-world applications, the discovered closest community has smaller graph size than triangle-connected truss community. (3) For triangle-connected model [6,98], the index-based algorithm in [6] is faster than that in [98].

5 *K*-clique-based community search

In this section, we survey CS solutions that use k-clique or its variants to capture the structure cohesiveness. We first briefly introduce the k-clique model and its variants in Sect. 5.1. Then, we present CS solutions using k-clique component and k-plex models in Sects. 5.3 and 5.3 . After that, we discuss the most influential CS using k-clique in Sect. 5.4. Finally, we discuss these studies in Sect. 5.5.

5.1 *K*-clique and its variants

Recall that by Definition 6, a k-clique is a complete graph with k vertices where there is an edge between every pair of vertices. The k-clique model has been widely used for the overlapping community detection (e.g., [4,151]). As the condition of k-clique is strict, some relaxed variants such as γ-quasi-k-clique [23,45] and k-plex [171] are proposed to identify cohesive subgraphs. Below are detailed definitions.

Definition 22 (γ-*quasi-k-clique* [23,45]) A γ-quasi-k-clique is a graph with k vertices and at least $\lfloor \gamma \frac{k(k-1)}{2} \rfloor$ edges, where $0 \leq \gamma \leq 1$.

When $\gamma = 1$, the corresponding γ-quasi-k-clique is a k-clique. We can tune the desired cohesiveness of the k vertices by varying γ value.

Definition 23 (*k-plex* [171]) A graph $G(V, E)$ is a k-plex, if for each vertex $v \in V$, v has at least $|V| - k$ neighbors in G, where $1 \leq k \leq |V|$.

When $k = 1$, the k-plex is exactly a k-clique. Clearly, by setting a smaller value of k, we can obtain a more cohesive k-plex. The problem of finding a k-plex from a given graph for an integer k is NP-hard [14].

Another way to relax the constraint of k-clique is to consider the connection of two vertices.

Definition 24 (*kr-clique* [125]) Given a graph G and two integers k and r, a kr-clique S is an induced subgraph of G such that: (1) the number of vertices in S is at least k; and (2) any two vertices in S can reach each other within r hops.

Clearly, the problem of finding kr-clique is NP-hard because kr-clique is a k-clique when $r = 1$.

5.2 *K*-clique-based community search

In Sect. 5.2.1, we introduce the seminar work on overlapping community detection [151], in which the k-clique component is proposed. Section 5.2.2 presents the community search algorithm based on the relaxation of k-clique component, while Sect. 5.2.3 studies the densest k-clique community search.

5.2.1 *K*-clique-based community

In [151], Palla *et* al. showed that that many real networks are characterized by well-defined overlapping communities. For instance, a person may belong to three different communities related to school, hobby, and family. For a given graph G, a *k-clique graph* G_k can be derived where each node is a k-clique in G and there is an edge if two nodes (k-cliques) are adjacent, i.e., they share $k-1$ vertices in G. Then, the k-clique communities are the union of all adjacent k-cliques, which are defined as follows:

Definition 25 (*k-clique component*) Let C denote a connected component in the k-clique graph, then a *k-clique component* is the union of all k-cliques represented by vertices in C.

One may explore the communities of the graph based on the k-cliques and their adjacency, and a graph vertex may belong to several communities. Efficient k-clique component detection algorithm is presented in [4]. Particularly, considering that each k-clique must be contained by at least one maximal clique, they first identify all maximal cliques of the network and then enumerate the communities by carrying out a standard component analysis of the clique overlap matrix.

5.2.2 *K*-clique-based community search

In [45], Cui *et* al. showed that there are two shortcomings in the k-clique community model: (1) there are overwhelming number of k-cliques communities in real-life graphs; and (2) the k-clique constraint and the definition of adjacent (i.e., sharing $k-1$ common vertices) are not flexible in practice. To address these two shortcomings, they proposed an online community search (OCS) problem. Instead of enumerating all communities, they focused on the search of the communities containing a given query vertex q. They relaxed the k-clique adjacent from $k-1$ common vertices to α vertices, namely α-adjacency. They also relaxed k-clique model to γ-quasi-k-clique model (Definition 22). By doing this, the k-clique components in the k-clique communities are relaxed to the γ-quasi-k-clique components. Below is the formal problem definition.

Problem 19 (*(α,γ)-OCS*) Given an undirected simple graph $G(V, E)$, a query vertex $q \in V$, and an integer k, an integer $\alpha \leq k-1$, and a real value γ with $0 \leq \gamma \leq 1$, find all γ-quasi-k-clique components containing query vertex q.

Clearly, a k-clique component search is a special case of (α, γ)-OCS with $\alpha = k-1$ and $\gamma = 1$. By reducing to k-clique decision problem, it is shown in [45] that the (α, γ)-OCS problem is $\#P$-complete. It is shown that the density of each community in (α, γ)-OCS is at least $2\max\{0, \min\{f(1), f(\alpha)\}\}$ where $f(x) = \frac{\gamma\binom{k}{2}\binom{k-x}{2}}{x}$. Both exact and approximate solutions are proposed in [45]. A naive algorithm for exact solution is to enumerate all γ-quasi-k-cliques containing the query vertex q and then compute the γ-quasi-k-clique components based on the α-adjacency. To avoid enumerating cliques belonging to none of the valid communities, a new computing framework is proposed to check the adjacency when a clique is discovered. By carefully maintaining the visit status of each clique, authors further optimize the searching cost. Authors also proposed an approximate solution. To reduce the search space, the approximate algorithm only enumerates an unvisited clique which contains at least one new vertex not contained by any existing community. A heuristic is proposed to choose a vertex sequence such that the resulting clique sequence is short, leading to a good approximation solution.

5.2.3 Densest clique percolation community search

Following the k-clique community model in [151], Yuan et al. studied the problem of densest clique percolation community search [205], where a *k-clique percolation community* (KCPC) is a *k-clique component* in [151]. In particular, they aimed to find the k-clique percolation community with the maximum k value that contains a given set of query vertices.

Problem 20 Given an undirected simple graph $G(V, E)$ and a set of query vertex $Q \subseteq V$, the problem of the *densest clique percolation community* (DCPC) search is to find the k-clique component with the maximum k value that contains all the vertices in Q.

Figure 21 in [205] illustrates a part of the collaboration network in DBLP, in which each vertex represents an author and each edge indicates the co-author relationship between two authors. G_1 is a 4-clique percolation community as it is a maximal union of five adjacent 4-cliques: $\{v_{14}, v_{15}, v_{16}, v_{17}\}$, $\{v_{14}, v_{15}, v_{16}, v_{18}\}$, $\{v_{14}, v_{15}, v_{17}, v_{18}\}$, $\{v_{14}, v_{16}, v_{17}, v_{18}\}$, $\{v_{15}, v_{16}, v_{17}, v_{18}\}$, and any two 4-cliques share 3 nodes. Similarly, G_2 is also a 4-clique percolation community. G_1 overlaps G_2 with nodes v_{14}, v_{15}. Given a query $q = \{v_9, v_{18}\}$, the densest clique percolation community of q is the 3-clique percolation community G_3 since G_3 is the k-clique percolation community with maximum k value that contains v_9 and v_{18}.

A baseline solution is to start from the maximal possible k value and check if there is a KCPC by applying the k-clique component detection algorithm in [151]. If there is no KCPC detected, the k value will be decreased by one until a KCPC is detected. To efficiently support online DCPC search, an index-based approach is developed in [205]. Particularly, based on the observation that a k-clique component can be treated as a union of *maximal* cliques, they take maximal cliques as building blocks of k-clique components and

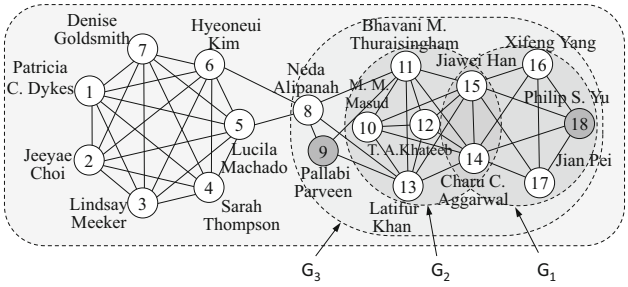


Fig. 21 Illustration of DCPC search [205]

propose a tree structure named *clique adjacency tree* which can efficiently identify the k-clique components for a given k value. The authors further developed a new tree structure named *ordered adjacency tree* such that only the subtrees related to the query vertices will be explored. Together with maximal cliques and their inverted indexes, a compact index structure named DCPC Index is proposed to support efficient DCPC queries.

5.3 *K*-plex-based community search

5.3.1 Social group query (SGQ)

Problem 21 presents SGQ, which was designed for suggesting attendees in activity planning [195].

Problem 21 (*SGQ*) Given a simple undirected graph $G(V, E)$, an activity initiator $q \in V$, three integers p, s, and k, return a set F of vertices from G such that the following properties hold:

1. $|F| = p$;
2. The length of the minimum distance path between v and q, $d_{v,q}$, is at most s;
3. Each vertex $v \in F$ is allowed to share no edges with at most k other vertices in F;
4. The total social distance $\Sigma_{v \in F} d_{v,q}$ is minimized.

In Problem 21, Property 1 controls the expected number of attendees in the activity; Property 2 specifies a radius constraints which requires each attendee is close to q in the graph G; Property 3 requires that each attendee is acquainted with other attendees by following the k-plex model; Property 4 ensures that the returned group is the most compact one among all the groups satisfying all the above properties.

The SGQ problem is computationally challenging because it is NP-hard, which can be proved by a reduction from the k-plex problem [14]. To answer SGQ, Yang et al. [195] proposed an efficient solution `SGSelect`. The idea is that we can first extract a subgraph $H \subseteq G$ by using the radius constraint. Then, starting from q, we iteratively explore vertices in H to derive the optimal solution. In each iteration, we can keep track of a set of vertices that satisfy the constraint of k, until the set has p vertices. To further speedup this process, some effective pruning criteria have been developed. For example, to choose vertices, we can give high priorities for vertices that may significantly increase the total social distance. Also, during the search process, we can prune vertices which would not lead to eventual answer by considering the acquaintance constraint p and social radius constraint s.

In addition, Yang et al. [195] studied another query, called social-temporal group query (STGQ), which generalizes SGQ by considering the available time of each candidate attendee. In specific, it finds a group of vertices satisfying: (1) all constraints in an SGQ; and (2) all the attendees are available in a time period $[t, t+\delta_t]$, where t is time slot and δ_t is query parameter. The STGQ problem is also NP-hard and some efficient solutions are developed. For details, please refer to [195].

5.3.2 Maximum *k*-plex community query (MCKPQ)

In [187], Wang et al. proposed and studied the maximum k-plex community query (MCKPQ):

Problem 22 (*MCKPQ*) Given a simple undirected graph $G(V, E)$, a set of query vertices $Q \in V$, an integer k, return a subgraph $G_Q(V_Q, E_Q) \subseteq G(V, E)$ such that the following properties hold:

1. *Connectivity* G_Q is connected and contains Q;
2. *Structure cohesiveness* G_Q is a k-plex;
3. *Maximal structure* There exists no other $G'_Q \subseteq G$ satisfying the above properties and $G_Q \subset G'_Q$.

A good property of MCKPQ is that the communities returned by an MCKPQ can avoid the free rider effect, which is introduced and discussed in Sect. 4. Nevertheless, the MCKPQ problem is very computationally challenging, because it is NP-complete, which can be proved by a reduction from the k-plex problem [14]. Moreover, it is hard to approximate for MCKPQ problem in polynomial time within a factor $n^{1-\epsilon}$.

A basic solution to the MCKPQ problem is to use the generate-and-verify method, which enumerates all the k-plexes in the whole search space and then returns the one with the largest size. Obviously, this method is too expensive and impractical for large graphs. To alleviate this issue, Wang et al. developed a more advanced method based on the branch-and-bound paradigm with some effective pruning criteria and a heuristic method which performs fast but has no theoretical guarantee [187]. We skip the details due to space limitation.

5.4 Most influential community search

In [125], Li et al. proposed the problem of most influential community search, which aim to find the most influential cohesive subgraph. The concept of kr-clique community (Definition 24) is proposed to capture the cohesiveness of a set of vertices. In addition to cohesiveness, authors also considered the influence of the community. Following the popular linear threshold (LT) model [120], the aggregate influence probability of a community C w.r.t a vertex v, denoted by $Pr(v|C)$, is defined as follows:

$$Pr(v|C) = 1 - \prod_{u \in C}(1 - P_{u \to v})$$

where $P_{u \to v}$ is the probability that v is influenced by u. Note that there is a influence probability P_{uv} for each edge (u, v) in G, and $P_{u \to v}$ is computed by multiplying the influence of the edges along the maximum influence path [120] from u to v. Given a probabilistic threshold Δ, the influence score of the community C is the number of vertices in $G \setminus C$ with aggregate influence not less than Δ, denoted by $score(C)$. Below is the problem definition.

Problem 23 Given a simple graph G where each edge has an influence probability, the problem of the most influential community search is to find a maximal kr-clique community with the highest influence score.

It is shown in [125] that the problem is NP-hard because of the clique computation. A baseline solution is to access the vertices by their individual influence and compute the maximal kr-clique for each vertex. To improve efficiency, a tree structure named C-Tree is proposed such that any kr-clique community can be generated efficiently. Four efficient search algorithms are developed to significantly prune the search space based on the kr-clique constraints and the influence scores.

5.5 Discussions

In this section, we survey the CS solutions [45,125,187,195,205] using k-clique model. We can divide them into two groups, where the first group [45,187,195,205] focuses on simple graphs, while the second group [125] is developed for attributed graphs. In the first group, the first one [45] uses quasi-clique model, the second one [205] adopts k-clique model, and the last two [187,195] are based on k-plex model. However, to our best knowledge, there is no systematic study to compare the goodness of different k-clique-based models in real-life applications, which is crucial for researchers and practitioners to choose desirable models in practice. Moreover, there is no investigation on the trade-off between the computing time complexity and the flexibility of these models. It will be interesting to fill these two gaps in the future study.

6 *K*-ECC-based community search

In this section, we review CS studies [25,95] that use the k-ECC model as the community structure cohesiveness. Given a graph G and a set Q of vertices, their general goals are to find a subgraph H of G, which contains Q and has the maximum edge connectivity, also called the Steiner Maximum-Connected Subgraph (SMCS). Their difference is that one maximizes the size of H [25], while the other one tries to minimize the size of H [95].

6.1 Maximum SMCS

In [25], Chang et al. computed the maximum SMCS for a set of query vertices Q, which is defined as follows:

Problem 24 Given an undirected simple graph $G(V, E)$, and a set of query vertices $Q \subseteq V$, return a subgraph $H(V_H, E_H)$ of G, such that

1. V_H contains Q;
2. $\lambda(H)$ is maximized;
3. There exists no other subgraph H' satisfying the above properties, such that $H \subset H'$.

For example, consider the graph in Fig. 22a. Let $Q = \{v_1, v_4\}$. Then, for this query we will return the subgraph g_1, and its connectivity is $\lambda(g_1) = 4$.

A basic solution of Problem 24 is to sequentially enumerate all the maximal k-ECCs by varying k from $|V|$ to 1, and stops when the first k-ECC which contains Q is found. Then, the first k-ECC is returned as the community. In the literature, there are two efficient k-ECC enumeration algorithms. One is based on graph decomposition [26], while the other one is based on the random contraction [7]. As shown in [25], the basic solution takes $O(|V| \cdot h \cdot l \cdot |E|)$ time if the first k-ECC enumeration algorithm is adopted, or $O(|V| \cdot t \cdot |E|)$ time if the second one is used, where h and l are bounded by small constants for real graphs, and $t = O(\log^2 \cdot |V|)$. Obviously, both of them are inefficient for large graphs.

To improve the query efficiency, Chang et al. proposed a novel compact index structure, which allows the query can be answered in optimal time cost, i.e., the time cost is linear to the size of H. The index is built based on a key observation that for any pair of vertices u and v in H, their connectivity $\lambda(u, v)$ is at least $\lambda(H)$. This implies that if the connectivity of each pair of vertices in G is preserved, then the query can be answered in linear time cost, because we can first get $\lambda(H)$

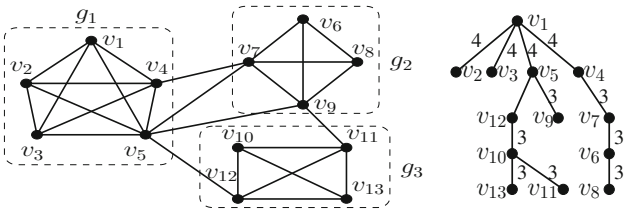


Fig. 22 An example for illustrating maximum SMCS [25]

by checking the connectivity of vertex pairs in Q, and then find H by traversing the connected edges whose connectivity are at least $\lambda(H)$.

To preserve all the connectivity information of G, Chang et al. developed the concept of *connectivity graph* G_c for the graph G, which has the same sets of vertices and edges with G, and for each edge $(u, v) \in G_c$, it is associated with a connectivity value denoting the edge connectivity between vertices u and v in G. Then, the maximum spanning tree (MST) of G_c is the index structure built for G. For example, Fig. 22b presents the index structure for the graph in Fig. 22a. The index can be built by first constructing the connectivity graph G_c and then computing the MST from G_c. Clearly, the space cost of the MST is $O(|V|)$ since it has $|V|$ vertices and at most $|V| - 1$ edges.

Based on the index MST, Chang et al. proposed an efficient query algorithm to solve Problem 24. Specifically, it first computes $\lambda(H)$ by using the MST and then finds the maximum SMCS by collecting the subtree of MST, whose edges have connectivity values being at least $\lambda(H)$. By using the technique of lowest common ancestor (LCA), the query can achieve a time cost of $O(|H_V|)$, which is optimal since outputting the vertex set of H takes $O(V_H)$ time.

In addition, the authors studied a variant of Problem 24 by imposing an additional constraint, which requires the number of vertices in H is at least L, where L is a parameter specified by the user. It can also be solved in optimal time cost with the index MST.

6.2 Minimum and minimal SMCS's

In [96], Hu et al. found that although the maximum SMCS has a high cohesiveness (i.e., high *connectivity*), the size of maximum SMCS's is often extremely large and complex. For example, on the DBLP bibliographical network that contains $803K$ vertices and $3.2M$ edges, the average number of vertices in a maximum SMCS is over $400K$. This not only hinders the analysis of the SMCS structure, but also makes it difficult to be used in real situations. To remedy this issue, Hu et al. examined the discovery of an SMCS that has a small number of vertices. Particularly, they studied the minimum SMCS and minimal SMCS problems:

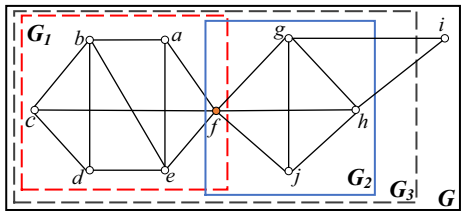


Fig. 23 The maximum SMCS (G_3), minimum SMCS (G_2), and minimal SMCS's (G_1 and G_2) for query $Q = \{f\}$ [96]

Problem 25 (*Minimum SMCS*) Given an undirected simple graph $G(V, E)$, and a set of query vertices $Q \subseteq V$, return a subgraph $H(V_H, E_H)$ of G, such that

1. V_H contains Q;
2. $\lambda(H)$ is maximized;
3. $|H_V|$ is minimized.

Problem 26 (*Minimal SMCS*) Given an undirected simple graph $G(V, E)$, and a set of query vertices $Q \subseteq V$, return a subgraph $H(V_H, E_H)$ of G, such that

1. V_H contains Q;
2. $\lambda(H)$ is maximized;
3. There exists no other subgraph $H' \subset H$ satisfying the above properties.

Obviously, a minimum SMCS is also a minimal SMCS, and both of them are much smaller than the maximum SMCS. For example, on the DBLP network, their average sizes are less than $0.23K$, while the average size of maximum SMCS is over $400K$. We illustrate these three kinds of SMCS in Fig. 23.

In [96], Hu et al. showed that the minimum SMCS problem is APX-hard, since it is a generalization of the STEINER TREE problem (see Sect. 3.1.2). Furthermore, unless $P = NP$, there does not exist any polynomial-time algorithm that approximates the minimum SMCS problem within any constant ratio. Therefore, it is not only intractable to obtain a minimum SMCS, but also hard to get its approximate version in an accurate manner. To trade-off the efficiency and result quality, Hu et al. [96] focused on the minimal SMCS problem.

A naive solution for Problem 26 is to first adopt the solution in [25] to compute the maximum SMCS G' and then iteratively refine G' to ensure its minimality. While this solution is simple, it has a high time complexity, since the cost of testing the minimality of an SMCS is high. To achieve higher efficiency, Hu et al. proposed an *Expand-Refine* framework to find a minimal SMCS, which consists of three steps. First, the Steiner connectivity of the query vertex set Q (i.e., the maximum $\lambda(H)$) is computed. Then, in

the *Expand* step, through local expansion of vertices starting from vertices in Q, a subgraph H' of G with connectivity being $\lambda(H)$ is obtained. In the *Refine* step, an algorithm is proposed to remove vertices based on the dependence of vertices on their minimal SMCS's. As a result, the minimal SMCS problem can be solved in a polynomial time cost, i.e., $O(t \cdot h \cdot l \cdot |E|)$, where $t < |H_V|$, and h and l are usually bounded by small constants. Besides, to further improve the efficiency, the authors relaxed the constraints from two perspectives, namely connectivity and minimality, and computed the approximate SMCS with theoretical guarantee.

In addition, for an important special case with only one query vertex (i.e., $|Q| = 1$), Hu et al. developed a customized algorithm for it. The main idea is to keep the processing information related to the current query in a small cache structure and use these information to answer the subsequent queries. As a result, it performs faster than the solution above.

6.3 Discussions

In this section, we review two CS studies that adopt the k-ECC model as the community cohesiveness metric. The first one [25] aims to find the maximum SMCS, while the second one [95,96] tries to find the minimum SMCS. In terms of efficiency, the maximum SMCS can be computed more efficiently. For example, by using the MST index [25], it can be computed in the optimal time cost. Nevertheless, the maximum SMCS may have size much larger than that of the minimum or minimal SMCS's. This also implies that for practitioners, they have to choose the specific algorithm, based on their specific requirements on community sizes and efficiency.

We remark that these two CS studies mainly focus on simple graphs. It is not clear how to adapt for them for other kinds of graphs, such as directed graphs and attributed graphs. Thus, an interesting future topic is to investigate how to perform CS on other kinds of graphs by adopting the k-ECC model.

7 Other metrics-based community search

In this section, we review a particular kind of community search, namely local community detection, which takes an input vertex as a seed and expands the community from the seed according to a specific goodness function. The representative goodness functions are local modularity [40,136], query biased density [190], personalized PageRank [114], and neighbor expansion [142].

7.1 Local modularity-based community search

Generally, studies of local modularity-based CS follow Problem 1 with a local modularity-based goodness function f. Two typical such functions are as follows:

• **Boundary-based local modularity** [40] Assume we have a simple undirected graph G and three sets of vertices, i.e., $\mathcal{C}, \mathcal{U}, \mathcal{B} \in G$. The known set $\mathcal{C}$ contains vertices in the known proportion of the community; the unknown set $\mathcal{U}$ is a set of vertices that are adjacent to vertices in $\mathcal{C}$; and the boundary set $\mathcal{B}$ is a subset of $\mathcal{C}$, which contains vertices having neighbors in $\mathcal{U}$.

By considering all the edges linked to sets $\mathcal{B}$ and $\mathcal{C}$, Clauset et al. [40] defined the local modularity of C as $f(\mathcal{C}) = I/T$, where I is the number of edges with no end vertex in $\mathcal{U}$, and T is the number of edges with at least one end vertex in $\mathcal{B}$. Intuitively, a good community has a sharp boundary, which means that there are few connections from its boundary set $\mathcal{B}$ to the unknown set $\mathcal{U}$, resulting in a higher value of $f(\mathcal{C})$.

To uncover a community, Clauset et al. developed an algorithm that works in vertex-at-a-time manner. Let q be a source (seed) vertex. Initially, it lets $\mathcal{C} = \{q\}$ and puts q's neighbors into set $\mathcal{U}$. At each step, it adds to $\mathcal{C}$ the neighboring vertex that results in the largest increase of the local modularity. This process continues until it has agglomerated either a given number of vertices k, or it has discovered the entire enclosing component, whichever happens first. As a result, its time complexity is $O(k^2 d)$, where d is the mean degree and k is the number of vertices to be explored.

• **Subgraph degree-based local modularity** [136] Given a subgraph $\mathcal{C}$ of a graph G, Luo et al [136] defined its in-degree, $ind()$, as the number of edges within $\mathcal{C}$, and its out-degree, $outd(\mathcal{C})$, as the number of edges that connect $\mathcal{C}$ to the remaining part of G. Then, they defined the subgraph modularity of S as $f(\mathcal{C}) = ind()/outd()$. Clearly, its value will increase if $\mathcal{C}$ has more internal edges and fewer external edges.

To find a community, Luo et al. proposed an algorithm consisting of an addition step and a deletion step. Initially, $\mathcal{C}$ contains a seed vertex q and its neighbors are in a set $\mathcal{N}$. In the addition step, it iteratively adds vertices from $\mathcal{N}$ to $\mathcal{C}$ that results in the greatest increase of $f(\mathcal{C})$, until a certain number of neighbors have been in the subgraph. In the deletion step, it iteratively removes vertices in $\mathcal{C}$ that result in the increase of $f(\mathcal{C})$ but not separating $\mathcal{C}$. The addition and deletion steps will be repeated until no vertex is added to $\mathcal{C}$. Note that there is no guarantee whether q will be in the returned community as it may be removed during the deletion step. It has the same time complexity as the algorithm for the boundary-based local modularity.

7.2 Query biased density-based community search

In [190], Wu et al. proposed the query biased density as the goodness function for CS. Before introducing the query biased density, the authors presented a vertex weighting scheme, which ensures that vertices far away from the query vertices will have large weights, resulting in high penalties to be included in the community. To assign each vertex u a weight $r(u)$ w.r.t a set Q of query vertices, they adopted the penalized hitting probability, which can be computed by random walk. Then, the query biased vertex weight of vertex u, $\pi(u)$, can be defined as the reciprocal of $r(u)$, i.e., $\pi(u) = 1/r(u)$.

Based on the weights, the authors defined the *query biased density* of a graph S as $\rho(S) = \frac{e(S)}{\pi(S)}$, where $e(S)$ is the sum of edges weights and $\pi(S)$ is the sum of query biased weights for vertices in S. After that, the authors proposed and studied the problem of finding the query biased densest subgraph S from a graph G (or QDS problem), which theoretically guarantees that QDS is a connected subgraph and contains Q.

Clearly, if $\pi(u) = 1$, the query biased density degenerates to the classical edge density (i.e., $\frac{e(S)}{|S|}$), and accordingly the QDS problem is reduced to the problem of densest subgraph discovery [78]. This also implies that after weighting $\pi(u)$, it forces the global densest subgraph shift to the neighborhood of the query vertices.

Unfortunately, the QDS problem is computationally intractable. To improve efficiency, the authors introduced two variants of the QDS problem by removing constraints that S is connected and Q is included in S, respectively. They showed that these variants can be solved in polynomial time and the results can be used to find an optimized solution for the QDS problem.

7.3 Personalized PageRank-based community search

In [114], Kloumann et al. studied the use of personalized PageRank (PPR) model for identifying the community of a set of seed vertices Q. We first introduce the PageRank model: suppose there are an infinite number of surfers walking on a graph. If at a certain timestamp a surfer is staying at vertex i, at the next timestamp she goes to a random neighbor vertex j. As time goes on, the expected percentage of surfers at each vertex i converges (under certain conditions) to a limit $r(i)$, called PageRank score of vertex i. Since $r(i)$ is independent of the distribution of starting vertices, it reflects the global importance of the vertex i.

Notice that $r(i)$ is computed with no preference for any particular vertices. However, in reality, for a particular user, some vertices, denoted by a set Q, may be more interesting than others, and they could be considered as the *preferred vertices*. To incorporate preferences of Q into the model above, we can make a modification: At each step, a surfer jumps back to a vertex in Q with probability c, and with probability $(1 - c)$ continues forth along a neighbor. The limit distribution of surfers in this model would favor vertices in Q and vertices which are close to Q. The modified model is also called PPR model. Clearly, if we let Q be a set of query vertices, the vertices whose limit probabilities are highest can be considered as Q's community members.

Now we formally introduce the PPR model. Consider a graph G and let $\deg_G(i)$ denote the degree of vertex i and $\mathbf{A}$ be the adjacent matrix of G, i.e., $A_{i,j} = \frac{1}{deg_G(i)}$ if vertex i is linked to vertex j, where $\deg_G(i)$ is the degree of vertex i. The *preference vector* $\mathbf{u}$ is defined over the seed vertices such that $|\mathbf{u}| = 1$ and $u(i) = \frac{1}{|Q|}$ if the i-th vertex is in Q. Then, the PPR equation is $\mathbf{v} = (1 - c)\mathbf{A}\mathbf{v} + c\mathbf{u}$, where $c \in (0, 1]$ is the decay factor and a typical value of c is 0.10 [114]. The solution $\mathbf{v}$, called PPR vector, is a steady-state distribution of surfers.

Problem 27 Given a graph $G(V, E)$, a set of query vertices $Q \subseteq V$, and an integer k, return a set C of vertices, such that

1. $Q \subseteq C$;
2. C contains k vertices, whose corresponding values in the PPR vector w.r.t Q are the highest.

In the literature [9,114], many efficient PPR algorithms have been developed, and thus can be applied to CS. We skip the details due to space limitation.

7.3.1 Neighbors expansion-based community search

In [142], Mehler et al. presented a neighbor expansion method to discover the community from representative seeds. Specifically, given a graph $G(V, E)$ and a set S of seed vertices, it repeatedly identifies the optimal "next" vertex v, which is not in the community C (initially $C = S$) but linked with vertices of C, based in some manner on the number or strength of v's neighbors who had previously been identified as community members. Details of vertex selection criteria and stopping rules of the expansion process are introduced as follows:

• **Selection criteria** Mehler et al. proposed to assign a score to each vertex in the graph and select the highest-scoring outside vertex to join the community. The score assignment criteria are as follows:

- *neighbor count* the number of v's neighbors in C;
- *juxtaposition count* consider the weights of edges when counting the number of v's neighbors in C;
- *neighbor ratio* normalize vertices' degrees and count the degree-normalized neighbors in C;

– *juxtaposition ratio* consider the weights of edges when computing the neighbor ratio;
– *binomial probability* compute the binomial probability that v is in C, given its neighbor count.

• **Stopping rules** The authors proposed to reserve some fraction of seed vertices as validation members and then monitor the frequency with which these validation members are incorporated into the community, during the expansion process. In the first phase, when community members are identified with high precision, we expect to add a new validation member with frequency equal to the fraction of community comprised by the validation set. After leaving the natural boundaries of the neighborhood, we expect to rediscover validation members according to their frequency in the entire graph. As a result, we can find the stopping vertex as the one that best splits the validation interval (i.e., the difference between the discovery times of the ith and $(i-1)$-st validation members) into two groups.

7.4 Discussions

In this section, we review CS studies that do not rely on metrics introduced in Sect. 2, which are often referred as local community detection. These studies mainly focus on simple undirected graphs and uncover the communities by seed expansion using link-based metrics, such as modularity, density, PageRank, etc. Unlike CS studies introduced before, these works often rely on good seed selection algorithms [146] and assume that there are some ground-truth communities. In other words, they might not aim to search communities in an online manner over big graphs, based on a query request. As a result, some of them may cost high running time for searching communities. Consequently, an interesting research direction is to develop index-based solutions for supporting efficient online CS queries using these metrics. Moreover, it would be interesting to study how to apply them for CS on attributed graphs.

8 Community search systems

Recently, many graph processing systems have been developed [18]. Generally, they can be classified into two groups. The first group (e.g., GraphX [80] and Pregel [138]) aims to provide a platform for supporting general graph tasks (e.g., computing PageRank scores). The second group is customized for specific graph tasks. For example, in [69], Fan et al. developed a graph system, called Expfinder, for finding experts in social networks; in [105], a system called VIIQ is developed for interactive graph query formulation; in [203], AutoG shows an interactive system to facilitate graph query formulation. However, none of them can be readily used for CS. To address this issue, recently some systems have been developed for searching, visualizing, and analyzing communities in large graphs. Below, we introduce two systems, namely C-Explorer [62] and VizCS [106].

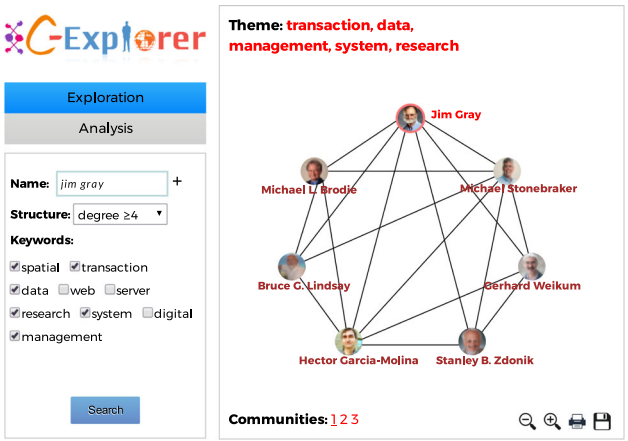


Fig. 24 Interface of C-Explorer [62]

8.1 C-Explorer

C-Explorer is a web-based system that enables community retrieval in a simple, online, and interactive manner. The key features of C-Explorers are as follows:

First, it implements several typical CS algorithms on simple undirected graphs and keyword-based attributed graphs, including `Global` and `Local` (see Sect. 3.1), ACQ algorithm (see Sect. 3.3). In addition, a CD algorithm called `CODICIL` [164] is included.

Second, it offers a user-friendly facility that enables online visualization of communities. Figure 24 shows the user interface of C-Explorer configured to run on the DBLP bibliographical network. On the left panel, a user inputs the name of an author (e.g., "jim gray") and the minimum degree of each vertex in the community she wants to have. The user can also indicate the labels or keywords related to her community. Once she clicks the "Search" button, the right panel will display a community of Jim Gray. The user can further click on one of the vertices (e.g., Michael Stonebraker) and continue to examine its community.

Third, it allows users to compare the communities retrieved by various CS and CD algorithms, in terms of community quality and statistics.

Finally, it provides a list of API functions so that other CS and CD algorithms can be plugged in. For public users, they can easily plug their own algorithms into C-Explorer using these API functions.

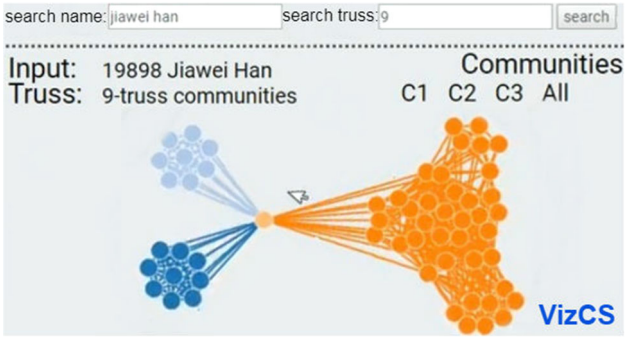


Fig. 25 Interface of VizCS [106]

8.2 VizCS

VizCS is an online query processing system for searching and visualizing communities in graphs [106]. VizCS exhibits four key innovative features as follows:

First, VizCS adopts a triangle-connected truss community model for dynamic graphs where vertices/edges undergo frequently insertions/deletions [98]. It provides the feature of CS over dynamic graphs, which can be uploaded with one file of graph updates by users.

Second, VizCS offers a user-friendly visual interface to formulate queries and a real-time response query processing engine. Figure 25 shows an example query of author vertex $q =$ "Jim Gray" and parameter $k = 8$. Thanks to efficient k-truss CS algorithms, the query results can be quickly obtained in real time.

Third, VizCS generates a community exploration wall by offering interactive community visualization, which facilitates users to in-depth understanding of the data. The community exploration wall uses graph visualization techniques to depict the community results and also presents informative features to users through various exploration channels, such as the profile search of community members by Google, structural statistic report, collaborator recommendation, and tag cloud. Figure 25 shows the community exploration wall.

Last but not least, VizCS is a CS platform that can visualize and compare different community results by various state-of-the-art algorithms and user-uploaded approaches. It benefits users to understand different models vividly and directly.

9 Comparison analysis

Recall that in the last subsections of Sects. 3, 4, 5, and 6, we have compared and analyzed the CS solutions using k-core, k-truss, k-clique, and k-ECC, respectively. In this section, we would like to further compare these CS solutions across different metrics. Due to the space limitation, we are unable to compare all the surveyed 27 CS problems as well as their solutions. In the following, we mainly compare the representative CS problems and solutions on simple graphs and attributed graphs, respectively, while other solutions can be considered as either their variants or less representative studies.

9.1 Simple graphs

In this section, we compare representative CS problems for cohesiveness metrics studied on simple graphs, which are Problem 1 for k-core, Problem 15 for k-truss, Problem 20 for k-clique, and Problem 24 for k-ECC. In the following, we first compare these solutions in terms of the complexities and scalability of the state-of-the-art online algorithms, index construction complexities, index-based query algorithms, community cohesiveness, and support for overlapped CS as well as dynamic graphs. After that, we perform an experiment on real large graphs by using these CS algorithms and compare their empirical performance.

To make a fair comparison, we consider a simple undirected graph $G(V, E)$, where $n = |V|$, $m = |E|$, and its arboricity is denoted by $\alpha(G)$ ($\alpha(G)$ is often much smaller than $\sqrt{m}$). We use h and l to denote small values that can be bounded by small constants [25]. In Table 4, we compare these representative CS solutions on G. Note that to measure the strength of algorithm scalability and community cohesiveness, we use notation ★; that is, an algorithm with more ★ means that it has better scalability or cohesiveness. Meanwhile, if a CS solution returns only one community C, we denote its community edge number by $|E(C)|$. If multiple communities are returned, we use C_i to denote the i-th ($1 \leq i \leq r$) community, where r is the total number of returned communities. We use "O" and "D" to denote whether the solutions support overlapped CS and dynamic graphs, respectively.

In addition, for the complexities of the k-clique-based algorithm, we adopt the notations in [205], where s is the average size of maximal cliques, T is the time to enumerate all maximal cliques, L is the number of maximal cliques, p is the average number of maximal cliques a vertex is contained in, Q is the number of maximal cliques containing at least one query vertex, and g is the height of the index tree.

From Table 4, we can make the observations:

– For online query algorithms, in terms of query time complexity, we can rank them as: k-core $\preceq$ k-ECC $\preceq$ k-truss $\preceq$ k-clique, which is consistent with the efficiency ranking relationship of these metrics in Sect. 2.2. As a result, the k-core-based algorithm achieves the highest scalability while the k-clique-based algorithm has the lowest scalability.

Table 4 Comparison analysis for representative CS solutions on simple graphs

Metric	Online algorithm		Index-based algorithm				Cohesiveness	O	D
	Query	Scalab.	Time	Space	Scalab.	Query			
k-core	$O(n)$ [175]	★★★★	$O(m)$ [17]	$O(n)$ [17]	★★★★	$O(\|E(C)\|)$	★	×	✓
k-truss	$O(m^{1.5})$ [98]	★★	$O(m^{1.5})$ [6]	$O(m)$ [6]	★★★	$O(\sum_{i=1}^{r} \|E(C_i)\|)$	★★★	✓	✓
k-clique	$O(\log(n)sT)$ [205]	★	$O(sLp)$ [205]	$O(sL)$ [205]	★	$O(g \log(g)Q)$ [205]	★★★★	✓	✓
k-ECC	$O(hlm)$ [25]	★★★	$O(\alpha(G)hlm)$ [25]	$O(m)$ [25]	★★	$O(\|E(C)\|)$	★★	×	✓

– For index construction algorithms, the ranking relationship above still holds. For index-based query algorithms, most of them except k-clique have the optimal time complexity, which is linear to the community edge number (i.e., $|E(C)|$).
– The community structure cohesiveness is in line with the cohesiveness of these four metrics.
– The k-core and k-ECC-based solutions can only return one community for each query, while the other two solutions may return multiple overlapped communities containing the query vertex.
– All algorithms support dynamic graphs where vertices and edges are inserted or deleted dynamically.

Next, we empirically evaluate the performance of algorithms in Table 4. The input of these algorithms except the k-truss-based one is a query vertex, and they aim to find communities containing the query vertex which will maximize the value of k. For the k-truss-based one (Problem 15), its input is a set of query vertices and an integer k. To make a fair comparison, we adapt its algorithm such that its input is a query vertex and the algorithm will maximize the value of k. To measure the quality of returned communities (subgraphs), we introduce four metrics, i.e., diameter, degree, density (i.e., the number of edges over the maximum number of possible edges in a graph), and clustering coefficient (CC). Generally, a lower value of diameter and higher values of degree, density, and CC mean the higher quality of the community.

To conduct the experiments, we use a real-world graph Google [4], which contains 875,713 vertices and 5,105,039 edges. We randomly select 100 vertices from the graph as query vertices, perform CS queries using these vertices, compute the average running time and community quality, and report experimental results in Table 5. Generally, the efficiency results in Table 5 are consistent with the complexity analysis in Table 4. More specifically, we have:

– For online query algorithms, the k-core-based algorithm is the fastest. The k-truss-based and k-ECC-based algorithms have similar time cost. The k-clique-based algorithm takes the highest time cost.
– To build indexes, the k-core-based algorithm is the fastest and the k-truss-based algorithm is slower than others.
– For index space cost, the k-core-based index takes the least space, while the space cost of others is around or over an order of magnitude larger than that of k-core-based algorithm.
– For index-based query algorithms, the k-core-based algorithm is slower than the k-truss-based algorithm (which also takes optimal query time cost), because its returned communities are larger than those of other algorithms.

[4] Available at http://snap.stanford.edu/data/index.html.

Table 5 Empirical comparison for representative CS solutions on a real large graph

Metric	Online algorithm	Index-based algorithm			Community quality				Community number
	Query (s)	Time (s)	Space (MB)	Query (s)	Diameter	Degree	Density	CC	
k-core	7.2	8.1	7.9	2.7	14.0	19.2	0.044	0.763	1
k-truss	55.1	103.1	179	0.2	4.1	13.9	0.476	0.868	1.31
k-clique	1872	61.6	108	4.3	10.6	9.2	0.424	0.709	1.05
k-ECC	39.9	38.3	68	0.15	10.5	18.4	0.152	0.774	1

The k-clique-based algorithm is the slowest, as its complexity is higher than others.

- In terms of community quality, the k-truss-based solution achieves the smallest diameter, highest density, and highest clustering coefficient, due to small and tight triangle-based community structure. The k-core-based algorithm achieves the highest degree, against other methods. The k-clique-based method achieves the smallest degree.
- In line with Table 4, the k-core-based and k-ECC-based solutions return one community, while k-truss-based and k-clique-based solutions, respectively, return 1.31 and 1.05 communities.

9.2 Attributed graphs

As shown in Table 1, for attributed graphs, five kinds of attributes have been considered for CS, which are keywords, locations, temporal information, profile, and influence values. However, the semantics of these attributed communities are different. Moreover, the problem definitions are also different. Therefore, it may not make sense to compare them under the same metrics.

For location, temporal information, and profile-based attributed graphs, only the k-core model has been studied on these graphs, which have been discussed and compared extensively in Sect. 3.8. For influence value-based graphs, the meanings of influences are very different. In k-core-based CS solutions [21,30,50,126–128], the influence values are associated with graph vertices, denoting their influence or importance. In k-truss-based CS solutions [216], the influence values are associated with graph edges, representing the influence or importance of edges. In k-clique-based CS solutions [125], the influence values are also associated with graph edges, but they are probability values, meaning how likely a vertex is influenced by another vertex. Meanwhile, none of these influence value-based graphs has been investigated with at least two different cohesiveness metrics, so we do not compare solutions for influence value-based graphs in this paper. In the following, we mainly focus on comparing and analyzing CS solutions on keyword-based attributed graphs.

For keyword-based attributed graphs, there are two representative studies, namely ACQ [58,61] and ATC [102]. Generally, both of them seek to find a densely connected community containing query vertex(es) with similar query keywords, but ACQ adopts the k-core model, while ATC uses the k-truss model. From the discussions in Sect. 2.2, we infer that the community of ATC is more structurally cohesive, but may take higher computational cost. Besides, in terms of keyword cohesiveness, ACQ model in Sect. 3.3 imposes a strict homogeneity constraint, requiring that each vertex shares same query attributes in the community; ATC model in Sect. 4.3 uses an attribute score function to quantify the query keyword coverage and allows missing some query keywords in the community.

In [102], Huang et al. empirically compared the community quality and efficiency of ACQ and ATC. They used 13 real graphs with ground-truth communities. For each graph, they ran 200 CS queries. Specifically, for each query, they randomly selected a ground-truth community and then randomly selected a vertex from the community as the query vertex. After that, they ran ACQ and ATC with the same parameters, i.e., $k = 4$, and two query keywords which are selected from the community. The results are consistent with the discussions above. Specifically, ATC achieves higher average F_1 score values than ACQ on all the datasets, which means that it is more accurate to search communities. On the other hand, in terms of efficiency, ACQ consistently outperforms ATC on all the datasets and is up to two orders of magnitude faster than ATC.

10 Related work

In this section, we review related studies, including community detection, cohesive subgraph discovery, graph keyword search, and graph pattern matching.

10.1 Community detection

Below, we review representative CD studies on undirected graphs, directed graphs, and attributed graphs.

10.1.1 Undirected graphs

A large number of studies aim to detect communities from simple graphs, and we can classify these studies based on the techniques they use. Some representative classes are as follows, to name a few:

1. Community quality optimization-based methods (e.g., modularity [148]);
2. Clustering methods (e.g., k-means [178], spectral clustering [182]);
3. Graph partitioning methods (e.g., Metis [109]);
4. Embedding-based methods (e.g., DeepWalk [132,155]);
5. Random walk-based methods (e.g., [157]);
6. Label propagation-based methods (e.g., [81]);
7. Information diffusion-based methods (e.g., [87]);
8. Statistic inference-based models (e.g., [89]);
9. Deep learning-based methods (e.g., [200]);
10. Centrality-based methods (e.g., [149]);
11. Locality sensitive hashing-based methods (e.g., [137]);
12. Physics-based methods (e.g., Potts low [189]);
13. Local metric-based methods (e.g., k-plex [42]);
14. Multi-commodity flow-based methods (e.g., [122]);
15. Hybrid-based methods (e.g., [91]).

For a detailed survey of CD, please refer to the following survey and empirical evaluation papers: [8,44,48,71,83,88,110,112,123,152,154,156,158,163,194,197]. Although these CD solutions are able to discover communities from networks, they may not well satisfy the desirable factors of CS on big graphs as we discuss in Sect. 1, because most of them often use a global predefined criterion for generating communities and cannot find communities in an online manner.

10.1.2 Directed graphs

In recent years, a number of studies have investigated CD on directed graphs. Here are some representative studies, to name a few. In [121], Leicht et al. extended the concept of modularity maximization [148], which was originally designed for undirected graphs, for detecting community structure in directed networks that makes explicit use of information contained in edge directions. In [70], Flake et al. identified communities from websites network, which can be considered as directed graphs. In [119], Lancichinetti et al. introduced new benchmark graphs to test CD methods on directed networks. In [113], Kim et al. also proposed a new modularity metric for CD on directed networks. In [201], Yang et al. developed a new stochastic block model for CD on directed networks. In [199], Yang et al. presented algorithms for detecting communities from both directed and undirected networks. Ning et al. [150] studied local community extraction in directed networks. A recent survey can be found in [139].

10.1.3 Keyword-based attributed graphs

To identify communities from keyword-based attributed graphs, recent works [33,99,159,164,176,220] often use clustering techniques. Zhou et al. [220] computed vertices' pairwise similarities using both links and keywords and then clustered the graph. Subbian et al. [176] explored noisy labeled information of graph vertices for finding communities. Qi et al. [159] dynamically maintained communities of moving objects using their trajectories. Ruan et al. [164] developed a method `CODICIL`, which augments the original graph by creating new edges based on content similarity and then performs clustering on the new graph.

Another common approach is based on topic models. In [135,147], the `Link-PLSA-LDA` and `Topic-Link LDA` models jointly model vertices' content and links based on the `LDA` model. In [192], the attributed graph is clustered based on probabilistic inference. In [165], the topics, interaction types, and the social connections are considered for discovering communities. `CESNA` [198] detects overlapping communities by assuming communities "generate" both the link and content. A discriminative approach [202] has also been considered for community detection. However, computing pairwise similarity among vertices is very costly, and thus, they are questionable for performing online CS queries.

10.1.4 Location-based attributed graphs

The problem of CD on location-based attributed graphs (or geo-social networks) [16] has been extensively studied [32,54,77,84,172]. In [77], Girvan et al. introduced the geo-community, which is a graph of intensely connected vertices being loosely connected with others, but it is more compact in space. Guo et al. [84] proposed the average linkage (ALK) measure for clustering objects in spatially constrained graphs. In [54], Expert et al. uncovered communities from spatial graphs based on modularity maximization. In [172], Shakarian et al. used a variant of Newman–Girvan modularity to mine the geographically dispersed communities. In [32], Chen et al. proposed a method using modularity maximization for detecting communities from geo-social networks.

10.1.5 Temporal graphs

Many recent studies aim to detect communities from temporal graphs. In [217], Zhou et al. studied CD over a temporal heterogeneous social network consisting of authors, document content, and the venues. In [134], Liu et al. studied persistent community detection for identifying communities

that exhibit persistent behavior over time. In [10], Angadi et al. detected communities from dynamic networks where data arrive as a stream to find the overlapping vertices in communities. In [19], Bazzi et al. investigated the detection of communities in temporal multilayer networks. In [51], DiTursi et al. proposed a filter-and-verify framework for community detection in dynamic networks. In [116], Kuncheva et al. presented a method by using spectral graph wavelets to detect communities in temporal graphs. For more related studies, please refer to survey papers [163,177].

10.2 Cohesive subgraph discovery

In this section, we review studies on cohesive subgraph discovery. Notice that CD is one kind of cohesive subgraph discovery, but the latter one is more general.

10.2.1 Simple graphs

For simple graphs, typical cohesive subgraph models are k-core [17,170], k-truss [41,166,212], k-clique [2,151], and k-ECC [76,95], as discussed in Sect. 2. To compute these subgraphs, there are many efficient in-memory algorithms (e.g., k-core [17], k-truss [184], k-clique [47], and k-ECC [7,26,218]). For graphs that are too large to be kept in memory, there are also some disk-based and parallel algorithms. For example, in [34,111,184,188], and [36], disk-based algorithms for computing k-core, k-truss, and k-clique are developed, respectively; in [145] and [29], parallel algorithms for computing k-core and k-truss are proposed, respectively. In addition, to maintain k-core and k-truss for dynamic graphs, some efficient algorithms are developed in [130,167,213] and [219], respectively.

Besides, there are many other cohesive subgraph models and the representatives are as follows: In [171], Seidman proposed the k-plex model (which is introduced in Sect. 5). In [141], Matsuda et al. introduced the concept of quasi-clique model. In [210], Zhang et al. proposed the (k, s)-core, which considers both user engagement and tie strength. In [168], the authors proposed the concept of nucleus, which is a generalization of k-core and k-truss. In [214], Zhao et al. introduced the mutual-friend subgraph. In [186], Wang et al. proposed the DN-Graphs by considering vertices' common neighbors. In [26], Chang et al. studied the problem of enumerating k-ECCs in a graph for a given k. In [222], Zhu et al. introduced the notion of coherent cores on multilayer graphs. In addition, Goldberg et al. [78] and Fang et al. [67] discovered the densest subgraph, Galbrun et al. [73] studied the top-k densest subgraphs, Tsourakais et al. [180] computed the quasi-clique-based dense subgraphs, and Qin et al. [161] studied the problem of finding top-k locally densest subgraphs.

10.2.2 Attributed graphs

For attributed graphs, in addition to CD methods, there are also many studies of finding cohesive subgraphs. In [196], Yang et al. studied the socio-spatial group query which finds a group of users that are cohesively linked and close to the rally point in a geo-social network. In [211], Zhang et al. studied the problem of finding (k, r)-cores on attributed graph and for a specific (k, r)-core, each vertex has at least k neighbors, and the attribute similarity of each pair of vertices is at least r. In [28], Chen et al. studied the problem of (k, d)-MCC (maximum colocated community) search on geo-social network, where a (k, d)-MCC is a connected k-truss and for any two vertices, their distance is at most d. In addition, Wu et al. [191] studied the problem of finding the densest connected subgraph from the dual network, which can be considered as an attributed graph.

10.3 Graph keyword search

Generally, graph keyword search [183,204,206,208] aims to find a tree or a subgraph, which contains a set of query keywords, from a large graph G. Earlier studies often output a tree structure. In [20], Bhalotia et al. developed a backward algorithm for finding Steiner trees. In [49], Ding et al. proposed a dynamic programming algorithm finding Steiner trees. In [79], Golenberg et al. presented a novel algorithm which produces Steiner trees with polynomial delay. In [107], Kacholia et al. proposed a bidirectional search algorithm, and He et al. [90] improved its efficiency by introducing a new index structure.

Recently, some solutions have output subgraphs. In [124], Li et al. proposed to find r-radius Steiner graphs that contain query keywords. Qin et al. [162] proposed to find multicentered subgraphs that contain query keywords within a given distance. Kargar et al. [108] studied the r-clique which is a set of vertices that cover query keywords and satisfy the distance constraint.

However, these works are substantially different from CS queries on keyword-based attributed graphs. First, they do not specify query vertices as required by CS queries. Second, the tree or subgraph produced do not guarantee structure cohesiveness. Third, their solutions do not ensure strong keyword cohesiveness.

10.4 Graph pattern matching (GPM)

For simple graphs, the problem of GPM is NP-complete [43] and it has been studied extensively under different settings: (1) in main memory [37,181]. For example, Ullmann [181] proposed a backtracking algorithms. (2) In external memory, Chu et al. [39] and Hu et al. [97] studied triangle counting; in [160], a novel GPM solution based on graph compres-

sion is presented. (3) In distributed platforms, both DFS-style approaches [5,153] and BFS-style approaches [117,118] are developed. The DFS-style approaches avoid intermediate results by using one-round computation, while BFS-style approaches shuffle a large number of intermediate results.

For attributed graphs, there are also many studies. Tong et al. [179] studied the use of lines, loops, and stars for finding the matched subgraphs; Zou et al. [223] developed a novel GPM solution based on distance join; Fan et al. [55] studied GPM by using bounded simulation; in [56], GPM has been studied for finding graph association rules; in [35], Cheng et al. studied the problem of top-k GPM. Recently, Fang et al. have studied a variant of the GPM problem on spatial databases [59,64], and it aims to find spatial objects that are matched with a given pattern. However, GPM is different from CS since (1) it often focuses on small patterns, so it cannot generate large communities; and (2) the subgraphs of GPM solutions often do not guarantee strong structure cohesiveness. Other related topics include subgraph search [207,209].

11 Future work

Recall that in Table 1, the cohesiveness metrics are orthogonal to graph types, so if a metric has not been studied for a particular type of graphs, then it is a future research direction to study CS by applying the metric on this type of graphs. Apart from this, we present a number of promising future directions as follows:

11.1 Optimization for query parameters

Most existing CS queries require users to input some parameters, in addition to the query vertex. A typical parameter is the integer k [15,46,175], which controls the structure cohesiveness of returned communities. For attributed graphs, existing works also require users to input some parameters related to attributes. For example, in ACQ [61] and ATC [102], a set of query keywords are required. Although these parameters provide strong flexibility and personalization for the query, it may not be easy for users to set proper values for these parameters. For example, if the integer k is too large, a false query may incur, i.e., the query returns empty result. On the other hand, if k is too small (e.g., $k = 1$ or 2), the returned community may contain too many vertices, which may make the community meaningless.

Unfortunately, most existing CS works assume that users can input proper values for these parameters. This assumption, however, is too strong, especially when users do not know much about the underlying network. To suggest query parameters, a possible research direction is to exploit historical query logs and suggest some values of parameters automatically [13,140]. Another direction is to study how to use crowdsourcing platforms (e.g., AMT [1]) to facilitate query suggestions.

11.2 More cohesiveness metrics

As aforementioned, in CS solutions, a community is required to satisfy certain cohesiveness metrics. Essentially, the cohesiveness metrics formally define the communities, so they play crucial roles in CS.

For structure cohesiveness, there are many other cohesiveness models (see Sect. 10.2) which have not been used for CS. Thus, it would be interesting to study CS using these models. For example, in [168,169], the authors have proposed the concept of nucleus, which is a generalization of k-core and k-truss.

For attribute-based cohesiveness, as discussed in Sect. 10.2, there are some studies finding cohesive subgraphs from attributed graphs. Thus, it is of interest to extend them for CS on attributed graphs. Besides, each existing CS solution only focuses on one particular type of attribute (e.g., keyword). This, however, may be problematic for many real applications because a real graph often involves multiple types of attributes. Thus, it is desirable to study how to perform CS by considering multiple types of attributes.

11.3 Other types of graphs

In recent years, many novel network models have been developed and the representative ones are as follows:

- *Public–private network* [11,38,100]. In a public–private network (e.g., Facebook), there is a public graph G, containing a set of vertices and a set of edges that are visible to all users of the network. In particular, each vertex u is associated with a private graph G_u, where vertices of G_u are vertices from the public graph G, and G_u is only known to u.
- *Uncertain graph* [94,104,131]. In many real applications (e.g., biology), the graph data are often noisy, inexact, and inaccurate, and they can be modeled as uncertain graphs, where each edge is associated with a value denoting its existence probability.
- *Signed graph* [193]. A signed graph is a graph whose edges carry signs. For example, in social networks, the relationship of two users is either positive (e.g., friendship) or negative (e.g., hostility). Thus, users' relationship can be modeled as a signed graph.
- *Multi-dimensional graphs* [68]. In many scenarios, a graph often contains various types of edges, which represent various types of relationships between entities. Such graphs are often called multi-dimensional graph, or multilayer graphs or multi-view graphs.

– *Heterogeneous information network (HIN)* [93,174]. HINs are networks with multiple typed objects and multiple typed links denoting different relations.

To our best knowledge, there is no prior research about CS on these graphs. Thus, it is still an open problem of how to perform CS on these graphs.

11.4 Real big graphs

Most existing CS studies assume that the graphs can be kept in the memory of a single machine. The graphs used for experimental evaluation are often million-scale, and only a few of them [66,127] are able to process billion-scale graphs. However, in many real applications (e.g., Facebook), the graphs may involve billions of vertices and edges [133]. As a result, existing CS solutions may fail to process such real big graphs within reasonable time cost. Hence, how to efficiently perform online CS on such big graphs is a challenging task.

For big graphs that cannot be kept by a single machine, some possible research directions are as follows: First, we can consider developing query algorithms based on distributed computation platforms (e.g., GraphX [80]), which are able to process big graphs in a cluster. Second, to save memory space, we may keep the graph data on disk and design I/O-efficient query algorithms.

11.5 An online repository for codes and datasets

For most of surveyed CS studies, their codes of algorithms and datasets are not publicly available. Thus, it is desirable to build an online repository to keep these codes and datasets. The major benefits of doing this are twofold: First, for researchers, the codes and datasets can serve as a benchmark for comparison studies. Second, practitioners can easily plug these CS solutions into their applications without re-implementation.

12 Conclusion

In this paper, we conduct an extensive survey on the topic of community search over large graphs. We systematically review over 30 research articles, which focus on the topic of community search, published between 2010 and 2019. We first analyze and compare different community cohesiveness metrics. Then, we classify studies about CS according to these metrics, and for each class of works, we review and discuss the representative studies on different types of graphs. Furthermore, two systems that are customized for the purpose of community search are discussed. Finally, we point out a list of future research topics as well as challenges. In summary, our survey provides an overview of the start-of-the-art research achievements on the topic of community search, and it will give researchers a thorough understanding of community search.

Acknowledgements We would like to thank Jiafeng Hu and Kai Wang for their helpful discussions, Dan Yin for the proof-reading, and Jinbin Huang for conducting experimental comparisons. Xin Huang is supported by the NSFC Project No. 61702435, and Hong Kong General Research Fund (GRF) Project No. HKBU 12200917. Lu Qin is supported by DP160101513. Ying Zhang is supported by FT170100128 and DP180103096. Wenjie Zhang is supported by DP180103096. Reynold Cheng is supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and HKU (Projects 102009508 and 104004129). Xuemin Lin is supported by 2019DH0ZX01, 2018YFB1003504, NSFC61232006, DP180103096, and DP170101628.

References

1. Amazon mechanical turk. https://www.mturk.com/
2. Clique (graph theory). https://en.wikipedia.org/wiki/Clique_(graph_theory)
3. Acquisti, A., Gross, R.: Imagined communities: awareness, information sharing, and privacy on the facebook. In: International Workshop on Privacy Enhancing Technologies, pp. 36–58 (2006)
4. Adamcsek, B., Palla, G., Farkas, I.J., Derényi, I., Vicsek, T.: Cfinder: locating cliques and overlapping modules in biological networks. Bioinformatics **22**(8), 1021–1023 (2006)
5. Afrati, F.N., Fotakis, D., Ullman, J.D.: Enumerating subgraph instances using map-reduce. In: ICDE, pp. 62–73. IEEE (2013)
6. Akbas, E., Zhao, P.: Truss-based community search: a truss-equivalence based indexing approach. PVLDB **10**(11), 1298–1309 (2017)
7. Akiba, T., Iwata, Y., Yoshida, Y.: Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In: CIKM, pp. 909–918 (2013)
8. Amelio, A., Pizzuti, C.: Overlapping community discovery methods: A survey. In: Social Networks: Analysis and Case Studies, pp. 105–125 (2014)
9. Andersen, R., Lang, K.J.: Communities from seed sets. In: WWW, pp. 223–232 (2006)
10. Angadi, A., Varma, P.S.: Overlapping community detection in temporal networks. Indian J. Sci. Technol. **8**(31), 1–6 (2015)
11. Archer, A., Lattanzi, S., Likarish, P., Vassilvitskii, S.: Indexing public-private graphs. In: WWW, pp. 1461–1470 (2017)
12. Armenatzoglou, N., Papadopoulos, S., Papadias, D.: A general framework for geo-social query processing. PVLDB **6**(10), 913–924 (2013)
13. Baeza-Yates, R., Hurtado, C., Mendoza, M. : Query recommendation using query logs in search engines. In: International Conference on Extending Database Technology, pp. 588–596. Springer (2004)
14. Balasundaram, B., Butenko, S., Hicks, I.V.: Clique relaxations in social network analysis: the maximum k-plex problem. Oper. Res. **59**(1), 133–142 (2011)
15. Barbieri, N., Bonchi, F., Galimberti, E., Gullo, F.: Efficient and effective community search. DMKD **29**(5), 1406–1433 (2015)
16. Barthélemy, M.: Spatial networks. Phys. Rep. **499**(1), 1–101 (2011)
17. Batagelj, V., Zaversnik, M.: An o(m) algorithm for cores decomposition of networks. arXiv:cs/0310049 (2003)
18. Batarfi, O., Shawi, R.E., Fayoumi, A.G., Nouri, R., Beheshti, S.-M.-R., Barnawi, A., Sakr, S.: Large scale graph processing

systems: survey and an experimental evaluation. Clust. Comput. **18**(3), 1189–1213 (2015)
19. Bazzi, M., Porter, M.A., Williams, S., McDonald, M., Fenn, D.J., Howison, S.D.: Community detection in temporal multilayer networks, with an application to correlation networks. Multiscale Model. Simul. **14**(1), 1–41 (2016)
20. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using banks. In: ICDE, pp. 431–440. IEEE (2002)
21. Bi, F., Chang, L., Lin, X., Zhang, W.: An optimal and progressive approach to online search of top-k influential communities. PVLDB **11**(9), 1056–1068 (2018)
22. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the web. Comput. Netw. **33**(1–6), 309–320 (2000)
23. Brunato, M., Hoos, H. H., Battiti, R.: On effectively finding maximal quasi-cliques in graphs. In: International Conference on Learning and Intelligent Optimization, pp. 41–55 (2007)
24. Cai, L., Meng, T., He, T., Chen, L., Deng, Z.: K-hop community search based on local distance dynamics. In: International Conference on Neural Information Processing, pp. 24–34 (2017)
25. Chang, L., Lin, X., Qin, L., Yu, J. X., Zhang, W.: Index-based optimal algorithms for computing Steiner components with maximum connectivity. In: SIGMOD, pp. 459–474 (2015)
26. Chang, L., Yu, J. X., Qin, L., Lin, X., Liu, C., Liang, W.: Efficiently computing k-edge connected components via graph decomposition. In: SIGMOD, pp. 205–216 (2013)
27. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: International Workshop on Approximation Algorithms for Combinatorial Optimization, pp. 84–95 (2000)
28. Chen, L., Liu, C., Zhou, R., Li, J., Yang, X., Wang, B.: Maximum co-located community search in large scale social networks. PVLDB **11**(10), 1233–1246 (2018)
29. Chen, P.-L., Chou, C.-K., Chen, M.-S. : Distributed algorithms for k-truss decomposition. In: International Conference on Big Data, pp. 471–480 (2014)
30. Chen, S., Wei, R., Popova, D., Thomo, A.: Efficient computation of importance based communities in web-scale networks using a single machine. In: CIKM, pp. 1553–1562 (2016)
31. Chen, Y., Fang, Y., Cheng, R., Li, Y., Chen, X., Zhang, J.: Exploring communities in large profiled graphs. TKDE **31**(8), 1624–1629 (2019)
32. Chen, Y., Xu, J., Xu, M.: Finding community structure in spatially constrained complex networks. Int. J. Geogr. Inf. Sci. **29**(6), 889–911 (2015)
33. Cheng, H., Zhou, Y., Huang, X., Yu, J.X.: Clustering large attributed information networks: an efficient incremental computing approach. Data Min. Knowl. Discov. **25**(3), 450–477 (2012)
34. Cheng, J., Ke, Y., Chu, S., Özsu, M.T.: Efficient core decomposition in massive networks. In: ICDE, pp. 51–62 (2011)
35. Cheng, J., Zeng, X., Yu, J. X.: Top-k graph pattern matching over large graphs. In: ICDE, pp. 1033–1044. IEEE (2013)
36. Cheng, J., Zhu, L., Ke, Y., Chu, S.: Fast algorithms for maximal clique enumeration with limited memory. In: SIGKDD, pp. 1240–1248 (2012)
37. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. **14**(1), 210–223 (1985)
38. Chierichetti, F., Epasto, A., Kumar, R., Lattanzi, S., Mirrokni, V.: Efficient algorithms for public-private social networks. In: SIGKDD, pp. 139–148. ACM (2015)
39. Chu, S., Cheng, J.: Triangle listing in massive networks and its applications. In: SIGKDD, pp. 672–680. ACM (2011)
40. Clauset, A.: Finding local community structure in networks. Phys. Rev. E **72**(2), 026132 (2005)
41. Cohen, J.: Trusses: cohesive subgraphs for social network analysis. Natl. Secur. Agency Tech. Rep. **16**, 3 (2008)
42. Conte, A., De Matteis, T., De Sensi, D., Grossi, R., Marino, A., Versari, L.: D2k: scalable community detection in massive networks via small-diameter k-plexes. In: SIGKDD, pp. 1272–1281 (2018)
43. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151–158. ACM (1971)
44. Coscia, M., Giannotti, F., Pedreschi, D.: A classification for community discovery methods in complex networks. Stat. Anal. Data Min. **4**(5), 512–546 (2011)
45. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: SIGMOD, pp. 277–288 (2013)
46. Cui, W., Xiao, Y., Wang, H., Wang, W.: Local search of communities in large graphs. In: SIGMOD, pp. 991–1002 (2014)
47. Danisch et al, M.: Listing k-cliques in sparse real-world graphs. In: WWW, pp. 589–598 (2018)
48. Danon, L., Diaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. J. Stat. Mech. Theory Exp. **2005**(09), P09008 (2005)
49. Ding, B., Yu, J. X., Wang, S., Qin, L., Zhang, X., Lin, X.: Finding top-k min-cost connected trees in databases. In: ICDE (2007)
50. Ding, L., Xie, Y., Shan, X., Song, B.: Search of center-core community in large graphs. In: CCF Conference on Big Data, pp. 94–107 (2018)
51. DiTursi, D. J., Ghosh, G., Bogdanov, P.: Local community detection in dynamic networks. arXiv preprint arXiv:1709.04033 (2017)
52. Edachery, J., Sen, A., Brandenburg, F.J.: Graph clustering using distance-k cliques. In: Proceedings of the 7th International Symposium on Graph Drawing, pp. 98–106 (1999)
53. Elzinga, J., Hearn, D.W.: Geometrical solutions for some minimax location problems. Transp. Sci. **6**(4), 379–394 (1972)
54. Expert, P., et al.: Uncovering space-independent communities in spatial networks. Proc. Natl. Acad. Sci. USA **108**(19), 7663–7668 (2011)
55. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y.: Graph pattern matching: from intractable to polynomial time. PVLDB **3**(1–2), 264–275 (2010)
56. Fan, W., Wang, X., Wu, Y., Xu, J.: Association rules with graph patterns. PVLDB **8**(12), 1502–1513 (2015)
57. Fang, Y., Cheng, R.: On attributed community search. In: International Workshop on Mobility Analytics for Spatio-temporal and Social Data, PVLDB, pp. 1–21 (2017)
58. Fang, Y., Cheng, R., Chen, Y., Luo, S., Hu, J.: Effective and efficient attributed community search. VLDB J. **26**(6), 803–828 (2017)
59. Fang, Y., Cheng, R., Cong, G., Mamoulis, N., Li, Y.: On spatial pattern matching. In: ICDE, pp. 293–304 (2018)
60. Fang, Y., Cheng, R., Li, X., Luo, S., Hu, J.: Effective community search over large spatial graphs. PVLDB **10**(6), 709–720 (2017)
61. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. PVLDB **9**(12), 1233–1244 (2016)
62. Fang, Y., Cheng, R., Luo, S., Hu, J., Huang, K.: C-explorer: browsing communities in large graphs. PVLDB **10**(12), 1885–1888 (2017)
63. Fang, Y., Cheng, R., Tang, W., Maniu, S., Yang, X.: Scalable algorithms for nearest-neighbor joins on big trajectory data. TKDE **28**(3), 785–800 (2016)
64. Fang, Y., Cheng, R., Wang, J., Budiman, L., Cong, G., Mamoulis, N.: Spacekey: exploring patterns in spatial databases. In: ICDE, pp. 1577–1580 (2018)
65. Fang, Y., Wang, Z., Cheng, R., Li, X., Luo, S., Hu, J., Chen, X.: On spatial-aware community search. TKDE **31**(4), 783–798 (2019)

66. Fang, Y., Wang, Z., Cheng, R., Wang, H., Hu, J.: Effective and efficient community search over large directed graphs. In: TKDE, p. 1 (2018)
67. Fang, Y., Yu, K., Cheng, R., Lakshmanan, L.V., Lin, X.: Efficient algorithms for densest subgraph discovery. In: PVLDB (2019)
68. Fang, Y., Zhang, H., Ye, Y., Li, X.: Detecting hot topics from twitter: a multiview approach. J. Inf. Sci. **40**(5), 578–593 (2014)
69. Fei Fan, W., Wang, X., Wu, Y.: Expfinder: finding experts by graph pattern matching. In: ICDE, pp. 1316–1319. IEEE (2013)
70. Flake, G.W., Lawrence, S., Giles, C.L. : Efficient identification of web communities. In: SIGKDD, pp. 150–160 (2000)
71. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
72. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. In: STOC, pp. 246–251 (1983)
73. Galbrun, E., Gionis, A., Tatti, N.: Top-k overlapping densest subgraphs. Data Min. Knowl. Discov. **30**(5), 1134–1165 (2016)
74. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
75. Giatsidis, C., Thilikos, D. M., Vazirgiannis, M.: D-cores: measuring collaboration of directed graphs based on degeneracy. In: ICDM, pp. 201–210 (2011)
76. Gibbons, A.: Algorithmic Graph Theory. Cambridge University Press, Cambridge (1985)
77. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA **99**(12), 7821–7826 (2002)
78. Goldberg, A.V.: Finding a Maximum Density Subgraph. University of California, Berkeley (1984)
79. Golenberg, K., Kimelfeld, B., Sagiv, Y.: Keyword proximity search in complex data graphs. In: SIGMOD, pp. 927–940. ACM (2008)
80. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: Graphx: graph processing in a distributed dataflow framework. OSDI **14**, 599–613 (2014)
81. Gregory, S.: Finding overlapping communities in networks by label propagation. New J. Phys. **12**(10), 103018 (2010)
82. Guimera, R., Amaral, L.A.N.: Functional cartography of complex metabolic networks. Nature **433**(7028), 895 (2005)
83. Gulbahce, N., Lehmann, S.: The art of community detection. BioEssays **30**(10), 934–938 (2008)
84. Guo, D.: Regionalization with dynamically constrained agglomerative clustering and partitioning (REDCAP). Int. J. Geogr. Inf. Sci. **22**(7), 801–823 (2008)
85. Guo, T., Cao, X., Cong, G.: Efficient algorithms for answering the m-closest keywords query. In: SIGMOD, pp. 405–418 (2015)
86. Guttman, A.: R-trees: a dynamic index structure for spatial searching, volume 14 (1984)
87. Hajibagheri, A., Alvari, H., Hamzeh, A., Hashemi, S.: Community detection in social networks using information diffusion. In: ASONAM, pp. 702–703 (2012)
88. Harenberg, S., Bello, G., Gjeltema, L., Ranshous, S., Harlalka, J., Seay, R., Padmanabhan, K., Samatova, N.: Community detection in large-scale networks: a survey and empirical evaluation. Wiley Interdiscip. Rev. Comput. Stat. **6**(6), 426–439 (2014)
89. Hastings, M.B.: Community detection as an inference problem. Phys. Rev. E **74**(3), 035102 (2006)
90. He, H., Wang, H., Yang, J., Yu, P. S.: Blinks: ranked keyword searches on graphs. In: SIGMOD, pp. 305–316. ACM (2007)
91. Henderson, K., Eliassi-Rad, T., Papadimitriou, S., Faloutsos, C.: HCDF: a hybrid community discovery framework. In: SDM, pp. 754–765 (2010)
92. Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms (1983)
93. Hu, J., Cheng, R., Chang, K. C., Sankar, A., Fang, Y., Lam, B.Y.H.: Discovering maximal motif cliques in large heterogeneous information networks. In: ICDE, pp. 746–757 (2019)
94. Hu, J., Cheng, R., Huang, Z., Fang, Y., Luo, S.: On embedding uncertain graphs. In: CIKM, pp. 157–166. ACM (2017)
95. Hu, J., Wu, X., Cheng, R., Luo, S., Fang, Y.: Querying minimal Steiner maximum-connected subgraphs in large graphs. In: CIKM, pp. 1241–1250 (2016)
96. Hu, J., Wu, X., Cheng, R., Luo, S., Fang, Y.: On minimal Steiner maximum-connected subgraph queries. In: TKDE, pp. 2455–2469 (2017)
97. Hu, X., Tao, Y., Chung, C.-W.: I/o-efficient algorithms on triangle listing and counting. ACM Trans. Database Syst. (TODS) **39**(4), 27 (2014)
98. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD, pp. 1311–1322 (2014)
99. Huang, X., Cheng, H., Yu, J.X.: Attributed community analysis: global and ego-centric views. IEEE Data Eng. Bull. **39**(3), 29–40 (2016)
100. Huang, X., Jiang, J., Choi, B., Xu, J., Zhang, Z., Song, Y.: PP-DBLP: modeling and generating attributed public-private networks with DBLP. In: IEEE International Conference on Data Mining Workshops (ICDMW), pp. 986–989 (2018)
101. Huang, X., Lakshmanan, L.V., Yu, J.X., Cheng, H.: Approximate closest community search in networks. PVLDB **9**(4), 276–287 (2015)
102. Huang, X., Lakshmanan, L.V.S.: Attribute-driven community search. PVLDB **10**(9), 949–960 (2017)
103. Huang, X., Lakshmanan, L.V.S., Xu, J.: Community search over big graphs: models, algorithms, and opportunities. In: ICDE, pp. 1451–1454 (2017)
104. Huang, X., Lu, W., Lakshmanan, L.V.: Truss decomposition of probabilistic graphs: semantics and algorithms. In: SIGMOD, pp. 77–90 (2016)
105. Jayaram, N., Goyal, S., Li, C.: VIIQ: auto-suggestion enabled visual interface for interactive graph query formulation. PVLDB **8**(12), 1940–1943 (2015)
106. Jiang, Y., Huang, X., Cheng, H., Yu, J. X.: VizCS: online searching and visualizing communities in dynamic graphs. In: ICDE, pp. 1585–1588 (2018)
107. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. In: VLDB, pp. 505–516. VLDB Endowment (2005)
108. Kargar, M., An, A.: Keyword search in graphs: finding r-cliques. PVLDB **4**(10), 681–692 (2011)
109. Karypis, G., Kumar, V.: Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. (1995)
110. Khan, B.S., Niazi, M.A.: Network community detection: a review and visual survey. arXiv:1708.00977 (2017)
111. Khaouid, W., Barsky, M., Srinivasan, V., Thomo, A.: K-core decomposition of large networks on a single PC. PVLDB **9**(1), 13–23 (2015)
112. Kim, J., Lee, J.-G.: Community detection in multi-layer graphs: a survey. SIGMOD Rec. **44**(3), 37–48 (2015)
113. Kim, Y., Son, S.-W., Jeong, H.: Finding communities in directed networks. Phys. Rev. E **81**(1), 016103 (2010)
114. Kloumann, I.M., Kleinberg, J.M.: Community membership identification from small seed sets. In: SIGKDD, pp. 1366–1375 (2014)
115. Kou, L., Markowsky, G., Berman, L.: A fast algorithm for Steiner trees. Acta Inf. **15**(2), 141–145 (1981)
116. Kuncheva, Z., Montana, G.: Multi-scale community detection in temporal networks using spectral graph wavelets. In: International Workshop on Personal Analytics and Privacy, pp. 139–154 (2017)

117. Lai, L., Qin, L., Lin, X., Chang, L.: Scalable subgraph enumeration in mapreduce. PVLDB **8**(10), 974–985 (2015)
118. Lai, L., Qin, L., Lin, X., Zhang, Y., Chang, L., Yang, S.: Scalable distributed subgraph enumeration. PVLDB **10**(3), 217–228 (2016)
119. Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys. Rev. E **80**(1), 016118 (2009)
120. Lee, J., Chung, C.: A query approach for influence maximization on specific users in social networks. TKDE **27**(2), 340–353 (2015)
121. Leicht, E.A., Newman, M.E.: Community structure in directed networks. Phys. Rev. Lett. **100**(11), 118703 (2008)
122. Leighton, T., Rao, S.: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: FOCS, pp. 422–431 (1988)
123. Leskovec, J., Lang, K.J., Mahoney, M.: Empirical comparison of algorithms for network community detection. In: WWW, pp. 631–640 (2010)
124. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: SIGMOD, pp. 903–914. ACM (2008)
125. Li, J., Wang, X., Deng, K., Yang, X., Sellis, T., Yu, J.X.: Most influential community search over large social networks. In: ICDE, pp. 871–882 (2017)
126. Li, R.-H., Qin, L., Ye, F., Yu, J. X., Xiao, X., Xiao, N., Zheng, Z.: Skyline community search in multi-valued networks. In: SIGMOD, pp. 457–472 (2018)
127. Li, R.-H., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. PVLDB **8**(5), 509–520 (2015)
128. Li, R.-H., Qin, L., Yu, J.X., Mao, R.: Finding influential communities in massive networks. VLDB J. **26**(6), 751–776 (2017)
129. Li, R.-H., Su, J., Qin, L., Yu, J. X., Dai, Q.: Persistent community search in temporal networks. In: ICDE, pp. 797–808 (2018)
130. Li, R.-H., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. TKDE **26**(10), 2453–2465 (2014)
131. Li, X., Cheng, R., Fang, Y., Hu, J., Maniu, S.: Scalable evaluation of k-NN queries on large uncertain graphs. In: EDBT, pp. 181–192 (2018)
132. Li, Y., Sha, C., Huang, X., Zhang, Y.: Community detection in attributed graphs: an embedding approach. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
133. Li, Z., Fang, Y., Liu, Q., Cheng, J., Cheng, R., Lui, J.: Walking in the cloud: parallel SimRank at scale. PVLDB **9**(1), 24–35 (2015)
134. Liu, S., Wang, S., Krishnan, R.: Persistent community detection in dynamic social networks. In: PAKDD, pp. 78–89 (2014)
135. Liu, Y., Niculescu-Mizil, A., Gryc, W.: Topic-link LDA: joint models of topic and author community. In: International Conference on Machine Learning, pp. 665–672 (2009)
136. Luo, F., Wang, J.Z., Promislow, E.: Exploring local community structures in large networks. In: ICWI, pp. 233–239 (2006)
137. Macropol, K., Singh, A.: Scalable discovery of best clusters on large graphs. PVLDB **3**(1–2), 693–702 (2010)
138. Malewicz, G., Austern, M. H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: SIGMOD, pp. 135–146. ACM (2010)
139. Malliaros, F.D., Vazirgiannis, M.: Clustering and community detection in directed networks: a survey. Phys. Rep. **533**(4), 95–142 (2013)
140. Marcel, P., Negre, E.: A survey of query recommendation techniques for data warehouse exploration. In: EDA, pp. 119–134 (2011)
141. Matsuda, H., Ishihara, T., Hashimoto, A.: Classifying molecular sequences using a linkage graph with their pairwise similarities. Theor. Comput. Sci. **210**(2), 305–325 (1999)
142. Mehler, A., Skiena, S.: Expanding network communities from representative examples. TKDD **3**(2), 7 (2009)
143. Mehlhorn, K.: A faster approximation algorithm for the steiner problem in graphs. Inf. Process. Lett. **27**, 125–128 (1988)
144. Meng, T., Cai, L., He, T., Chen, L., Deng, Z.: K-hop community search based on local distance dynamics. KSII Trans. Internet Inf. Syst. **12**(7) (2018)
145. Montresor, A., De Pellegrini, F., Miorandi, D.: Distributed k-core decomposition. IEEE Trans. Parallel Distrib. Syst. **24**(2), 288–300 (2013)
146. Moradi, F., Olovsson, T., Tsigas, P.: A local seed selection algorithm for overlapping community detection. In: ASONAM, pp. 1–8 (2014)
147. Nallapati, R.M., Ahmed, A., Xing, E.P., Cohen, W.W.: Joint latent topic models for text and citations. In: SIGKDD, pp. 542–550 (2008)
148. Newman, M.E.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(6), 066133 (2004)
149. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**(2), 026113 (2004)
150. Ning, X., Liu, Z., Zhang, S.: Local community extraction in directed networks. Phys. A Stat. Mech. Appl. **452**, 258–265 (2016)
151. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**, 814–818 (2005)
152. Papadopoulos, S., Kompatsiaris, Y., Vakali, A., Spyridonos, P.: Community detection in social media. DMKD **24**(3), 515–554 (2012)
153. Park, H.-M., Myaeng, S.-H., Kang, U.: Pte: enumerating trillion triangles on distributed systems. In: SIGKDD, pp. 1115–1124. ACM (2016)
154. Parthasarathy, S., Ruan, Y., Satuluri, V.: Community discovery in social networks: applications, methods and emerging trends. In: Social Network Data Analytics, pp. 79–113 (2011)
155. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: SIGKDD, pp. 701–710 (2014)
156. Plantié, M., Crampes, M.: Survey on social community detection. In: Social Media Retrieval, pp. 65–85 (2013)
157. Pons, P., Latapy, M.: Computing communities in large networks using random walks. In: International Symposium on Computer and Information Sciences, pp. 284–293 (2005)
158. Porter, M.A., Onnela, J.-P., Mucha, P.J.: Communities in networks. Not. AMS **56**(9), 1082–1097 (2009)
159. Qi, G.-J., Aggarwal, C.C., Huang, T.S.: Online community detection in social sensing. In: WSDM, pp. 617–626 (2013)
160. Qiao, M., Zhang, H., Cheng, H.: Subgraph matching: on compression and computation. Proc. VLDB Endow. **11**(2), 176–188 (2017)
161. Qin, L., Li, R.-H., Chang, L., Zhang, C.: Locally densest subgraph discovery. In: SIGKDD, pp. 965–974 (2015)
162. Qin, L., Yu, J. X., Chang, L., Tao, Y.: Querying communities in relational databases. In: ICDE (2009)
163. Rossetti, G., Cazabet, R.: Community discovery in dynamic networks: a survey. ACM Comput. Surv. **51**(2), 35:1–35:37 (2018)
164. Ruan, Y., Fuhry, D., Parthasarathy, S.: Efficient community detection in large networks using content and links. In: WWW, pp. 1089–1098 (2013)
165. Sachan, M., Contractor, D., Faruquie, T.A., Subramaniam, L.V.: Using content and interactions for discovering communities in social networks. In: WWW, pp. 331–340 (2012)
166. Saito, K., Yamada, T., Kazama, K.: Extracting communities from complex networks by the k-dense method. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **91**(11), 3304–3311 (2008)
167. Sarıyüce, A.E., Gedik, B., Jacques-Silva, G., Wu, K.-L., Çatalyürek, Ü.V.: Incremental k-core decomposition: algorithms and evaluation. VLDB J. **25**(3), 425–447 (2016)
168. Sariyüce, A.E., Pinar, A.: Fast hierarchy construction for dense subgraphs. PVLDB **10**(3), 97–108 (2016)

169. Sariyuce, A.E., Seshadhri, C., Pinar, A., Catalyurek, U.V.: Finding the hierarchy of dense subgraphs using nucleus decompositions. In: WWW, pp. 927–937 (2015)
170. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
171. Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. J. Math. Sociol. **6**(1), 139–154 (1978)
172. Shakarian, P., Roos, P., Callahan, D., Kirk, C.: Mining for geographically disperse communities in social networks by leveraging distance modularity. In: SIGKDD, pp. 1402–1409 (2013)
173. Shang, J., Wang, C., Wang, C., Guo, G., Qian, J.: An attribute-based community search method with graph refining. J. Supercomput. 1–28 (2017)
174. Shi, C., Li, Y., Zhang, J., Sun, Y., Philip, S.Y.: A survey of heterogeneous information network analysis. IEEE Trans. Knowl. Data Eng. **29**(1), 17–37 (2017)
175. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: SIGKDD, pp. 939–948 (2010)
176. Subbian, K., Aggarwal, C.C., Srivastava, J., Yu, P.S.: Community detection with prior knowledge. In: SDM, pp. 405–413 (2013)
177. Tamimi, I., El Kamili, M.: Literature survey on dynamic community detection and models of social networks. In: International Conference on Wireless Networks and Mobile Communications, pp. 1–5 (2015)
178. Tang, L., Liu, H.: Scalable learning of collective behavior based on sparse social dimensions. In: CIKM, pp. 1107–1116 (2009)
179. Tong, H., Faloutsos, C., Gallagher, B., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In: KDD, pp. 737–746. ACM (2007)
180. Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., Tsiarli, M.: Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: SIGKDD, pp. 104–112 (2013)
181. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM (JACM) **23**(1), 31–42 (1976)
182. Von Luxburg, U.: A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
183. Wang, H., Aggarwal, C.C.: A survey of algorithms for keyword search on graph data. In: Managing and Mining Graph Data, pp. 249–273. Springer (2010)
184. Wang, J., Cheng, J.: Truss decomposition in massive networks. PVLDB **5**(9), 812–823 (2012)
185. Wang, K., Cao, X., Lin, X., Zhang, W., Qin, L.: Efficient computing of radius-bounded k-cores. In: ICDE, pp. 233–244 (2018)
186. Wang, N., Zhang, J., Tan, K.-L., Tung, A.K.: On triangulation-based dense neighborhood graph discovery. PVLDB **4**(2), 58–68 (2010)
187. Wang, Y., Jian, X., Yang, Z., Li, J.: Query optimal k-plex based community in graphs. Data Sci. Eng. **2**(4), 257–273 (2017)
188. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/o efficient core graph decomposition: application to degeneracy ordering. IEEE Trans. Data Eng. **31**(1), 75–90 (2019)
189. Wu, F.-Y.: The potts model. Rev. Mod. Phys. **54**(1), 235 (1982)
190. Wu, Y., Jin, R., Li, J., Zhang, X.: Robust local community detection: on free rider effect and its elimination. PVLDB **8**(7), 798–809 (2015)
191. Wu, Y., Jin, R., Zhu, X., Zhang, X.: Finding dense and connected subgraphs in dual networks. In: ICDE, pp. 915–926 (2015)
192. Xu, Z., Ke, Y., Wang, Y., Cheng, H., Cheng, J.: A model-based approach to attributed graph clustering. In: SIGMOD, pp. 505–516 (2012)
193. Yang, B., Cheung, W., Liu, J.: Community mining from signed social networks. IEEE Trans. Knowl. Data Eng. **19**(10), 1333–1348 (2007)
194. Yang, B., Liu, D., Liu, J.: Discovering communities from social networks: methodologies and applications, pp. 331–346 (2010)
195. Yang, D.-N., Chen, Y.-L., Lee, W.-C., Chen, M.-S.: On social–temporal group query with acquaintance constraint. PVLDB **4**(6), 397–408 (2011)
196. Yang, D.-N., Shen, C.-Y., Lee, W.-C., Chen, M.-S.: On socio-spatial group query for location-based social networks. In: SIGKDD, pp. 949–957 (2012)
197. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowl. Inf. Syst. **42**(1), 181–213 (2015)
198. Yang, J., McAuley, J., Leskovec, J.: Community detection in networks with node attributes. In: ICDM, pp. 1151–1156 (2013)
199. Yang, J., McAuley, J., Leskovec, J.: Detecting cohesive and 2-mode communities indirected and undirected networks. In: WSDM, pp. 323–332 (2014)
200. Yang, L., Cao, X., He, D., Wang, C., Wang, X., Zhang, W.: Modularity based community detection with deep learning. In: IJCAI, pp. 2252–2258 (2016)
201. Yang, T., Chi, Y., Zhu, S., Gong, Y., Jin, R.: Directed network community detection: a popularity and productivity link model. In: SDM, pp. 742–753 (2010)
202. Yang, T., Jin, R., Chi, Y., Zhu, S.: Combining link and content for community detection: a discriminative approach. In: SIGKDD, pp. 927–936 (2009)
203. Yi, P., Choi, B., Bhowmick, S.S., Xu, J.: AutoG: a visual query autocompletion framework for graph databases. VLDB J. **26**(3), 347–372 (2017)
204. Yu, J.X., Qin, L., Chang, L.: Keyword Search in Databases. Synthesis Lectures on Data Management (2009)
205. Yuan, L., Qin, L., Zhang, W., Chang, L., Yang, J.: Index-based densest clique percolation community search in networks. TKDE **30**(5), 922–935 (2018)
206. Yuan, Y., Lian, X., Chen, L., Yu, J.X., Wang, G., Sun, Y.: Keyword search over distributed graphs with compressed signature. TKDE **29**(6), 1212–1225 (2017)
207. Yuan, Y., Wang, G., Chen, L., Wang, H.: Efficient subgraph similarity search on large probabilistic graph databases. PVLDB **5**(9), 800–811 (2012)
208. Yuan, Y., Wang, G., Chen, L., Wang, H.: Efficient keyword search on uncertain graph data. TKDE **25**(12), 2767–2779 (2013)
209. Yuan, Y., Wang, G., Wang, H., Chen, L.: Efficient subgraph search over large uncertain graphs. PVLDB **4**(11), 876–886 (2011)
210. Zhang, F., Yuan, L., Zhang, Y., Qin, L., Lin, X., Zhou, A.: Discovering strong communities with user engagement and tie strength. In: DASFAA, pp. 425–441 (2018)
211. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: efficient (k, r)-core computation on social networks. PVLDB **10**(10), 998–1009 (2017)
212. Zhang, Y., Parthasarathy, S.: Extracting analyzing and visualizing triangle k-core motifs within networks. In: ICDE, pp. 1049–1060 (2012)
213. Zhang, Y., Yu, J. X., Zhang, Y., Qin, L.: A fast order-based approach for core maintenance. In: ICDE, pp. 337–348 (2017)
214. Zhao, F., Tung, A.K.: Large scale cohesive subgraphs discovery for social network visual analysis. PVLDB **6**, 85–96 (2012)
215. Zheng, D., Liu, J., Li, R.-H., Aslay, C., Chen, Y.-C., Huang, X.: Querying intimate-core groups in weighted graphs. In: IEEE International Conference on Semantic Computing, pp. 156–163. IEEE (2017)
216. Zheng, Z., Ye, F., Li, R.-H., Ling, G., Jin, T.: Finding weighted k-truss communities in large networks. Inf. Sci. **417**(C), 344–360 (2017)
217. Zhou, D., Councill, I., Zha, H., Giles, C.L.: Discovering temporal communities from social network documents. In: ICDM, pp. 745–750 (2007)

218. Zhou, R., Liu, C., Yu, J. X., Liang, W., Chen, B., Li, J.: Finding maximal k-edge-connected subgraphs from a large graph. In: EDBT, pp. 480–491 (2012)
219. Zhou, R., Liu, C., Yu, J. X., Liang, W., Zhang, Y.: Efficient truss maintenance in evolving networks. arXiv preprint arXiv:1402.2807 (2014)
220. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. PVLDB **2**(1), 718–729 (2009)
221. Zhu, Q., Hu, H., Xu, C., Xu, J., Lee, W.-C.: Geo-social group queries with minimum acquaintance constraints. VLDB J. **26**(5), 709–727 (2017)
222. Zhu, R., Zou, Z., Li, J.: Diversified coherent core search on multi-layer graphs. In: ICDE, pp. 701–712. IEEE (2018)
223. Zou, L., Chen, L., Özsu, M.T.: Distance-join: pattern match query in a large graph database. PVLDB **2**(1), 886–897 (2009)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.