MORGAN & CLAYPOOL PUBLISHERS

Community Search over Big Graphs

Xin Huang
Laks V.S. Lakshmanan
Jianliang Xu

SYNTHESIS LECTURES ON DATA MANAGEMENT

H.V. Jagadish, Series Editor

CHAPTER 2
Cohesive Subgraphs

In many real applications where information is modeled using graphs, communities are formed by a set of similar entities that are densely connected with certain relationships. Massive networks can often be understood and analyzed in terms of these communities [165]. In the literature, several different models for cohesive (dense) subgraphs and communities have been proposed, which we will discuss in Chapters 3–6. Specifically, dense subgraphs of various kinds are often used as building blocks of communities. In this chapter, we will review various kinds of dense/cohesive subgraphs, including clique [31, 45, 166, 182], quasi-clique [162], k-DBDSG [132], k-clan [132], k-club [132], k-plex [165], k-core [19, 44], k-truss [50, 165], k-ecc [38], and densest subgraphs [15, 106]. We also investigate their relationships and compare their structural properties using illustrative examples.

2.1 COMMUNITY SEARCH AND COHESIVE SUBGRAPHS

This book focuses on the community models based on cohesive subgraphs. Unlike community detection, there are three desirable properties for community search models: cohesive structure for high-quality communities, support for easy querying and personalization, and fast algorithms for efficient query processing. The detailed reasons are as follows.

- In real-world applications, communities usually have many kinds of topological structures in different shapes. This brings significant challenges for developing a perfect model to fit all of the variously shaped community structures. However, the intuition that community search shares with community discovery is that community members often have densely connected relationships with members of the same community, and have seldom connections with others beyond this community. This observation naturally motivates the model of community as dense subgraphs. In addition, diameter and connectivity have been considered as important features for modeling communities. Small diameter and high connectivity are proposed as criteria for good communities in [61, 81]. Small diameter guarantees that any two vertices in a community can be found within a short distance of each other. High connectivity ensures the connectivity between vertices in the community is strong and robust.

- Besides good structural properties, high efficiency is highly desirable for community search. This is because community search, being query driven, should be answered in an efficient way, so as to allow the user to interactively modify her query and explore the resulting communities found. There exist several classical dense subgraphs, including clique [31], quasi-clique [162], k-DBDSG [132], k-clan [132], k-club [132], and k-plex [165]. In principle, any of these dense subgraphs could be used as a basis for community search. However, one drawback of these dense subgraph models is that they are NP-hard to discover. Consequently, the search of communities based on such dense subgraphs given a query, is inefficient and intractable for real-time query processing. Recently proposed dense subgraphs k-core [19, 44] and k-truss [50, 165] make real-time retrieval of communities possible, since they can be computed in polynomial time. Moreover, using a single parameter of k, we can get a hierarchical community structure of a query vertex [89]. Polynomial-time complexity for computing cohesive subgraphs of k-core and k-truss is a game-changer in the field of community search, which attracts significant interest in the study of problems related to community search in social and information networks [18, 55, 64, 65, 89, 93, 96, 122–124, 157, 199].

In the following sections, we first introduce the basic concepts of graph theory in Section 2.2. Then, we introduce various kinds of classical dense subgraphs, which are NP-hard to compute, in Section 2.3. Next, we present the definitions of k-core and k-truss, recently widely used in community search, which have a polynomial-time complexity, in Section 2.4. We not only introduce their concepts but also their decomposition algorithms, which are particularly useful in building indexes for speeding up the query processing. Finally, more dense subgraphs including the densest subgraph and k-ecc are introduced in Section 2.5.

2.2 NOTATIONS AND NOTIONS

We start with a brief introduction of the notations and notions used in this book.

2.2.1 GRAPHS AND SUBGRAPHS

A graph $G(V, E)$ is a set of nodes (or vertices) V together with a set of lines (edges) E. Any line of E connects a pair of nodes, say u, v: we denote this as an edge $(u, v) \in E$ and say that u and v are adjacent to each other in G. We also denote the number of vertices and the number of edges, respectively, as $n = |V|$ and $m = |E|$. A self-loop is an edge connecting a node to itself. Unless otherwise specified, all graphs discussed in this book are simple graphs: finite, non-empty, undirected, and having no self-loops. In addition, a complete graph K_p is a graph of p vertices, where every pair of vertices is adjacent to each other.

Figure 2.1 shows an example of graph $G(V, E)$ with 10 nodes and 14 edges, where $|V| = |\{v_1, v_2, \ldots, v_{10}\}| = 10$ and $|E| = 14$. As we can see, G is a simple and undirected graph without self-loops.

A graph $H = (V(H), E(H))$ is a subgraph of graph $G(V, E)$ denoted $H \subset G$, iff $V(H) \subseteq V$ and $E(H) \subseteq E$. In addition, when a graph H is a subgraph of graph G, we say that graph G is a supergraph of H. Furthermore, we give the definition of induced subgraph as follows.

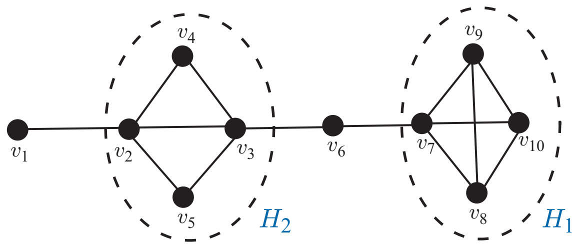

Figure 2.1: An example of graph G with 10 nodes and 14 edges. Two subgraphs H_1 and H_2 of G.

Definition 2.2.1 (Induced Graph) *Given a graph $G(V, E)$ and a vertex set $S \subseteq V$, the induced subgraph of G by S is $G_S = (S, E_S)$ where the edge set $E_S = \{(v, u) : v, u \in S, (v, u) \in E\}$.*

In addition, we define the maximality using supergraphs. A subgraph $H \subseteq G$ satisfying a given property is maximal provided no proper supergraph of H, that is also a subgraph of G, satisfies that property.

In Figure 2.1, the subgraph H_1 of G consists of 4 vertices and 6 edges, i.e., the vertex set $V(H_1) = \{v_7, v_8, v_9, v_{10}\}$, and the edge set $E(H_1) = \{(v_7, v_8), (v_7, v_9), (v_7, v_{10}), (v_8, v_9), (v_8, v_{10}), (v_9, v_{10})\}$. As we can see, for any pair of vertices in H_1, there exists an edge connecting them, indicating H_1 is a complete subgraph of 4 nodes, i.e., a 4-clique. In addition, since we cannot find any supergraph H' containing H_1 to be a clique, H_1 is the maximal complete subgraph K_4 of G.

2.2.2 DEGREE AND NEIGHBORS

The set of neighbors of a vertex v in graph G is denoted by $N_G(v)$, i.e., $N_G(v) = \{u \in V : (v, u) \in E(G)\}$. The degree of a vertex v in G is the number of neighbors (vertices) adjacent to v, denoted by $d_G(v) = |N_G(v)|$. When the context is obvious, we drop the subscript and denote the set of neighbors as $N(v)$ and the degree as $d(v)$. We use $d_{\max} = \max_{v \in V} d(v)$ to denote the maximum vertex degree in G. Given a subgraph H of G, we use $N_H(v)$ to represent the set of neighbors to v in the subgraph H only. Similar extensions may be made to degree, path, and other definitions below.

For example, in the graph G shown in Figure 2.1, vertex v_7 has 4 neighbors $N(v_7) = \{v_6, v_8, v_9, v_{10}\}$, thus the degree of v_7 is 4 as $d(v_7) = 4$. In the subgraph H_1 of G, v_7 has 3 neighbors as $N_{H_1}(v_7) = \{v_8, v_9, v_{10}\}$ and $d_{H_1}(v_7) = 3$.

2.2.3 PATH, CYCLE, CONNECTIVITY, AND DIAMETER

A path, connecting two vertices v, u of a subgraph H, consists of a series of vertices v, w_1, w_2, …, w_{l-1}, $u \in V(H)$, such that $(v, w_1) \in E(H)$, $(w_i, w_{i+1}) \in E(H)$ for $1 \leq i < l - 1$, and $(w_{l-1}, u) \in E(H)$. The length l of a path is given by the number of its edges.

A cycle C_l of length l is a path of length l, where $u = v$. A triangle in G is a cycle of length 3. We denote a triangle involving vertices $u, v, w \in V$ as $\triangle_{uvw}$.

Consider the graph G in Figure 2.1 and two vertices v_8, v_9, one path P connecting v_8 and v_9 can be $P =< v_8, v_7, v_9 >$ which consists of two edges (v_8, v_7) and (v_7, v_9). The length of P is 2. Moreover, the cycle formed by vertices v_7, v_8, v_9 is a triangle $\triangle_{v_7 v_8 v_9}$.

A subgraph H of G is connected if every pair of vertices $u, v \in H$ is connected by a path in H. W.l.o.g we assume in this book that the graph G we consider is connected. Note that this implies that $m \geq n - 1$.

Furthermore, we give the definition of the shortest path. For two nodes $u, v \in G$, we denote by $\mathsf{dist}_G(u, v)$ the length of the shortest path between u and v in G, where $\mathsf{dist}_G(u, v) = +\infty$ if u and v are not connected. Based on the concept of the shortest path, we can define graph diameter as follows. The diameter of a graph G is defined as the maximum length of a shortest path in G, i.e., $\mathsf{diam}(G) = \max_{u,v \in G}\{\mathsf{dist}_G(u, v)\}$. One well-known relation will be frequently used in this book, that is, if H is a subgraph G, then for every pair of vertices $v, u \in H$, $\mathsf{dist}_G(u, v) \leq \mathsf{dist}_H(u, v)$. In other words, the distance between any two vertices in a subgraph of G cannot be shorter than their distance in G itself.

Continuing the above example in Figure 2.1, we know that the shortest path between v_8 and v_9 is the edge (v_8, v_9) of length as 1, thus the path $P =< v_8, v_7, v_9 >$ is not the shortest path as $|P| = 2$. Considering the graph G, the shortest path between v_1 and v_{10} has 5 edges, $\mathsf{dist}_G(v_1, v_{10}) = 5$, which is the longest shortest path in G. As such, the diameter of G is $\mathsf{diam}(G) = \max_{v,u \in G}\{\mathsf{dist}_G(v, u)\} = 5$. In addition, consider the subgraph $H = H_1 \cup H_2$, the distance between v_2 and v_7 in graph H is $\mathsf{dist}_H(v_2, v_7) = +\infty$.

2.3 CLASSICAL DENSE SUBGRAPHS

A community (group) is usually regarded as a set of members that are interconnected with dense substructures. For example, in social networks, there are several concepts associated with communities that indicate various types and configurations of social groups, such as social circles, peer groups, coteries, and so on [132]. To effectively represent such more or less closely knit groups, several definitions of subgraphs are developed with the help of graph theory and network science. A well-known concept of peer group is the clique: a group all members of which are in contact with, or are friends with, or know each other. However, relaxed concepts of cliques are also necessary to denote less loosely knit, yet significantly homogeneous social groups, e.g., for every pair of members, even if they are not in mutual contact, have multiple common third contacts [132].

There is a large body of work on mining dense subgraph patterns, including clique [31, 45, 166, 182], quasi-clique [162], k-DBDSG, k-clan, k-club, k-core [19, 44], k-truss [50, 165, 192], dense neighborhood graph [168], to name a few. In the following, we first introduce cliques and their relaxed variants.

2.3.1 CLIQUE AND QUASI-CLIQUE

We start with the well-known notion of k-cliques.

Definition 2.3.1 (k-Clique) *A k-clique is a complete subgraph of k vertices, where every pair of vertices is adjacent.*

A k-clique is the densest graph among all k-node graphs. Let G be the graph shown in Figure 2.2, with subgraphs H_1 (Figure 2.2a) and H_2 (Figure 2.2b). H_1 is a 4-clique, while H_2 misses being one by just one edge. Nevertheless, the vertices of H_2 form a closely knit cohesive subgraph. Another dense subgraph called γ-quasi-k-clique has been introduced for capturing subgraphs that are "close" to being cliques, based on the notion of edge density, defined next.

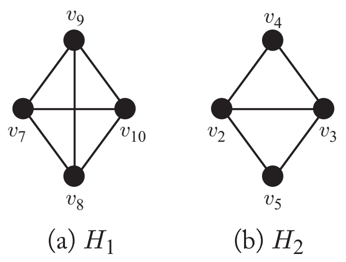

Figure 2.2: Examples of k-clique as H_1 and γ-quasi-k-clique as H_2 where $k = 4$ and $\gamma = 0.8$.

Definition 2.3.2 (Edge Density) *Given a graph $G(V, E)$, the edge density of G is defined as* $\text{den}(G) = \frac{2|E|}{|V|(|V|-1)}$.

By tuning the edge density of the graph using a parameter γ, a generalization of k-clique, called γ-quasi-k-clique is defined as follows.

Definition 2.3.3 (γ-Quasi-k-Clique) *A γ-quasi-k-clique is a graph of k vertices with at least $\lfloor \gamma \frac{k(k-1)}{2} \rfloor$ edges where $0 \leq \gamma \leq 1$.*

In Figure 2.2, H_2 has 4 vertices and 5 edges, and the edge density of H_2 is $\text{den}(H_2) = \frac{2*5}{4*(4-1)} = \frac{5}{6} = 0.833$. Thus, H_2 is a 0.8-quasi-4-clique where $\text{den}(H_2) \geq 0.8$.

Clique and quasi-clique enumeration methods include the classical algorithm [31], the external-memory H^*-graph algorithm [45], redundancy-aware clique enumeration [166], maximum clique computation using MapReduce [182], and optimal quasi-clique mining [162].

2.3.2 k-DBDSG, k-CLAN, k-CLUB, AND k-PLEX

Other kinds of dense subgraphs proposed in the literature include k-DBDSG, k-clan, k-club, and also k-plex.

Definition 2.3.4 (k-distance bounded dense subgraph [132]) *A k-distance bounded dense subgraph (k-DBDSG) is a loose definition of a clique on n vertices. Specifically, a k-DBDSG of a graph G is a maximal subgraph $H \subseteq G$ such that every pair of vertices $u, v \in V(H)$ is at most a distance k apart in G, i.e.,* $\mathsf{dist}_G(u, v) \leq k, \forall u, v \in V(H)$.

Note that, due to the maximality of H, for each node $x \in V(G) \setminus V(H)$, there exists a node $v \in V(H)$ with $\mathsf{dist}_G(x, v) > k$.

From this definition, it can be seen that the k-DBDSG is a global notion, in that it is based on the overall structure of the network, i.e., based on the entire graph and reflected in its distance matrix. We note that k-DBDSG was originally defined in [132], where the term n-clique was used to describe it, where n is the bound on the distance between pairs of nodes. Owing to the obvious confusion this may cause with the standard notion of k-clique, we have changed the terminology to k-DBDSG.

Notice that the bound k is on the distance between vertices of H, where the distance is measured in the original graph G. Consequently, the diameter of H may exceed k. For example, consider the graph G in Figure 2.3a, and the two subgraphs H_1 and H_2 of G presented in Figures 2.3b and 2.3c. First, both subgraphs are 2-DBDSG's, as $\mathsf{dist}_G(u, v) \leq 2$ holds for any pair of vertices u, v in H_1 and H_2. However, the diameter of H_1 is 3 as the shortest path connecting v_4 and v_5 passes through vertices v_2 and v_3. On the other hand, the diameter of H_2 is 2.

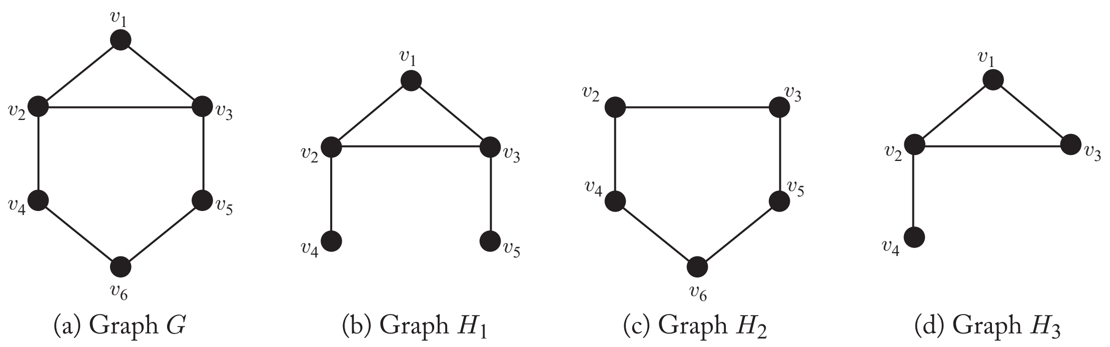

Figure 2.3: Several examples of k-DBDSGs, k-clans, k-clubs, and also k-plexes. Here $k = 2$. H_1 is a k-DBDSG of G, but not a k-clan. H_2 is a k-DBDSG, a k-clan, a k-club, and also a 3-plex with 5 vertices. H_3 is a k-club, but not a k-DBDSG or a k-clan.

Definition 2.3.5 (k-clan [132]) *A k-clan H of a graph G is a k-DBDSG of G such that for every pair of vertices v, u in H, the distance in H is at most k, i.e.,* $\mathsf{dist}_H(v, u) \leq k$.

In the above example, H_1 is not a 2-clan, since the distance $\mathsf{dist}_{H_1}(v_4, v_5) = 3 > 2$. In contrast, H_2 is a k-clan and k-DBDSG for $k = 2$. Notice that since a k-clan is a 2-DBDSG, it is required to be maximal.

Consequently, a k-clan H of G satisfies the following conditions:
(1) for all vertices $u, v \in V(H)$: $\mathsf{dist}_H(u, v) \leq k$; and
(2) for all vertices $w \in V(G) \setminus V(H)$, there exist a vertex $u \in V(H)$ with $\mathsf{dist}_G(u, w) > k$.

It is easy to see that a k-clan is a stronger notion than a k-DBDSG. By definition, k-clans are k-DBDSGs of diameter at most k in H.

Definition 2.3.6 (k-club [132]) *A k-club H of a graph G is a maximal subgraph of G with diameter at most k, i.e.,* $\mathsf{diam}(H) \leq k$.

k-clan **v.s.** k-club While the definitions of k-club and k-clan look very similar, their maximality conditions are issued on different distance constraints. The distance function of k-club focuses on the local subgraph H, which requires the maximal subgraph achieving $\mathsf{dist}_H(v, u) \leq k$. On the other hand, a k-clan H is first a k-DBDSG by definition. Thus, the distance function of k-clan focuses on the global graph G, which requires the maximal subgraph achieving $\mathsf{dist}_G(v, u) \leq k$. The following example shows the difference of k-clan and k-club.

Example 2.3.1 *Consider the subgraph H_3 in Figure 2.3d of graph G in Figure 2.3a. H_3 is a k-club (k=2), since* $\mathsf{diam}(H_3) = 2$ *and there exists no supergraph H' of H_3 with* $\mathsf{diam}(H') = 2$. *On the other hand, H_3 is not a 2-DBDSG (k=2) since it violates the maximality constraint; as a supergraph of H_3, H_1 is a 2-DBDSG (k=2). Hence, H_3 is not k-clan (k=2) by definition.*

The notion of k-plex relaxes the degree of each vertex within a clique of n vertices from $(n-1)$ to $(n-k)$ [165].

Definition 2.3.7 (k-plex) *A k-plex of size n is a subgraph H of size n where each vertex is adjacent to at least $n-k$ vertices in H.*

For example, H_2 in Figure 2.3c is a 3-plex of size 5, as each vertex has at least $(5-3) = 2$ neighbors in H_2.

Comparisons and Algorithms. All the above-mentioned cohesive subgraphs are relatively small substructures in a graph. The basic ones are the cliques (i.e., complete subgraphs) and maximal cliques. As the definition of clique is often too rigid, more relaxed forms of cohesive subgraphs have been studied. The k-DBDSG relaxes the distance between any two vertices in a clique from 1 to k. The k-clan is the same as the k-DBDSG except for imposing a constraint on the diameter. The k-club removes the k-DBDSG requirement from the k-clan. The k-plex relaxes the degree of each vertex within a clique of n vertices from $n-1$ to $n-k$. The quasi-clique can be either a relaxation on the density or the degree. However, the computation of all the above cohesive subgraphs is NP-hard [132, 162, 165].

The algorithms typically need to enumerate the subgraphs corresponding to all subsets of vertices for finding all maximal cliques, quasi-cliques, k-DBDSGs, k-clans, k-clubs, and k-plexes [132].

2.4 k-CORE AND k-TRUSS

In this section, we introduce two common dense subgraph definitions of k-core and k-truss. In addition, we present algorithms for core decomposition and truss decomposition on graphs, which can efficiently find k-core and k-truss for any possible value k.

2.4.1 k-CORE

The k-core of a graph G is the largest subgraph of G in which every vertex is adjacent to at least k other vertices within the subgraph. In other words, we can define k-core as follows.

Definition 2.4.1 *[K-Core] Given a graph G and an integer k, a k-core H of G is the largest subgraph such that $\forall v \in V(H)$, $\deg_H(v) \geq k$.*

If a vertex v is present in the k-core subgraph, but not in the $(k+1)$-core subgraph, we say the core number of v is k. We next give the formal definition of *core number*.

Definition 2.4.2 *[Core Number] The* core number *of a vertex $v \in V$, denoted $\varphi(v)$, is the maximum integer k for which there exists a k-core of G that contains v.*

We denote the maximum core number of any vertex in a graph as $c_{\max}$. Based on the core number of vertices, we can partition vertices into different classes. The k-class of G, $\Psi(k)$, is defined as $\Psi(k) = \{v : v \in V, \varphi(v) = k\}$.

We illustrate the concepts of k-core and k-class using an example. Consider the graph G in Figure 2.4. The whole graph G is a 1-core, since each vertex has degree at least 1. The 1-class $\Psi(1) = \{v_1\}$, 2-class $\Psi(2) = \{v_2, v_3, v_4, v_5, v_6\}$, and 3-class $\Psi(3) = \{v_7, v_8, v_9, v_{10}\}$. In this example, the largest core number of any vertex is $c_{\max} = 3$. The 3-core is formed by the subgraph of G induced by $\Psi(3)$, and the core number of vertex v_7 is $\varphi(v_7) = 3$.

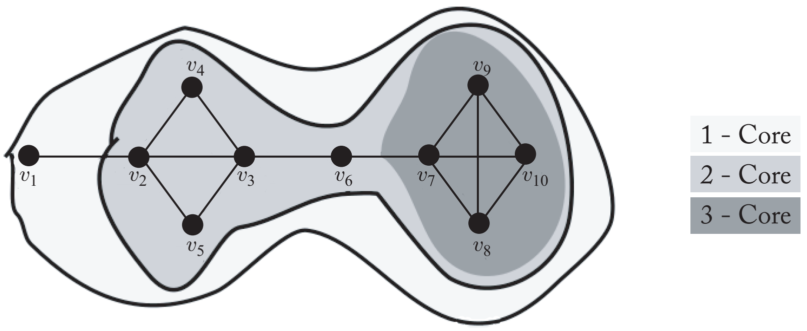

Figure 2.4: An example of k-core in graph G where $1 \leq k \leq 3$. There exists no 4-core in G.

Core decomposition. The problem of core decomposition of a graph G is to find all possible k-cores of G, for $k = 0, 1, \ldots, c_{\max}$, where $c_{\max}$ is the maximum core number of any vertex

Algorithm 2.1 Core Decomposition

Input: $G = (V, E)$
Output: $\varphi(v)$ for each $v \in V$

```
1: sort the vertices in $G$ in ascending order of their degree;
2: while ($G$ is not empty)
3:     let $d$ be the minimum vertex degree in $G$;
4:     while (there exists a vertex $v$ with degree of at most $d$)
5:         $\varphi(v) \leftarrow d$;
6:         remove $v$ and all edges incident to $v$ from $G$;
7:         update the vertex degrees and reorder the remaining vertices in ascending order of their degree;
8: return $\varphi(v)$ for each $v \in V$;
```

in G. Equivalently, the problem is to find all k-classes of G for $k = 0, 1, \ldots, c_{\max}$. Given the core decomposition of G, we can easily obtain the k-core of G for any k, as the subgraph of G induced by the vertex set $\bigcup_{k \le i \le k_{\max}} \Psi(i)$.

Algorithm. For the sake of exposition, we describe a basic core decomposition algorithm [19] that computes the *core number* of each vertex in G. The skeleton of core decomposition is outlined in Algorithm 2.1. It is a simple bottom-up method that computes the k-class from smaller to larger values of k [44].

The algorithm first sorts the vertices in G in ascending order of their degrees. Then the algorithm iteratively removes from G a vertex v with the minimum degree, together with all the edges incident to it, and assigns d, the current minimum degree in G, as its core number $\varphi(v)$. Upon the removal of v, we also update the degrees of the remaining vertices and reorder them according to their new degrees. The algorithm terminates when all vertices are removed from G. In this way, we can compute the core numbers of all vertices in G.

Batagelj and Zaversnik [19] apply bin-sort to order the vertices, leading to an overall running time complexity of $O(m)$ for the algorithm.

Clearly, k-cores constitute dense subgraphs, since each vertex is required to have degree at least k. On the other hand, a k-core may be disconnected, e.g., consider a graph G consisting of two isolated 4-cliques shown in Figure 2.5a. Then the 3-core of the graph G is the union of the two disjoint 4-cliques. However, in the application of community detection and search, it is critical that the community be connected. This motivates the following.

Definition 2.4.3 *[Connected K-Core] Given a graph G and an integer k, a connected k-core is a connected subgraph $H \subseteq G$, such that $\forall v \in V(H)$, $\deg_H(v) \ge k$.*

Figure 2.5b shows a subgraph H of G in Figure 2.5a. H is a connected 3-core, since every vertex has a degree of 3 in H and H is connected.

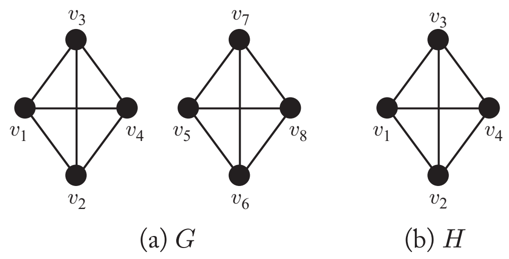

Figure 2.5: An example of classical k-core G and connected k-core H. Here $k = 3$.

2.4.2 k-TRUSS

Recall that the k-core of a graph is the largest subgraph such that each vertex has degree at least k in this subgraph. Similar to the definition of k-core, the k-truss, as a definition of cohesive subgraph of a graph G, requires that each edge be contained in at least $(k-2)$ triangles within this subgraph. Consider the graph G in Figure 2.6; in the subgraph of the whole grey region (i.e., excluding the nodes v_1 and v_6), each edge is contained in at least one triangle. Thus, the subgraph is a 3-truss. It is well known that most of the real-world social networks are triangle-based, in the sense that connections are induced by triangle closures, which always have a high local clustering coefficient.[1] Triangles are also known as the fundamental building blocks of networks [165]. In a social network, a triangle indicates two friends having a common friend, which shows a strong and stable relationship among the three friends. Intuitively, the more common friends two people have, the stronger their relationships. In a k-truss, every pair of friends is "endorsed" by at least $(k-2)$ common friends. Thus, a k-truss with a large value of k signifies strong interconnections between members of the subgraph.

We use the notation $\triangle_{uvw}$ to denote a triangle over u, v, w, i.e., a subgraph over the vertices $\{u, v, w\}$, every pair of which is adjacent. Given an edge $e(u,v) \in E$ in G, the *support* of edge e, is defined as the number of triangles containing e, denoted $sup_G(e) = |\{\triangle_{uvw} : w \in V\}|$. When the context is obvious, we drop the subscript and denote the support as $sup(e)$. Based on the definition of support, we formally define the k-truss as follows.

Definition 2.4.4 *[K-Truss] Given a graph G and an integer k, a k-truss H of G is the largest subgraph such that* $\forall e \in E(H)$, $sup_H(e) \geq (k-2)$.

Note that just like the k-core, the k-truss may be disconnected. For example, consider the graph G in Figure 2.6: the 3-truss consists of two components, viz., the subgraphs of G induced by vertex sets $\{v_2, v_3, v_4, v_5\}$ and $\{v_7, v_8, v_9, v_{10}\}$. Similar to the definition of k-core in Definition 2.4.3, we define a connected k-truss based on the definition of k-truss [50, 165] as follows.

[1]The local clustering coefficient of a vertex v in a graph measures how close its neighborhood is to being a clique: $L_v := \frac{2|\{(v,w)\in E | v,w \in N(v)\}|}{|N(v)|(|N(v)|-1)}$.

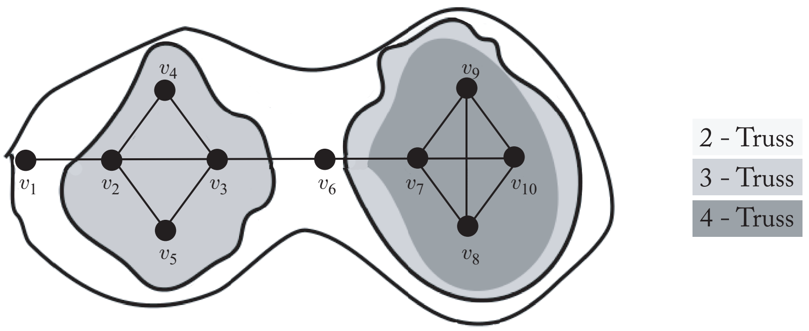

Figure 2.6: An example of k-truss in graph G where $2 \leq k \leq 4$. There exists no 5-truss in G.

Definition 2.4.5 *[Connected K-Truss] Given a graph G and an integer k, a connected k-truss is a connected subgraph $H \subseteq G$, such that $\forall e \in E(H)$, $sup_H(e) \geq (k-2)$.*

Intuitively, a connected k-truss is a connected subgraph such that each edge (u, v) in the subgraph is "endorsed" by $k-2$ common neighbors of u and v [50]. In a connected k-truss graph, each node has degree at least $k-1$ and a connected k-truss is also a connected $(k-1)$-core [19]. Next, we define the *trussness* of a subgraph, an edge, and a vertex.

Definition 2.4.6 *[Trussness] The trussness of a subgraph $H \subseteq G$ is the minimum support of an edge in H plus 2, i.e., $\tau(H) = 2 + \min_{e \in E(H)}\{sup_H(e)\}$. The trussness of an edge $e \in E(G)$ is $\tau(e) = \max_{H \subseteq G \wedge e \in E(H)}\{\tau(H)\}$. The trussness of a vertex $v \in V(G)$ is $\tau(v) = \max_{H \subseteq G \wedge v \in V(H)} \{\tau(H)\}$.*

Consider the graph G in Figure 2.6 as an example. The trussness of the edge $e(v_2, v_3)$ is $\tau(e) = 3$, since the edge $e(v_2, v_3)$ is present in a 3-truss, but not in a 4-truss.

Truss Decomposition. The problem of truss decomposition is to find all possible k-trusses of the graph for $k = 0, 1, \ldots, k_{tmax}$, where k_{tmax} is the maximum truss number of any edge in G. Equivalently, the problem is to find the trussness of all edges of G. For any k, we can easily see that the edge set of the k-truss of G is $\{e : e \in E, k \leq \tau(e) \leq k_{tmax}\}$.

The basic idea of truss decomposition is similar to the core decomposition, that is, to successively remove the edge with the smallest support in each iteration.

We present the truss decomposition algorithm proposed in [165]. Invoked on a graph G, it computes the trussness of each edge. As outlined in Algorithm 2.2, after the initialization, for each k starting from $k = 2$, the algorithm iteratively removes a lowest support edge $e(u, v)$ with $sup(e) \leq k-2$. We assign the trussness of the removed edge as $\tau(e) = k$. Upon removal of e, we decrement the support of all other edges that form a triangle with e, and reorder them according to their new supports. This process continues until all edges with support less than or equal to $(k-2)$ are removed. In this way, we can compute the trussness of all edges in G, and complete the truss decomposition of G.

Algorithm 2.2 Truss Decomposition

Input: $G = (V, E)$
Output: $\tau(e)$ for each $e \in E$

```
 1: $k \leftarrow 2$;
 2: compute $sup(e)$ for each edge $e \in E$;
 3: sort all the edges in ascending order of their support;
 4: while($\exists e$ such that $sup(e) \leq (k-2)$)
 5:     let $e = (u, v)$ be the edge with the lowest support;
 6:     assume, w.l.o.g, $\deg(u) \leq \deg(v)$;
 7:     for each $w \in N(u)$ and $(v, w) \in E$ do
 8:         $sup((u, w)) \leftarrow sup((u, w)) - 1$;
            $sup((v, w)) \leftarrow sup((v, w)) - 1$;
 9:         reorder $(u, w)$ and $(v, w)$ according to their new support;
10:     $\tau(e) \leftarrow k$, remove $e$ from $G$;
11: if(not all edges in $G$ are removed)
12:     $k \leftarrow k + 1$;
13:     goto Step 4;
14: return $\{\tau(e) | e \in E\}$;
```

The algorithms of core decomposition and truss decomposition presented in this and previous sections are both in-memory algorithms, i.e., they assume that the whole graph can fit in main memory. Various studies have been done on core decomposition and truss decomposition in different settings, including in-memory [19, 50, 192], external-memory [44, 165], and MapReduce [51]. In addition, [89, 192] design incremental algorithms for updating a k-truss w.r.t. edge insertions/deletions.

2.5 MORE DENSE SUBGRAPHS

The dense subgraphs are often interpreted as "communities" [40], based on a basic assumption that the connections inside a community are much denser than those between communities. The problem of finding the dense subgraphs of a graph is an important primitive in data analysis, with wide-ranging applications from community mining to spam detection and to the discovery of biological network modules [15]. In Section 3.4, we will introduce one community search model based on the densest subgraph.

In this section, we first give an introduction to densest subgraphs, and then briefly describe other types of dense subgraphs.

2.5.1 DENSEST SUBGRAPHS

Definition 2.5.1 (Classical Edge Density) *Given a graph $G(V, E)$ and a vertex set $S \subseteq V$, the density $\rho(S)$ is defined as* $\rho(S) = \frac{|E(S)|}{|S|} = \sum_{v \in S} \frac{\deg_{G_S}(v)}{2|S|}$.

With these two definitions, the problem of finding the densest subgraph with the maximum edge density can be formulated as follows [15, 106].

Definition 2.5.2 (Densest Subgraph) *Given a graph G, we find a vertex set S^* such that the induced subgraph $G_{S^*} \subseteq G$ has the maximum edge density, i.e., $S^* = \arg\max_{S \subseteq V}\{\rho(S)\}$. Then, the induced subgraph G_{S^*} is the densest subgraph of G.*

It is well known that finding a subgraph with the maximum edge density can be solved optimally using the parametric flow or linear programming relaxation [15]. We denote the problem of finding the densest subgraph as DS-Problem. However, given a positive integer k, finding the maximum density of a subgraph G_S containing at least k vertices is NP-hard [106]. We denote the problem of computing the maximum edge density with at least k vertices as DalK-Problem. Specifically, the problem can be formulated as follows [15, 106].

Definition 2.5.3 (DalK-Problem) *Given a graph G and a positive integer $k > 0$, the DalK-Problem is to find a vertex set $S^*_{\geq k}$ such that $S^*_{\geq k} = \arg\max_{S \subseteq V, |S| \geq k}\{\rho(S)\}$. Then, the induced subgraph $G_{S^*_{\geq k}}$ is the densest subgraph of k vertices.*

For example, consider the graph G in Figure 2.1, the subgraph H_1 is the densest subgraph of 4 vertices.

Approximation. For $\alpha \geq 1$, we say that an algorithm achieves an α-approximation to a maximization (minimization) problem P with objective function f, provided on every input instance with optimal solution $A*$, the algorithm outputs a feasible answer A such that $f(A) \geq f(A^*)/\alpha$ (resp., $f(A) \leq \alpha \times f(A^*)$).

Algorithms. In the literature, various algorithms have been proposed to address the DS-Problem and the DalK-Problem. For the DS-Problem, there exist several exact algorithms [15] to find the densest subgraph of an arbitrary size, including parametric flow [113] and linear programming relaxation [39]. Kortsarz and Peleg [107] and Charikar [39] independently propose 2-approximation algorithms for the DS-Problem. On a high level, the combinatorial approximation algorithm proposed by Charikar [39], iteratively removing the worst node (w.r.t. degree) from the graph in each iteration, is similar to the core decomposition algorithm. For the DalK-Problem, Andersen and Chellapilla [10] apply a similar idea of core decomposition to achieve a 3-approximation to the DalK-Problem. In addition, Khuller and Saha [106] further develop a greedy algorithm to obtain a 2-approximation solution for the DalK-Problem. In the following, we present an algorithm, Algorithm 2.3, called FindLargeDenseSubgraph, which achieves 3-approximation to the DalK-Problem [10].

The algorithm is outlined in Algorithm 2.3. It starts from the original graph G as H_i and proceeds in passes. In each pass, the vertex with the smallest degree is removed. We output one of the intermediate subgraphs $\{H_{|V|}, H_{|V|-1}, \ldots, H_k\}$ with at least k vertices and the largest edge density to form an approximation to the DalK-Problem.

Algorithm 2.3 FindLargeDenseSubgraph

Input: $G = (V, E)$, k
Output: an induced subgraph of G with at least k vertices

```
1: Let $i \leftarrow |V|$ and $H_i \leftarrow G$;
2: $d \leftarrow 0$; $H^* \leftarrow \emptyset$;
3: while ($i \geq k$)
4:     $\rho(H_i) = \frac{E(H_i)}{V(H_i)}$;
5:     if $\rho(H_i) \geq d$ then
6:         $d \leftarrow \rho(H_i)$; $H^* \leftarrow H_i$;
7:     let $v$ be a vertex with the minimum degree in $H_i$;
8:     remove $v$ and all edges incident to $v$ from $H_i$;
9:     $H_{i-1} \leftarrow H_i$;
10:    $i \leftarrow i - 1$;
11: return $H^*$;
```

In terms of time complexity analysis, Algorithm 2.3 runs in time $O(m + n)$ in a graph G with n vertices and m edges. The approximation ratio of Algorithm 2.3 is shown in the following theorem.

Theorem 2.5.1 *FindLargeDenseSubgraph(G, k) of Algorithm 2.3 is a 3-approximation algorithm for the* DalK-Problem *[10].*

Theorem 2.5.1 shows that the approximation ratio of Algorithm 2.3 is 3, which indicates the discovered subgraph H^* by Algorithm 2.3 has at least 1/3 of the density of an optimal solution OPT for the DalK-Problem, i.e., $\rho(H^*) \geq \rho(\text{OPT})/3$.

2.5.2 k-ECC AND k-VCC

In this section, we introduce two kinds of dense subgraphs, namely k-edge-connected component (k-ecc) [38, 197] and k-vertex-connected component (k-vcc) [126, 178], which, respectively, enforce the constraints of edge connectivity and vertex connectivity. We start with the definition of k-edge-connected graphs.

Definition 2.5.4 (k-edge-connected) *A graph $G(V, E)$ is k-edge-connected if G is still connected after removing fewer than k edges from G. In other words, if $G'(V, E \setminus X)$ is connected for any $X \subseteq E$ where $|X| < k$, then G is k-edge-connected.*

Consider the graphs in Figure 2.3. Graph G in Figure 2.3a is 2-edge-connected, as G remains connected if any one edge is removed from G. On the other hand, graph H_1 in Figure 2.3b is 1-edge-connected, since H_1 will become disconnected if the edge (v_2, v_4) is removed from H_1.

Definition 2.5.5 (k-edge-connected component (k-ecc) [38]) *A subgraph $H \subseteq G$ is a k-edge-connected component of graph G, if (i) H is k-edge-connected and (ii) any proper supergraph of H in G is not k-edge-connected.*

Consider the graph H_1 in Figure 2.3b. H_1 is 1-edge-connected. The triangle $\triangle_{v_1 v_2 v_3}$ is a subgraph of H_1, and $\triangle_{v_1 v_2 v_3}$ is 2-edge-connected. Moreover, $\triangle_{v_1 v_2 v_3}$ is a 2-edge-connected component of H_1, since any supergraph of $\triangle_{v_1 v_2 v_3}$ in H_1 is 1-edge-connected.

Using Definition 2.5.5, in the following we show several properties of k-edge-connected components. First, a k-edge-connected component is an induced subgraph of G. Second, a k-edge-connected component is maximal in that adding any vertices and their incident edges into the component would make the new graph no longer k-edge-connected. Third, the k-edge-connected components of a graph are pairwise disjoint. The problem of computing all k-edge-connected components is to decompose a graph G into a set of disjoint k-edge-connected components. To this end, Chang et al. [38] propose a novel graph decomposition paradigm to iteratively decompose a graph G.

Similar to k-edge-connected graphs, k-vertex-connected graphs can be defined as follows [178].

Definition 2.5.6 (k-vertex-connected) *A connected graph G is said to be k-vertex-connected if it has more than k vertices and remains connected whenever fewer than k vertices are removed.*

Notice that by definition, when a vertex is removed, all edges incident on the vertex are removed as well. For example, consider the graph G and H_1, respectively, in Figures 2.3a and 2.3b. According to Definition 2.5.6, G is 2-vertex-connected, since the removal of vertices v_4 and v_5 would make vertex v_6 disconnected from the remaining vertices in G. On the other hand, the removal of any one vertex from graph G leaves G connected. Thus, G is 2-vertex-connected.

The notion of k-vcc can be defined analogously to k-ecc. It is easy to see that the k-vertex-connected components, i.e., maximal k-vertex-connected subgraphs, of a given graph may be overlapping. Wen et al. [178] propose a polynomial-time algorithm to enumerate all k-vertex-connected components of a graph by recursively partitioning the graph into overlapping subgraphs. We note that between k-vcc and k-ecc, k-ecc has been studied more extensively in the literature on dense subgraphs and community search. In our subsequent discussion, we thus focus more on k-ecc. Exploration of community models based on k-vcc may be interesting future work.

2.5.3 OTHER DENSE SUBGRAPHS

Besides the above-mentioned dense subgraphs, there are other types of dense subgraphs. For example, Wang et al. [168] define a dense neighborhood graph based on common neighbors, which is a connected subgraph in which the lower bound on the number of triangles of edges is locally maximized. Their definition renders the problem NP-hard and their proposed solution is approximate [165]. Gibson et al. [74] study dense bipartite subgraphs that are pairs of subsets $A, B \subseteq V$ such that the nodes of A are densely connected with the nodes of B.

2.6 SUMMARY

We close this chapter by comparing the computational efficiency and structural cohesiveness of four representative dense subgraphs: k-clique, k-ecc, k-core, and k-truss. Among them, k-core, k-truss, and k-ecc are often used in community models, since they have decomposition algorithms with a polynomial-time complexity. On the other hand, k-clique is a typical dense subgraph that is NP-hard to compute. Similar dense subgraphs include k-plex, k-clan, k-club, and γ-quasi-k-clique. We compare them in terms of theoretical analysis and experimental evaluation.

Theoretical Efficiency. In terms of theoretical computational efficiency of finding them, the four subgraphs can be ordered as follows: k-core is the most efficient, then k-ecc and k-truss, and k-clique is the most inefficient. This is because the algorithm of core decomposition takes $O(n+m)$ time and $O(n+m)$ space, for a given graph with n nodes and m edges [19]. While the decomposition of k-ecc for a specific k takes $O(hlm)$ time and $O(n+m)$ space, where h and l are often bounded by constant numbers for real-world graphs [38]. As for k-truss decomposition, it has an $O(m^{1.5})$ time complexity [165], which is more expensive than the decomposition of k-core and k-ecc. Finally, due to the NP-hardness of the Maximum Clique problem, the decomposition problem, i.e., finding all k-clique's takes exponential time in the graph size [31].

Theoretical Cohesiveness. In terms of cohesiveness analysis, the structural cohesiveness ranking among these four subgraphs is: k-clique has the most cohesive structure, followed by k-truss, k-ecc, and k-core. Obviously, a k-clique is a complete subgraph of k nodes, which has the smallest diameter of 1 and the highest density of 1. Moreover, k-clique has the following properties: (1) each node has at least $k-1$ neighbors; (2) the graph remains connected after deleting fewer than $(k-2)$ edges; and (3) each pair of nodes has at least $k-2$ neighbors. Thus, k-clique is a subgraph of $(k-1)$-core, $(k-2)$-ECC, and $(k-2)$-truss, indicating its strongest cohesiveness. In addition, k-truss and k-ecc both are subgraphs of $(k-1)$-core, obtained by filtering the disqualified subgraphs based on cohesiveness constraints. Moreover, k-truss has the triangle support constraint for each edge and naturally satisfies the edge connectivity of k-ecc. As a result, the decreasing order of cohesiveness among the four subgraphs is as mentioned above.

Experimental Comparison. We have conducted experiments to compare the four dense subgraph models: k-core, k-ecc, k-truss, and k-clique. We report the running time (in seconds) for comparing their efficiency and various structural metrics for comparing their cohesiveness quality.

First, we compare the decomposition algorithms of k-core [19], k-ecc [38], k-truss [165], and k-clique [56]. We tested five real-world graph datasets: Email-Enron, Google, Livejournal,[2]

[2]Email-Enron, Google, Liverjournal are available at https://snap.stanford.edu/data/index.html.

Wise,[3] and UK-2002.[4] Table 2.1 reports the statistics of these datasets and the running time results of the decomposition algorithms. As expected, k-core is the most efficient method on all datasets; k-ecc is faster than k-truss and k-clique; not surprisingly, k-clique is the worst of all.

Table 2.1: Efficiency comparison for four dense subgraph decomposition algorithms. Here, **K**$= 10^3$ and **M**$= 10^6$.

Datasets	$\|V\|$	$\|E\|$	k-core [19]	k-ecc [38]	k-truss [165]	k-clique [56]
email-Enron	36.7 K	183.8 K	0.2 s	0.8 s	5 s	201 s
Google	876 K	5.1 M	8.9 s	40.8 s	65 s	>24 h
Livejournal	4.8 M	69 M	85 s	854 s	1,726 s	>24 h
Wise	58.6 M	265.1 M	553 s	5,764 s	32,221 s	>24 h
UK-2002	18.6 M	298.1 M	387 s	5,967 s	18,830 s	>24 h

Second, we perform the quality comparison on the Livejournal network. We compare the cohesiveness of the four dense subgraph models with the same parameter $k = 6$. We randomly select 100 query nodes and find the corresponding connected subgraphs of k-clique, k-core, k-truss, and k-ecc that contain the query nodes. We evaluate six structural metrics of those discovered subgraphs: the number of vertices, the number of edges, diameter, density, average degree, and clustering coefficient. Given a subgraph $H \subseteq G$, the average degree is defined as $\frac{2|E(H)|}{|V(H)|}$; and the clustering coefficient is a measure of the degree to which the nodes in a graph tend to cluster together.[5] The quality results are reported in Table 2.2. Obviously, k-clique achieves the best quality with the highest density and clustering coefficient, and has the smallest diameter and graph size. In terms of density, average degree, and clustering coefficient, k-truss performs better than k-core and k-ecc, which is consistent with the theoretical quality analysis.

Note that a k-truss has a good robustness of connectivity. A connected k-truss remains connected when fewer than $(k-1)$ edges are removed from the graph. For example, consider the 4-truss of G in Figure 2.6, the 4-truss remains connected whenever any 2 edges are moved from the 4-truss.

In summary, all discussed dense subgraphs are generally useful for community models, but also have their own disadvantages on either efficiency or quality. In the following chapters, we present different community search problems based on these models.

[3]Wise is available at http://www.wise2012.cs.ucy.ac.cy/challenge.html
[4]UK-2002 is available at http://law.di.unimi.it/datasets.php.
[5]https://en.wikipedia.org/wiki/Clustering_coefficient

Table 2.2: Quality comparison for k-core, k-ecc, k-truss, and k-clique on Livejournal dataset. Here, $k = 6$.

Quality Metrics	k-core	k-ecc	k-truss	k-clique
The number of vertices	1,894,460	1,880,000	1,164,210	6
The number of edges	29,924,900	29,777,900	20,834,200	21
Diameter	17	15	18	1
Density	1.67×10^{-5}	1.69×10^{-5}	3.07×10^{-5}	1.0
Average degree	31.59	31.69	35.79	5
Clustering coefficient	0.303	0.298	0.434	1.0