Discovering Strong Communities with User Engagement and Tie Strength

Fan Zhang[1], Long Yuan[1(✉)], Ying Zhang[2], Lu Qin[2], Xuemin Lin[1], and Alexander Zhou[3]

[1] University of New South Wales, Sydney, Australia
fan.zhang3@unsw.edu.au,{longyuan,lxue}@cse.unsw.edu.au
[2] Centre for AI, University of Technology Sydney, Sydney, Australia
{ying.zhang,lu.qin}@uts.edu.au
[3] University of Queensland, Brisbane, Australia
alexander.zhou@uqconnect.edu.au

Abstract. In this paper, we propose and study a novel cohesive subgraph model, named (k,s)-core, which requires each user to have at least k familiars or friends (not just acquaintances) in the subgraph. The model considers both user engagement and tie strength to discover strong communities. We compare the (k,s)-core model with k-core and k-truss theoretically and experimentally. We propose efficient algorithms to compute the (k,s)-core and decompose the graph by a particular sub-model k-fami. Extensive experiments show (1) our (k,s)-core and k-fami are effective cohesive subgraph models and (2) the (k,s)-core computation and k-fami decomposition are efficient on various real-life social networks.

1 Introduction

Graphs are widely used to represent the abundant interactions in social networks, where each vertex represents a user and each edge represents a relationship between two users. A variety of cohesive subgraph models have been proposed to find social communities, while most of which suffer from computational intractability and other drawbacks. Clique [12] is the most cohesive subgraph model where each vertex is adjacent to every other vertex in the clique. Due to the exponential number of maximal cliques in most social networks and the NP-completeness of the clique decision problem [5], a lot of clique-relaxation models are proposed.

The increasing volume of real-life social networks requires outstanding computation efficiency on cohesive subgraph models, which leads us to the k-core [14] and k-truss [6], the popular and well-studied models with polynomial computation time. The k-core is defined as a maximal subgraph where each vertex has a *degree* of at least k (i.e., at least k neighbors) in the k-core. The k-core is computed by deleting every vertex with degree less than k, which is efficient. However, the simple definition leads to promiscuous subgraphs and thus the k-core is considered as "seedbeds, within which cohesive subsets can precipitate out" [14]. Another concern of the k-core is that the definition on the neighbor

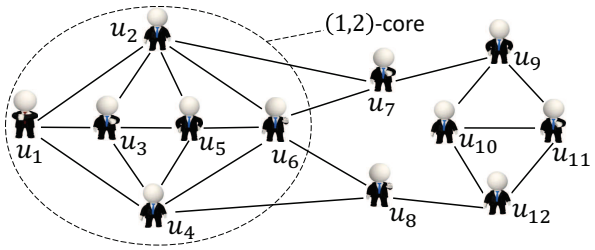


Fig. 1. Motivation Example

number of a vertex treats each incident edge equally (i.e., the strength of every edge is always same). However, the ties (i.e., edges) in social networks have quite different strength, e.g., two users can be acquaintances who only met once, or close friends who meet every day. So the k-truss model is proposed and defined as a maximal subgraph where each edge has a *support* of at least is k (i.e., is contained in at least k triangles) in the k-truss. The support of an edge has been shown to be effective on dynamically estimating the strength of the edge [6, 8, 16]. The k-truss is computed by deleting every edge with support less than k, and then delete isolated vertices. In Figure 1, we show a social network G, consisting of 12 users who form the 3-core. The 4-core of G is empty. The 1-truss of G is G minus the edges $e(u_7, u_9)$ and $e(u_8, u_{12})$. The 2-truss of G is empty.

There are two concerns of the k-truss model. (1) All the edges with support less than k are deleted including those between the vertices in the k-truss, which is not consistent with reality. Like in Figure 1, the edges $e(u_7, u_9)$ and $e(u_8, u_{12})$ are not a part of the 1-truss. The friendship between two users, even if weak, always exists in a community as long as the two users are in the community. Besides, the enforced deletion of weak ties in k-truss makes the tie strength estimation inaccurate, as some triangles are unnecessarily deleted. (2) In social communities, the existence of a relationship between two users is dependent on the existence of the two users (i.e., the communities are vertex-oriented). Since the computation of k-truss is based on the removal of weak ties in the network, which makes the k-truss edge-oriented (i.e., the existence of a vertex in k-truss is decided by the existence of its incident edges). Like in Figure 1, there are many edges with support 2 initially, while the 2-truss computation recursively deletes the edges with support less than 2, which leads to an empty 2-truss.

To address the above concerns, we introduce the (k,s)-core model which is a maximal subgraph where each vertex has an *engagement* of at least k (i.e., at least k strong ties) in the (k,s)-core. Towards tie strength, like in k-truss, an edge is a strong tie if it has a support of at least s (i.e., is contained in at least s triangles) in the (k,s)-core. In Figure 1, the (1,2)-core consists of u_1,u_2,...,u_6 where each user has at least a strong tie. We can see the (1,2)-core is tightly connected which cannot be found by k-core or k-truss for any k. The definition of (k,s)-core ensures there is sufficient number of close friends for each user in the (k,s)-core, which strongly encourages the user to keep engaged in the (k,s)-core. Besides, this definition preserves all the weak ties as long as the incident vertices exist in the (k,s)-core, which is more consistent with reality

and allows for a more accurate estimation of tie strength. The definition of vertex engagement ensures that the (k,s)-core is vertex-oriented and possesses more potential on computation efficiency than k-truss. An efficient algorithm is proposed to compute the (k,s)-core.

The two parameters in (k,s)-core enable the model to have high flexibility in regards to adjusting different requirements for user engagement and tie strength. However, this makes the decomposition more complex as it needs to compute all the (k,s)-cores for any given k and s. To make the decomposition more affordable, we introduce a representative sub-model k-fami which is a $(k, k-1)$-core. We propose an efficient algorithm to decompose a graph into hierarchical structures by the k-fami. Extensive experiments show our (k,s)-core computation and k-fami decomposition are more efficient than k-truss computation and its decomposition, respectively. With the definitions based on characteristics of social communities, our (k,s)-core and k-fami produce more convincing cohesive subgraphs for finding strong communities.

2 Problem Definition

In this section, we give some notations and formally define the cohesive subgraph models including our novel (k,s)-core. The notations are summarized in Table 1.

We consider an unweighted and undirected graph $G = (V, E)$, where V (resp. E) represents the set of vertices (resp. edges) in G. We denote $n = |V|$, $m = |E|$ and assume $m > n$. $N(u, G)$ is the set of adjacent vertices of u in G. We say a vertex u is incident to an edge e, or e is incident to u, if u is one of the endpoints of e. Let S denote a subgraph of G. We use $deg(u, S)$, the *degree* of u in S, to represent the number of adjacent vertices of u in S. When the context is clear, we omit the the input graph in notations, such as $deg(e)$ for $deg(e, G)$.

Definition 1. ***k-core****. Given a graph G, a subgraph S is the k-core of G, denoted by $C_k(G)$, if (i) S satisfies the degree constraint, i.e., $deg(u, S) \geq k$ for every $u \in S$; and (ii) S is maximal, i.e., any subgraph $S' \supset S$ is not a k-core.*

User Engagement. For each user (vertex), the k-core model uses the number of acquaintances (neighbors) in the k-core to measure the engagement of this user. In our (k,s)-core model, we consider the number of friends or familiars to better represent the engagement of a user.

Towards the k-core model, one straightforward concern is that the relationships (edges, i.e., ties) between users are enforced to have equal strength, which is not consistent with reality. Consequently, the model of k-truss is proposed where each tie has different strength. We define a triangle as a cycle of length 3 in the graph. A containing-e-triangle is a triangle which contains e. The *support* of e in S, i.e., $sup(e, S)$, represents the number of containing-e-triangles in S.

Definition 2. ***k-truss****. Given a graph G, a subgraph S is the k-truss of G, denoted by $T_k(G)$, if (i) $sup(e, S) \geq k$ for every edge $e \in S$; (ii) S is maximal, i.e., any subgraph $S' \supset S$ is not a k-truss; and (iii) S is non-trivial, i.e., no isolated vertex in S.*

Table 1. Summary of Notations

Notation	Definition
G	an unweighted and undirected graph
u, v	a vertex in the graph
e; $e(u, v)$	an edge in the graph; the edge with u and v as endpoints
n, m	the number of vertices and edges in G, respectively
$N(u, G)$	the set of adjacent vertices of u in G
$deg(u, G)$	the number of adjacent vertices of u in G
$sup(e, G)$	the number of triangles each containing e in G
k, s	the thresholds
$eng(u, G)$	the number of edges where each edge e has $sup(e, G) \geq s$ and e is incident to u in G
$C_k(G)$; $T_k(G)$	the k-core of G; the k-truss of G
$C_{k,s}(G)$; $F_k(G)$	the (k,s)-core of G; the k-fami of G
$fn(u)$	fami number of the vertex u
$E(u, G)$	the edge set where each edge is incident to u and is in G

Tie Strength. For each tie (edge), the k-truss model uses the number of triangles containing it (common neighbors of two endpoints) in the k-truss to estimate the strength of this tie. All weak ties are deleted in k-truss. In our (k,s)-core model, we preserve the weak ties between community members to better estimate the strength of a tie.

In real-life social network, the relationship between two users is concurrent with the existence of the users. This means the weak ties between the users in k-truss should not be deleted. Furthermore, the enforced removing of the weak ties leads to the incompletion of some triangles and thus the inaccuracy on the estimation of tie strength. To overcome these concerns, we firstly define strong tie and strong engagement as the following.

Definition 3. ***strong tie****. Given a graph G and an integer s, an edge e is called a strong tie in G if $sup(e, G) \geq s$; or a weak tie if $sup(e, G) < s$.*

We use $eng(u, S)$, the *engagement* of u in S, to represent the number of strong ties where each edge e has $sup(e, S) \geq s$ and e is incident to u.

Definition 4. ***strong engagement****. Given a graph G and an integer k, a vertex u is strongly engaged in G if u is incident to at least k strong ties in G, i.e., $eng(u, G) \geq k$; or weakly engaged if $eng(u, G) < k$.*

If a user has at least k close friends or familiars in a community, he/she is strongly encouraged to stay engaged in the community, which naturally leads us to the following definition for modeling strong communities.

Definition 5. ***(k,s)-core****. Given a graph G, a subgraph S is the (k,s)-core of G, denoted by $C_{k,s}(G)$, if (i) every vertex in S is strongly engaged in S, i.e., $eng(u, S) \geq k$ for each $u \in S$; and (ii) S is maximal, i.e., any subgraph $S' \supset S$ is not a (k,s)-core.*

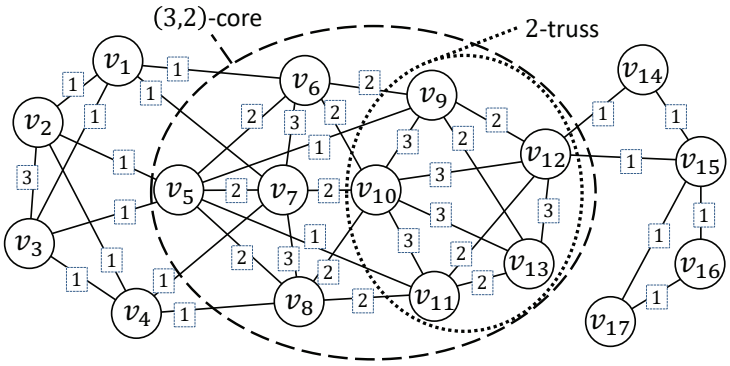


Fig. 2. Running Example

The (k,s)-core model ensures that each inside user has at least k strong ties in the (k,s)-core, and each tie is preserved if the two corresponding users (endpoints) exist in the (k,s)-core. Consequently, the (k,s)-core overcomes the above mentioned concerns in k-core and k-truss, and can be more consistent with real-life scenarios. Note that we have $C_{k,0}(G) = C_k(G)$, $C_{k,s}(G) \subseteq C_k(G)$ and $T_k(G) \subseteq C_{k+1,k}(G)$ according to above definitions.

Example 1. In Figure 2, the social network G consists of 17 vertices where each edge is labeled by its support in G. The G itself is a 2-core, 1-truss and (2,1)-core, respectively. The $C_{3,2}(G)$ is induced by $\{v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}\}$ which is a tightly connected vertex set. Note that although there are some edges between $\{v_1, v_2, v_3, v_4\}$ and $C_{3,2}(G)$, each of the edges has a support of only 1, which means the connection is weak. The $C_{3,2}(G)$ cannot be found by k-core or k-truss because (i) the $C_3(G) = C_4(G)$ is induced by $G \setminus \{v_{14}, v_{15}, v_{16}, v_{17}\}$; (ii) the $C_5(G) = \emptyset$; (iii) the $T_2(G)$ is induced by v_9, v_{10},..., v_{13}; and (iv) the $T_3(G) = \emptyset$. Note that in the computation of $T_2(G)$, the edge supports decrease accordingly when removing each edge with a support of less than 2.

In real-life applications, the value of k (resp. s) can be determined by users based on their requirement for engagement level (resp. tie strength), or learned according to ground-truth communities. The parameters k and s provide more flexibility on adjusting the resulting communities from (k,s)-core.

3 (k, s)-Core Computation

In this section, we introduce an efficient algorithm for finding the (k,s)-core. The following theorem shows we can correctly compute the (k,s)-core on a small k'-core.

Theorem 1. *When $k > 0$, the (k,s)-core of G is a subgraph of k'-core of G (i.e., $C_{k,s}(G) \subseteq C_{k'}(G)$) where $k' = max(k, s+1)$.*

Proof. The (k,s)-core of G, $C_{k,s}(G)$, is a subgraph of the k-core of G because each vertex in $C_{k,s}(G)$ has at least k neighbors in $C_{k,s}(G)$. When $k > 0$, each

Algorithm 1: **ComputeCore**(G, k)

```
   Input  : G : a social network, k : degree constraint
   Output : C_k(G)
1  while exists u ∈ G with deg(u, G) < k do
2      G := G \ {u ∪ E(u, G)};
3  return G
```

Algorithm 2: **ComputeKSCore**(G, k, s)

```
   Input  : G : a social network, k : engagement constraint, s : tie strength
            constraint
   Output : C_{k,s}(G)
1  k' := max(k, s + 1); G := ComputeCore(G, k');
2  s(e) := sup(e, G) for each e ∈ G; d(u) := eng(u, G) for each u ∈ G;
3  while exists u ∈ G with d(u, G) < k do
4      for each v ∈ N(u) and d(v, G) ≥ k do
5          G := G \ e(u, v);
6          if s(e(u, v)) ≥ s then
7              d(v) := d(v) − 1;
8          for each w ∈ N(u) ∩ N(v) and d(w) ≥ k and s(e(v, w)) ≥ s do
9              s(e(v, w)) := s(e(v, w)) − 1;
10             if s(e(v, w)) = s − 1 then
11                 d(w) := d(w) − 1;
12     G := G \ {u ∪ E(u, G)};
13 return G
```

vertex in $C_{k,s}(G)$ is incident to at least a strong tie which is contained in at least s triangles in $C_{k,s}(G)$, which means the vertex has a degree of at least $s+1$. Since $k' = max(k, s+1)$, every vertex in $C_{k,s}(G)$ has a degree of at least k', i.e., the (k,s)-core is a subgraph of the k'-core.

Theorem 1 allows us to compute the k'-core first as a base for (k,s)-core computation. The algorithm for computing k-core is shown in Algorithm 1 with a time complexity of $O(m+n)$.

Algorithm 2 gives the algorithm for computing the (k,s)-core. In Line 1, we compute the k'-core. Then we do triangle counting on the k'-core to generalize the support and engagement values in Line 2. For each vertex with insufficient engagement (Line 3), we delete the vertex and its incident edges (Line 12 and 5) where the support and engagement values are updated accordingly. Specifically, in Line 4, we can delete the incident edge of u one by one. Then we update the affected engagement value (Line 7, and 11) and the edge support in affected triangles (Line 8 and 9). Note that we do not need to update the vertex engagements which are already less than k and the edge supports which are already less than s.

Example 2. In Figure 2, the social network G consists of 17 vertices where each edge is labeled by its support in G. In the computation of (3,2)-core, we firstly compute the 3-core of G, which deletes $\{v_{14}, v_{15}, v_{16}, v_{17}\}$ from G. Then we compute the support for each edge in current G and count the engagement for each vertex in G. We push v_1, v_2, v_3 and v_4 in queue for deletion since their engagement is less than 3. Note that when we delete a vertex and its incident edges, we do not need to update the corresponding supports and engagements which are already insufficient. Like when v_1 is deleted, we do not need to update the engagement of v_2 and the support of $e(v_2, v_3)$. Once the engagement of a vertex drops from 3 to 2, it is pushed into the queue. We get the (3,2)-core after deleting every vertex and its incident edges in the queue.

Complexity. The most time-consuming steps are computing $sup(e, G)$ for each e (Line 2) and its update (Line 8) which both take $O(m^{1.5})$ [18]. The vertex deletion and edge deletion take $O(n)$ and $O(m)$ respectively. So the time complexity is $O(m^{1.5})$. We need $O(n)$ space to store engagement set and $O(m)$ space to store the neighbor set and edge support set in G. So the space complexity is $O(m)$.

Correctness. The correctness is straightforward if no (1) $d(u, G) < k$ in Line 4 and no (2) $d(w) \geq k$ and $s(e(v, w)) \geq s$ in Line 8. The reason for (1) and $d(w) \geq k$ in (2) is that all the vertices with already less than k engagements will be deleted with their incident edges, the update for their engagements and edge supports is not necessary. The reason for $s(e(v, w)) \geq s$ in (2) is that all the edges with less than s supports are already weak ties which cannot be affected by the deletion of other edges. Note that the existence of edges is concurrent with the existence of their incident vertices.

4 Fami Decomposition

In this section, we propose the model of k-fami and its decomposition algorithm. Firstly, we introduce the following theorem which reveals the hierarchical structure from the (k,s)-core.

Theorem 2. *Given k and s, the (k,s)-core of G is a subgraph of (k', s')-core of G (i.e., $C_{k,s}(G) \subseteq C_{k',s'}(G)$) if $k \geq k'$ and $s \geq s'$.*

Proof. When $s \geq s'$, we have $C_{k,s}(G) \subseteq C_{k,s'}(G)$ because (i) every strong tie e in $C_{k,s}(G)$ is also a strong tie on $sup(e) \geq s'$; and (ii) each vertex in $C_{k,s}(G)$ has at least k strong ties to fulfill the requirement for existing in $C_{k,s'}(G)$. When $k \geq k'$, we have $C_{k,s'}(G) \subseteq C_{k',s'}(G)$ because $eng(u) \geq k'$ for each $u \in C_{k,s'}(G)$. Consequently, $C_{k,s}(G) \subseteq C_{k',s'}(G)$ if $k \geq k'$ and $s \geq s'$.

For a given k and s, Theorem 2 shows that we can find a (k', s')-core which contains the (k,s)-core ($k' \leq k$ and $s' \leq s$) and a (k'', s'')-core which is contained in the (k,s)-core ($k'' \geq k$ and $s'' \geq s$). It motivates us to introduce a particular sub-model k-fami as a representation for the (k,s)-core to show the hierarchical

Algorithm 3: **FamiDecomp**(G)

```
   Input  : G : a social network
   Output : fn(u) for every u ∈ G
 1 k := 1; G' := G;
 2 d(u) := deg(u, G) for each u ∈ G;
 3 s(e) := sup(e, G) for each e ∈ G;
 4 order the vertices in G with increasing order of d(u) for each u;
 5 order the edges in G with increasing order of s(e) for each e;
 6 while G is not empty do
 7     while exists u ∈ G with d(u) < k in the order do
 8         fn(u) := k − 1;
 9         for each vertex v ∈ N(u) and d(v) ≥ k do
10             G := G \ e(u, v);
11             if s(e(u, v)) ≥ k − 1 then
12                 d(v) := d(v) − 1 and reorder the vertices in G;
13             for each w ∈ N(u) ∩ N(v) and d(w) ≥ k and s(e(v, w)) ≥ k − 1 do
14                 s(e(v, w)) := s(e(v, w)) − 1 and reorder the edges in G;
15                 if s(e(v, w)) = k − 2 then
16                     d(w) := d(w) − 1 and reorder the vertices in G;
17         G := G \ {u ∪ E(u, G)};
18     for each edge e(u, v) ∈ G with s(e(u, v)) = k − 1 in the order do
19         d(u) := d(u) − 1 and reorder the vertices in G;
20         d(v) := d(v) − 1 and reorder the vertices in G;
21     k := k + 1;
22 return fn(u) for every u ∈ G'
```

structure of a graph. The computation of all the (k,s)-core for any k and s is time-consuming due to the large number of combinations of k and s. Our k-fami decomposition can produce the hierarchical structure of a graph in $O(m^{1.5})$, which runs faster than k-truss decomposition in our experiments.

Definition 6. ***k*-*fami***. *Given a graph G, a subgraph S is the k-fami of G (k-familiar in full), denoted by $F_k(G)$, if S is a $(k, k-1)$-core, i.e., $S = C_{k,k-1}(G)$.*

With the k-fami model, every vertex in the graph can have a fami number.

Definition 7. ***fami number***. *Given a graph G, the fami number of a vertex u is k^*, denoted by $fn(u, G)$, if (i) there is a k^*-fami which contains u, i.e., $u \in F_{k^*}(G)$; and (ii) there is no other $k' > k^*$ such that $u \in F_{k'}(G)$.*

Fami decomposition is to compute the fami number for every vertex in the graph. Algorithm 3 presents an algorithm for fami decomposition. Line 1 to 3 initialize the arguments including $d(u)$ (engagement) for each vertex u and $s(e)$ (support) for each edge e. Line 4 orders the vertices with increasing order of their engagements. Note that the order can be updated in $O(1)$ time in Line 12, 16, 19 and 20 by using bin sort. Line 5 orders the edges with increasing order of supports. The order can also be updated in $O(1)$ time in Line 14 by using bin sort.

A good implementation can be found in [10]. Then we compute the k-fami from $k = 1$ which computes the $(k-1)$-fami. The algorithm terminates and produces all fami numbers when G becomes empty in Line 6. In Line 7 and 8, the fami number of every vertex u with less than k engagement is recorded as $k-1$. For each neighbor of u with at least k engagement, we delete the edge $e(u, v)$ and update its engagement if necessary (Line 9 to 12). We also update the supports and engagements affected by the deletion of $e(u, v)$ (Line 13 to 16). Note that we do not need to update the engagements of vertices with less than k engagements and the supports of their incident edges, because these vertices are already in the waiting list to be deleted (Line 7). After deleting all vertices with less than k engagements, we increase k by 1 in Line 21 and update the engagements since some edges becomes weak ties with the increase of k in 18 to 20.

Example 3. In Figure 2, the social network G consists of 17 vertices where each edge is labeled by its support in G. In the k-fami decomposition, we firstly compute the support for each edge in G and count the engagement for each vertex in G. Then the edges are ordered with increasing supports and the vertices are ordered with increasing engagements. Note that the orders are implemented in integer buckets with $O(1)$ update time as in [10]. Then we compute the k-fami from $k = 1$ to k_{max}. When $k = 1$, there is no $u \in G$ with $d(u) < k$. Then we lift the strong tie threshold by 1, which decrease the $d(u)$ and $d(v)$ by 1 for each $e(u, v)$ with support $k-1$, and reorder the vertices. Then we lift k by 1 and find the vertices with $d(u) < k$ in the order. The computation of k-fami is the same as in Algorithm 2 except that $s = k + 1$ and the reorder for vertices and edges. When $k = 2$, there is still no $u \in G$ with $d(u) < k$. When $k = 3$, 8 vertices with $d(u) < k$ are deleted (v_1,...,v_4,v_{14},...,v_{17}) and marked with fami number 2. Recursively, the algorithm terminates when all the vertices are deleted and marked. We thus have the k-fami for k from 1 to 3.

Complexity. In Algorithm 3, the most time-consuming steps are computing $sup(e)$ for every $e \in G$ (Line 3) and updating the edge supports (Line 13 and 14), which takes $O(m^{1.5})$ time [18]. The removal of all vertices and edges takes $O(m)$ time. The orders, engagement reorders and support reorders take $O(m)$ time. So the time complexity of Algorithm 3 is $O(m^{1.5})$. Towards the space complexity, the neighbor set for every vertex, the edge support set and the support order dominate the complexity, where each takes $O(m)$ space. So the space complexity of Algorithm 3 is $O(m)$.

Correctness. We show that for each k in Algorithm 3, it computes a correct $(k-1)$-fami. When $k = 1$, the isolated vertices are removed from G (Line 17) with fami number 0 (Line 8). When $k = 2$, the engagement numbers are correctly updated (Line 18 to 20). Then every vertex with less than k engagement is deleted, where the incident edges are deleted one by one (Line 10 and 17). Note that the engagements of vertices with less than k engagement need not to be updated according to edge deletions because these vertices are already in waiting list for deletion. The supports of their incident edges also need not to be updated. Besides, the supports less than $k-1$ need not to be updated

Table 2. Statistics of Datasets

Dataset	Nodes	Edges	d_{avg}	k_{max}^{core}	k_{max}^{truss}	k_{max}^{fami}	$\|\triangle\|$
Facebook	4,039	88,234	43.7	115	95	102	1,612,010
Brightkite	58,228	194,090	6.7	52	40	43	449,717
Gowalla	196,591	456,830	4.7	43	21	25	1,061,143
YouTube	1,134,890	2,987,624	5.3	51	17	24	3,056,386
DBLP	1,566,919	6,461,300	8.3	118	117	118	15,389,320
Pokec	1,632,803	8,320,605	10.2	27	18	19	6,971,538
LiveJournal	3,997,962	34,681,189	17.4	360	350	353	177,820,130
Orkut	3,072,441	117,185,083	76.3	253	76	83	627,584,181

Table 3. Summary of Algorithms

Algorithm	Description
k-core	computing the k-core [3], i.e., Algorithm 1
k-truss	computing the k-truss [6]
k-fami	computing the k-fami, i.e., Algorithm 2
ks-core	computing the (k,s)-core, i.e., Algorithm 2
coreDecomp	core decomposition in [10], i.e., computing the largest k for every vertex $u \in G$ such that the k-core contains u
trussDecomp	truss decomposition in [18], i.e., computing the largest k for every vertex $u \in G$ such that the k-truss contains u
famiDecomp	fami decomposition, i.e., Algorithm 3

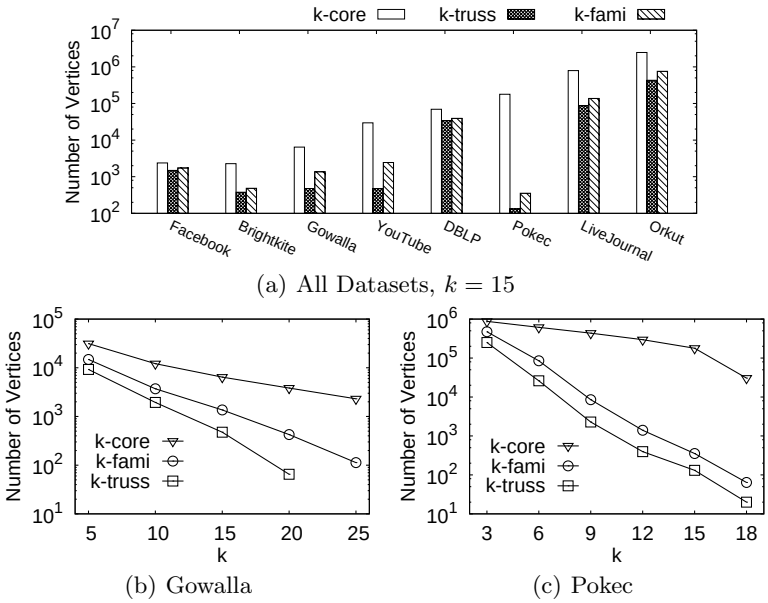
because they are already weak ties and cannot affect the vertex engagements. At Line 21, all vertices with less than k engagement are deleted with correct updates of all supports and engagements. Current G is a $(k-1)$-fami. For $k > 2$, the correctness can be ensured by recursion and Theorem 2.

5 Experimental Evaluation

5.1 Experimental Setting

Datasets. Eight real-life networks were deployed in our experiments and we assume all vertices in each network are initially engaged. The original data of DBLP was downloaded from http://dblp.uni-trier.de/ and the others from http://snap.stanford.edu/. In DBLP, we consider each author as a vertex and there is an edge for a pair of authors if they have at least one co-authored paper. There are existing vertices and edges in other datasets. Table 2 shows the statistics of the 8 datasets, listed in increasing order of their edge numbers.

Algorithms. To the best of our knowledge, no existing work investigates the (k,s)-core and k-fami. We tested 4 algorithms (k-core, k-truss, k-fami and ks-core) to produce and compare different resulting subgraphs. We also implemented and evaluated the decomposition algorithms including core decomposition (coreDecomp), truss decomposition(trussDecomp) and our fami decomposition (famiDecomp). Table 3 shows the summary of the algorithms.



(a) All Datasets, $k = 15$

(b) Gowalla

(c) Pokec

Fig. 3. Vertex Number in k-core, k-truss and k-fami

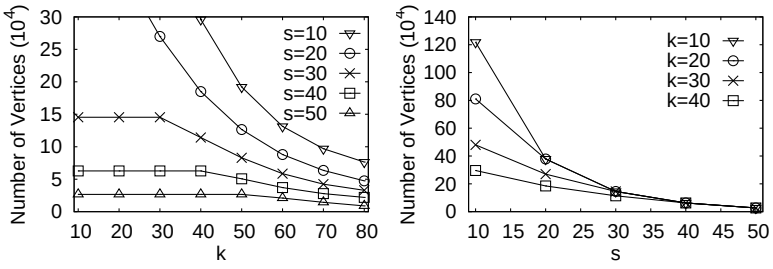
Parameters. We conducted experiments under different settings by varying the engagement constraint k from 3 to 80 and the support constraint s from 10 to 50. We also report the result of 3 graph decompositions.

All programs were implemented in standard C++ and compiled with G++ in Linux. All experiments were performed on a machine with Intel Xeon 2.8GHz CPU and Redhat Linux System.

5.2 Effectiveness

Statistics. We report the maximum core number (k_{max}^{core}), truss number (k_{max}^{truss}) and fami number (k_{max}^{fami}) on each dataset in Table 2. The k_{max}^{fami} is usually between the values of k_{max}^{core} and k_{max}^{fami}, which shows our k-fami model captures unique hierarchical structures of the graphs. We show the number of vertices in k-core, k-truss and k-fami in Figure 3. When $k = 15$, Figure 3 (a) shows the size of k-fami is always between k-core and k-truss where the difference varies on all datasets due to the different natures of the datasets. Figure 3 (b) and (c) show the decrease of the size in 3 models with the growth of k. When $k = 25$, the k-truss in Gowalla is empty. The margin between the sizes of k-fami and the other 2 models varies with different k.

In Figure 4, we report the size of (k,s)-core with different k and s. Figure 4 (a) shows the trend of the (k,s)-core size when we fix s and vary k. Figure 4 (b) shows the trend of the (k,s)-core size when we fix k and vary s. In both Figure 4 (a) and (b), for a given s, We can see that the (k,s)-core sizes are almost the same when $k \leq s$ because they all belong to $(s + 1)$-core according



(a) Orkut (b) Orkut

Fig. 4. Vertex Number in (k,s)-core with Different k and s

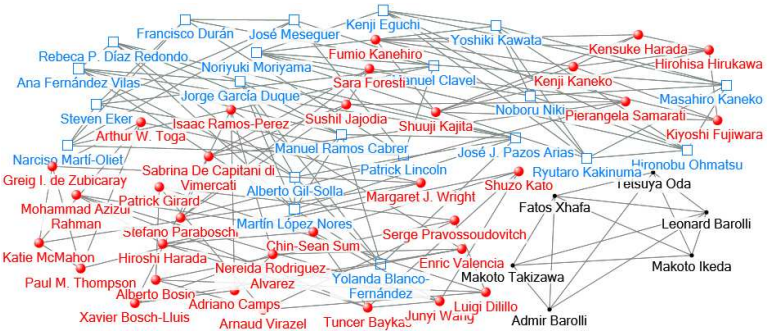


Fig. 5. Case Study of k-core, k-truss and k-fami on DBLP

to Theorem 1. When $k > s$, the (k,s)-core becomes smaller with the increase of k. Figure 4 reveals the hierarchical structure of a graph with the changing of k and s in the (k,s)-core.

Case Study on DBLP. Figure 5 depicts the k-core, k-truss and k-fami on the `DBLP-30` dataset with $k = 15$. In `DBLP-30`, to make this case study visible, each edge between two authors represents that there are at least 30 co-authored papers between the two authors. The whole graph in Figure 5 is the k-core of `DBLP-30`. The (k,s)-core excludes the sparse group at the bottom right corner with 5 authors. The k-truss is formed by all the square vertices (in blue) which excludes all the sphere vertices (in red) and their incident edges from (k,s)-core. We can see all the square and sphere vertices connect tightly, which shows the advantages of our (k,s)-core model. Specifically, the (k,s)-core is superior than the other 2 models in the sense that (1) the k-core is relatively large since it tolerates some vertices with low engagement and (2) the k-truss enforcedly excludes all the weak ties which makes the tie strength estimation (number of triangles) inaccurate and the non-concurrence of the vertices and its incident edges in resulting communities.

5.3 Efficiency

Decompositions. In Figure 6, we report the decomposition time for k-core, k-truss and k-fami. The `coreDecomp` and `trussDecomp` are the state-of-the-art al-

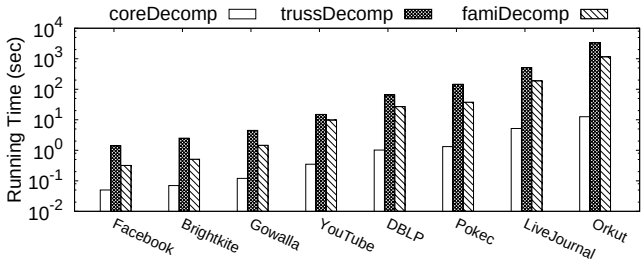
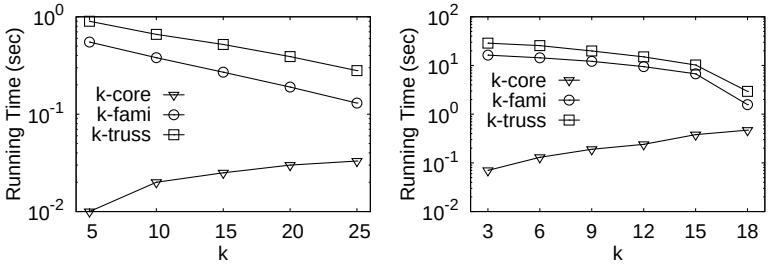


Fig. 6. Running Time for Graph Decompositions



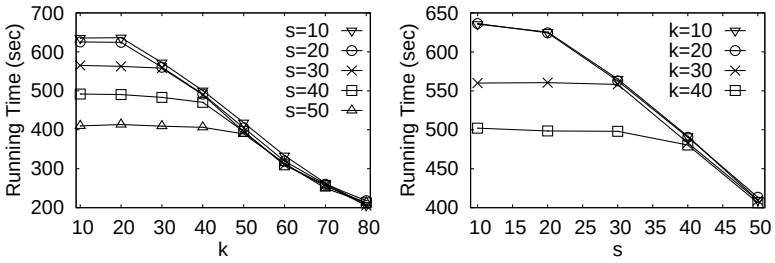
(a) Gowalla (b) Pokec

Fig. 7. Running Time of k-core, k-truss and k-fami Computation

gorithms for in-memory core decomposition and truss decomposition, respectively. Figure 6 shows that `coreDecomp` is faster than `trussDecomp` and `famiDecomp` because it does not need triangle listing and support updates on the graph. However, `coreDecomp` treats each edge equally and ignores the difference in tie strength. Our `famiDecomp` algorithm outperforms `trussDecomp` in running time by up to 2 times, because `famiDecomp` is a vertex-oriented algorithm where the existence of edges depends on the incident vertices, while `trussDecomp` is an edge-oriented algorithm. Besides, our `famiDecomp` can avoid unnecessary updating of some supports and engagements as Algorithm 3 shows.

Effect of k and s. We show the effect of k in k-core, k-truss and k-fami computation in Figure 7. As discussed above, `k-core` is faster than `k-truss` or `k-fami` but it ignores the strength of ties in discovering cohesive subgraphs. The running time of `k-truss` and `k-fami` becomes smaller as k increases because both of them compute a k'-core first which reduces the candidate set for their computation. So performance of 3 algorithms tends to be closer when k becomes larger. In Figure 8, we show the running time of (k,s)-core on different k and s. The runtime of `ks-core` becomes smaller when $max(k, s+1)$ becomes larger, because `ks-core` computes a $max(k, s+1)$-core first to reduce the candidate set. For the same reason, when we fix s and $k \leq s$, the runtime of `ks-core` does not change much. It also explains the consistent runtime of `ks-core` when we fix k and $s < k$.

Different Datasets. Figure 9 reports the running time of k-core, k-truss and k-fami on all datasets with $k = 15$. `k-core` still outperforms `k-truss` and `k-fami`



(a) Orkut (b) Orkut

Fig. 8. Running Time of (k,s)-core with Different k and s

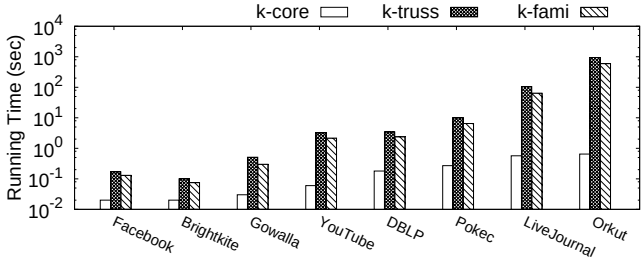


Fig. 9. Running Time of k-core, k-truss and k-fami Computation

by ignoring the tie strength in computation. Our `k-fami` is faster than `k-truss` on all datasets because `k-fami` is a vertex-oriented algorithm and avoids updating unnecessary support and engagement values in Algorithm 2. It further confirms the effectiveness of our (k,s)-core model in producing cohesive subgraphs with the consideration of tie strength.

6 Related Work

There are various cohesive subgraph models to accommodate different scenarios in the literature. Clique [12] is an extremely cohesive subgraph where every vertex is adjacent to every other vertex in the clique. Because the definition of clique is usually too restrictive, some clique relaxation models have been proposed, such as k-plex [15], k-core [14] and k-truss [6], and so on. Among these cohesive subgraph models, k-core and k-truss are the widely studied models with polynomial computation time.

Seidman [14] proposes the k-core where each vertex has at least k neighbors in the k-core. The k-core has a wide spectrum of applications such as social contagion [17], community detection [21], user engagement [20] and so on. Batagelj and Zaversnik [2] present an algorithm for core decomposition of a graph with time complexity of $O(m+n)$. Zhang *et* al. [22] propose a fast order-based algorithm to maintain k-core in dynamic graphs. Bhawalkar *et* al. [4] propose the problem of anchored k-core to prevent network unraveling. Zhang *et* al. [19] present an efficient algorithm to solve the anchored k-core problem. However, the k-core uses vertex degree to determine the user engagement which treats

each edge equally. In real-life social networks, the strength of user relationships (edges) varies a lot and cannot be always identical [7].

Further considering the strength of ties, Cohen [6] proposes the model of k-truss where every edge exists in at least k triangles in the k-truss with its decomposition algorithm in $O(\sum_{v \in V(G)}(\deg(v)^2))$ time. Rotabi *et* al. [13] show that most strong tie detection methods are based on structural information, especially on triangles. Wang and Cheng [18] reduce the time complexity of truss decomposition to $O(m^{1.5})$ and study the I/O efficient truss decomposition. Shao *et* al. [16] study the k-truss detection problem on distributed systems and propose an efficient parallel algorithm. Zhao and Tung [23] use the k-truss to capture the cohesion in social interactions and propose a visualization system based on k-truss. Huang *et* al. [8] study the k-truss based community model, which further requires edge connectivity inside the community. Akbas and Zhao [1] propose a truss-equivalence based index to speed up the search of the truss based community. Huang and Lakshmanan [9] study the attributed k-truss community search where the largest attribute relevance score is satisfied. However, k-truss deletes all the weak ties even if the corresponding users exists in the k-truss, which makes the tie strength estimation inaccurate and that some users are excluded from k-truss unreasonably. Besides, k-truss is edge-oriented while social communities are user(vertex)-oriented.

To the best of our knowledge, we for the first time propose the novel (k,s)-core to overcome above concerns in k-core and k-truss. Lee *et* al. [11] propose the (k, d)-core where each vertex has at least k neighbors in the subgraph and each edge is contained in at least d triangles in the subgraph. The (k, d)-core is essentially a subgraph of d-truss with additional requirement for vertex degree of at least k, which can be regarded as a strengthened d-truss. Thus, the model still has the same concerns as in k-truss and is different than our (k,s)-core model.

7 Conclusion

In this paper, we propose a novel cohesive subgraph, (k,s)-core, which requires each user to have at least k familiars or friends in the subgraph. The (k,s)-core addresses the concerns in k-core and k-truss including (1) k-core enforces the strength of each tie to be equal; (2) k-truss deletes all the weak ties; and (3) k-truss is edge-oriented while social communities are user(vertex)-oriented. We propose an efficient algorithm to compute the (k,s)-core. A particular k-fami is introduced to efficiently decompose a graph. Extensive experiments validate the effectiveness of our models and the efficiency of our algorithms.

Acknowledgments

Fan Zhang and Long Yuan are supported by Huawei YBN2017100007. Ying Zhang is supported by ARC FT170100128 and DP180103096. Lu Qin is supported by ARC DP160101513. Xuemin Lin is supported by NSFC 61672235, ARC DP170101628, DP180103096 and Huawei YBN2017100007.

References

1. E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.
2. V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.
3. V. Batagelj and M. Zaversnik. Fast algorithms for determining (generalized) core groups in social networks. *Adv. Data Analysis and Classification*, 5(2):129–145, 2011.
4. K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.
5. C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.
6. J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, page 16, 2008.
7. M. S. Granovetter. The strength of weak ties. *American journal of sociology*, 78(6):1360–1380, 1973.
8. X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.
9. X. Huang and L. V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.
10. W. Khaouid, M. Barsky, S. Venkatesh, and A. Thomo. K-core decomposition of large networks on a single PC. *PVLDB*, 9(1):13–23, 2015.
11. P. Lee, L. V. S. Lakshmanan, and E. E. Milios. CAST: A context-aware story-teller for streaming social content. In *CIKM*, pages 789–798, 2014.
12. R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.
13. R. Rotabi, K. Kamath, J. M. Kleinberg, and A. Sharma. Detecting strong ties using network motifs. In *WWW*, pages 983–992, 2017.
14. S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.
15. S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.
16. Y. Shao, L. Chen, and B. Cui. Efficient cohesive subgraphs detection in parallel. In *SIGMOD*, pages 613–624, 2014.
17. J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 109(16):5962–5966, 2012.
18. J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.
19. F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. OLAK: an efficient algorithm to prevent unraveling in social networks. *PVLDB*, 10(6):649–660, 2017.
20. F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *AAAI*, pages 245–251, 2017.
21. F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: Efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.
22. Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin. A fast order-based approach for core maintenance. In *ICDE*, pages 337–348, 2017.
23. F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.