When Engagement Meets Similarity: Efficient (k,r)-Core Computation on Social Networks

Fan Zhang[‡†], Ying Zhang[†], Lu Qin[†], Wenjie Zhang[§], ✉Xuemin Lin[‡§]

[‡]*East China Normal University,* [†]*CAI, University of Technology Sydney,* [§]*University of New South Wales*
fanzhang.cs@gmail.com, {ying.zhang, lu.qin}@uts.edu.au, {zhangw, lxue}@cse.unsw.edu.au

ABSTRACT

In this paper, we investigate the problem of (k,r)-core which intends to find cohesive subgraphs on social networks considering both user engagement and similarity perspectives. In particular, we adopt the popular concept of k-core to guarantee the engagement of the users (vertices) in a group (subgraph) where each vertex in a (k,r)-core connects to at least k other vertices. Meanwhile, we consider the pairwise similarity among users based on their attributes. Efficient algorithms are proposed to enumerate all *maximal* (k,r)-cores and find the *maximum* (k,r)-core, where both problems are shown to be NP-hard. Effective pruning techniques substantially reduce the search space of two algorithms. A novel (k,k')-core based (k,r)-core size upper bound enhances performance of the maximum (k,r)-core computation. We also devise effective search orders for two mining algorithms where search priorities for vertices are different. Comprehensive experiments on real-life data demonstrate that the maximal/maximum (k,r)-cores enable us to find interesting cohesive subgraphs, and performance of two mining algorithms is effectively improved by proposed techniques.

1. INTRODUCTION

Nowadays data becomes diverse and complex in real-life social networks, which not only consist of users and friendship, but also have various attribute values on each user. As such, social networks can be naturally modeled as *attributed graphs* where vertices represent users, edges represent friendship and vertex attribute is associated with specific properties, such as locations or keywords. Mining cohesive subgraphs is one of the most fundamental graph problems which aims to find groups of well-connected nodes (e.g., people), and a variety of models have been proposed such as clique [8], k-core [24], and k-truss [17]. Most of the existing work only consider the structure cohesiveness of the subgraphs. However, in practice we usually need to consider both structure and attribute perspectives when we aim to find a cohesive subgraph. In this paper, we move beyond the simple structure-based cohesive subgraph models and advocate a complicated but more realistic cohesive subgraph model on attributed graphs, namely (k,r)-core. Particularly, we consider two intuitive and important criteria for a cohesive subgraph in real-life social networks: *engagement* and *similarity*.

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

Engagement. It is a common practice to encourage the engagement of the group members by using the positive influence from their friends in the same group (e.g., [3, 11, 21, 22, 29]); that is, ensure there are a considerable number of friends for each individual user (vertex) in the group (subgraph). In [3], Bhawalkar and Kleinberg et al. use the game-theory to formally demonstrate that the popular k-core model can lead to a stable group (i.e., a cohesive subgraph regarding graph structure). In this paper, we adopt the k-core model on the graph structure, where each vertex in the subgraph has at least k neighbors (*structure constraint*).

Similarity. In addition to the engagement, we usually need to consider attribute similarities among users (vertices) in the group (subgraph). The similarity of two users can be derived from a given set of attributes (e.g., location, interests, and user generated content), which varies in different scenarios (e.g., [4, 15, 16, 23, 25]). By connecting two users (vertices) whose similarity exceeds a given threshold r, we get a *similarity graph* to capture the similarities among users. In this paper, we adopt the well-known *clique* model to capture the cohesiveness of users from similarity perspective; that is, the vertices of a cohesive subgraph in this paper form a clique on the similarity graph, which can ensure pairwise similarity among users (*similarity constraint*).

(k,r)-Core. The engagement and similarity criteria may often be used together to measure the sustainability of social groups. For instance, Facebook shows that both engagement (the number of friends in the group) and similarity (e.g., similar pages liked and distance closeness) are two important criteria when an existing Facebook group is recommended to a user [1]. To capture both engagement and similarity, we introduce the (k,r)-core model which is defined as follows. We say a connected subgraph of G is a (k,r)-core if and only if it satisfies both structure and similarity constraints. More specifically, given an attributed graph G, a (k,r)-core is a k-core of G (*structure constraint*) and the vertex set of the (k,r)-core induces a clique on the corresponding similarity graph (*similarity constraint*). A (k,r)-core is maximal if none of its supergraphs is a (k,r)-core. It is less interesting to find non-maximal (k,r)-cores. In this paper, we aim to efficiently enumerate the maximal (k,r)-cores.

Applications. In many social network applications, (k,r)-core can help to discover interesting groups, which are promising to become stable and active. These groups can greatly enhance the users' stickiness and their experience. For instance, game designers should increase the *stickiness* of games by encouraging, or even forcing, team playing [7]. By identifying candidate groups with high quality in terms of the engagement and similarity, (k,r)-core can help to quickly discover and recommend new groups. These groups have good potential to become active and stable. Below are detailed examples.

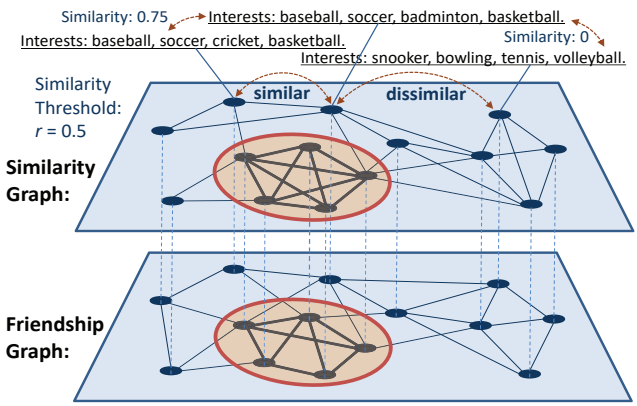


Figure 1: Group by Friendship and Interests

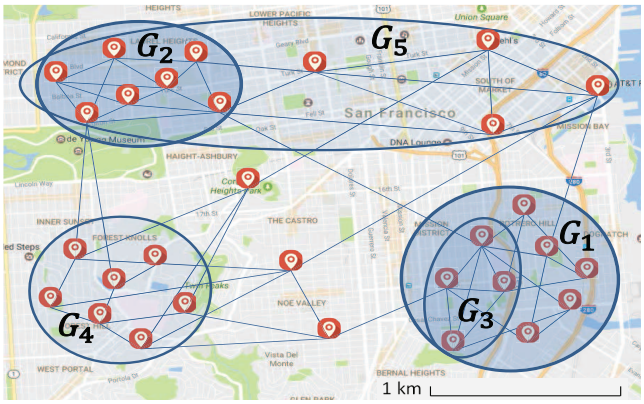


Figure 2: Group by Friendship and Locations

EXAMPLE 1 (INTEREST-BASED SOCIAL GROUPS). *In social networks such as Facebook and Weibo, the friendship information and mutual interests are widely used to recommend existing groups to users. In this paper, (k,r)-core can be used for group discovery, which recommends new promising groups to relevant users. In Figure 1, we use a set of keywords to describe the interests of each user (e.g., Facebook user). Jaccard similarity metric can be employed to measure the user similarity. Suppose $k = 3$ and the similarity threshold $r = 0.5$, the group within red circle is a maximal (k,r)-core [1]. In this group, each user has at least 3 friends and their interests are similar to others. This group is likely to remain stable, become active and enhance the interactions among users.*

EXAMPLE 2 (LOCATION-BASED GAME TEAMS). *By recommending game teams (groups) with potential to become sustainable and active, the game companies can greatly enhance user stickiness and improve game experience [7]. In many location-based online games such as Pokemon Go and Ingress, users play the games based on their locations and surroundings. For Pokemon Go, players would like to play the game with a group of people, in which there are some friends, to catch pokemons and attack gyms together. Because it is a location-based mobile game, players usually play it within a geographical range of frequently visited places such as their homes. Due to the diverse distribution of pokemons and pokestops, the places near the home of every player are likely to be visited, which naturally requires pairwise closeness (i.e., similarity) among group members. Consequently, (k,r)-cores can be good candidate groups where user engagement and similarity are guaranteed. With similar rationale, (k,r)-cores are useful to party games (e.g., Werewolf) which may be hosted at the homes of the team players.*

As illustrated in Figure 2, we can model the players, their friendship and locations as a graph on the map. Suppose $k = 3$ and the distance (similarity) threshold $r = 1$ km, G_4 and G_5 are not good candidate groups. Although each player pair in G_4 has close distance, their friendship is weak. Likewise, although each player in G_5 has at least k friends, some players cannot conveniently play with others because they are far away from others. However, maximal (k,r)-cores (i.e., G_1 and G_2) can effectively identify good candidate groups because each player has at least k friends in the same group, and the distance for every two players is at most r (i.e., they are similar). Note that although G_3 is also a (k,r)-core, it is less interesting because it is fully contained by a larger group, G_1.

[1]Note that the values of k and r can be tuned or learned for different cohesiveness requirements.

The size of a social group is an important factor to measure the potential impact and influence of this group. The groups with larger size are more likely to attract attention from the social network companies and users. Thus, it is interesting to find the maximum (k,r)-core which is the (k,r)-core with the largest number of vertices, and find the top-m maximal (k,r)-cores with the top-m largest numbers of vertices. In Example 1, the maximum (k,r)-core can help to evaluate the social impact potential of the interest-based social groups. In Example 2, the top-m maximal (k,r)-cores can identify m groups in which the game companies are more likely to be interested. In this paper, we also study the problem of efficiently finding the maximum (k,r)-core and top-m maximal (k,r)-cores.

Challenges and Contributions. Although there is a linear algorithm for k-core computation [2] (i.e., only consider structure constraint), we show that the problem of enumerating all maximal (k,r)-cores and finding the maximum (k,r)-core are both NP-hard because of the similarity constraint involved. A straightforward solution is the combination of the existing k-core and clique algorithms, e.g., enumerating the cliques on similarity graph and then checking the structure constraint on the graph. In Section 3 and the empirical study, we show that this is not promising because of the isolated processing of structure and similarity constraints. In this paper, we show that the performance can be immediately improved by considering two constraints (i.e., two pruning rules) at the same time without explicitly materializing the similarity graph. Then our technique contributions focus on further reducing the search space of two mining algorithms from the following three aspects: (i) effective pruning, early termination and maximal check techniques. (ii) (k,k')-core based approach to derive tight upper bound for the problem of finding the maximum (k,r)-core. and (iii) good search orders in two mining algorithms. Following is a summary of our principal contributions.

- We advocate a novel cohesive subgraph model for attributed graphs, called (k,r)-core, to capture the cohesiveness of subgraphs from both the graph structure and the vertex attributes. We prove that the problem of enumerating all maximal (k,r)-cores and finding the maximum (k,r)-core are both NP-hard. (Section 2)
- We develop efficient algorithms to enumerate the maximal (k,r)-cores with candidate pruning, early termination and maximal checking techniques. (Section 5)
- We also develop an efficient algorithm to find the maximum (k,r)-core. Particularly, a novel (k,k')-core based approach is proposed to derive a tight upper bound for the size of the candidate solution. (Section 6)

- Based on some key observations, we propose three search orders for enumerating maximal (k,r)-cores, checking maximal (k,r)-cores, and finding maximum (k,r)-core algorithms, respectively. (Section 7)
- Our empirical studies on real-life data demonstrate that interesting cohesive subgraphs can be identified by maximal (k,r)-cores and maximum (k,r)-core. The extensive performance evaluation shows that the techniques proposed in this paper can greatly improve the performance of two mining algorithms. (Section 8)

2. PRELIMINARIES

In this section, we first formally introduce the concept of (k,r)-core, then show that the two problems are NP-hard. Table 1 summarizes the mathematical notations used throughout this paper.

2.1 Problem Definition

We consider an undirected, unweighted, and attributed graph $G = (V, \mathcal{E}, A)$, where $V(G)$ (resp. $\mathcal{E}(G)$) represents the set of vertices (resp. edges) in G, and $A(G)$ denotes the attributes of the vertices. By $sim(u, v)$, we denote the similarity of two vertices u, v in $V(G)$ which is derived from their corresponding attribute values (e.g., users' geo-locations and interests) such as Jaccard similarity or Euclidean distance. For a given similarity threshold r, we say two vertices are dissimilar (resp. similar) if $sim(u, v) < r$ (resp. $sim(u, v) \geq r$) [2].

For a vertex u and a set S of vertices, $DP(u, S)$ (resp. $SP(u, S)$) denotes the number of other vertices in S which are dissimilar (resp. similar) to u regarding the given similarity threshold r. We use $DP(S)$ denote the number of dissimilar pairs in S. We use $S \subseteq G$ to denote that S is a subgraph of G where $\mathcal{E}(S) \subseteq \mathcal{E}(G)$ and $A(S) \subseteq A(G)$. By $deg(u, S)$, we denote the number of adjacent vertices of u in $V(S)$. Then, $deg_{min}(S)$ is the minimal degree of the vertices in $V(S)$. Now we formally introduce two constraints.

DEFINITION 1. ***Structure Constraint.*** *Given a positive integer k, a subgraph S satisfies the structure constraint if $deg(u, S) \geq k$ for each vertex $u \in V(S)$, i.e., $deg_{min}(S) \geq k$.*

DEFINITION 2. ***Similarity Constraint.*** *Given a similarity threshold r, a subgraph S satisfies the similarity constraint if $DP(u, S) = 0$ for each vertex $u \in V(S)$, i.e., $DP(S) = 0$.*

We then formally define the (k,r)-core based on structure and similarity constraints.

DEFINITION 3. ***(k,r)-core.*** *Given a connected subgraph $S \subseteq G$, S is a (k,r)-core if S satisfies both structure and similarity constraints.*

In this paper, we aim to find all maximal (k,r)-cores and the maximum (k,r)-core, which are defined as follows.

DEFINITION 4. ***Maximal (k,r)-core.*** *Given a connected subgraph $S \subseteq G$, S is a maximal (k,r)-core if S is a (k,r)-core of G and there exists no (k,r)-core S' of G such that $S \subset S'$.*

DEFINITION 5. ***Maximum (k,r)-core.*** *Let $\mathcal{R}$ denote all (k,r)-cores of an attributed graph G, a (k,r)-core $S \subseteq G$ is maximum if $|V(S)| \geq |V(S')|$ for every (k,r)-core $S' \in \mathcal{R}$.*

[2]Following the convention, when the distance metric (e.g., Euclidean distance) is employed, we say two vertices are similar if their distance is not larger than the given distance threshold.

Table 1: The summary of notations

Notation	Definition
G	a simple attributed graph
S,J,R	induced subgraphs
u, v	vertices in the attributed graph
$sim(u, v)$	similarity between u and v
$deg(u, S)$	number of adjacent vertex of u in S
$deg_{min}(S)$	minimal degree of the vertices in S
$DP(u, S)$	number of dissimilar vertices of u w.r.t S
$DP(S)$	number of dissimilar pairs of S
$SP(u, S)$	number of similar vertices of u w.r.t S
M	vertices chosen so far in the search
C	candidate vertices set in the search
E	relevant exclusive vertices set in the search
$\mathcal{R}(M,C)$	maximal (k,r)-cores derived from $M \cup C$
$SF(S)$ (i.e.,$SF_C(S)$)	every u in S with $DP(u, C) = 0$
$SF_{C \cup E}(S)$	every u in S with $DP(u, C \cup E) = 0$

Problem Statement. Given an attributed graph G, a positive integer k and a similarity threshold r, we aim to develop efficient algorithms for the following two fundamental problems: (i) enumerating all maximal (k,r)-cores in G; (ii) finding the maximum (k,r)-core in G.

EXAMPLE 3. *In Figure 2, all vertices are from the k-core where $k = 2$. G_1, G_2 and G_3 are the three (k,r)-cores. G_1 and G_2 are maximal (k,r)-cores while G_3 is fully contained by G_1. G_1 is the maximum (k,r)-core.*

2.2 Problem Complexity

We can compute k-core in linear time by recursively removing the vertices with a degree of less than k [2]. Nevertheless, the two problems studied in this paper are NP-hard due to the additional similarity constraint.

THEOREM 1. *Given a graph $G(V, \mathcal{E})$, the problems of enumerating all maximal (k,r)-cores and finding the maximum (k,r)-core are NP-hard.*

PROOF. Given a graph $G(V, \mathcal{E})$, we construct an attributed graph $G'(V', \mathcal{E}', A')$ as follows. Let $V(G') = V(G)$ and $\mathcal{E}(G') = \{(u, v) \mid u \in V(G'), v \in V(G'), u \neq v\}$, i.e., G' is a complete graph. For each $u \in V(G')$, we let $A(u) = adj(u, G)$ where $adj(u, G)$ is the set of adjacent vertices of u in G. Suppose a Jaccard similarity is employed, i.e., $sim(u, v) = \frac{|A(u) \cap A(v)|}{|A(u) \cup A(v)|}$ for any pair of vertices u and v in $V(G')$, and let the similarity threshold $r = \epsilon$ where ϵ is an infinite small positive number (e.g., $\epsilon = \frac{1}{2|V(G')|}$). We have $sim(u, v) \geq r$ if the edge $(u, v) \in \mathcal{E}(G)$, and otherwise $sim(u, v) = 0 < r$. Since G' is a complete graph, i.e., every subgraph $S \subseteq G'$ with $|S| \geq k$ satisfies the structure constraint of a (k,r)-core, the problem of deciding whether there is a k-clique on G can be reduced to the problem of finding a (k,r)-core on G' with $r = \epsilon$, and hence can be solved by the problem of enumerating all maximal (k,r)-cores or finding the maximum (k,r)-core. Theorem 1 holds due to the NP-hardness of the k-clique problem [14]. □

3. THE CLIQUE-BASED APPROACH

Let G' denote a new graph named *similarity graph* with $V(G') = V(G)$ and $\mathcal{E}(G') = \{(u, v) \mid sim(u, v) \geq r \ \& \ u, v \in V(G)\}$, i.e., G' connects the similar vertices in $V(G)$. Then, the set of vertices in a (k,r)-core satisfies the structure constraint on G and is a *clique* (i.e., a complete subgraph) on the similarity graph G' (because every vertex pair is similar in a (k,r)-core). This implies that we can use the existing clique algorithms on the similarity

graph to enumerate the (k,r)-core candidates, followed by a structure constraint check. More specifically, we may first construct the similarity graph G' by computing the pairwise similarity of the vertices. Then we enumerate the cliques in G', and compute the k-core on each induced subgraph of G for each clique. We can find the maximal (k,r)-cores after the maximal check. We may further improve the performance of this clique-based approach in the following three ways.

- Instead of enumerating cliques on the similarity graph G', we can first compute the k-core of G, denoted by S. Then, we apply the clique-based method on the similarity graph of each connected subgraph in S.
- An edge in S can be deleted if its corresponding vertices are *dissimilar*, i.e., there is no edge between these two vertices in similarity graph S'.
- We only need to compute k-core for each maximal clique because any maximal (k,r)-core derived from a non-maximal clique can be obtained from the maximal cliques.

The above three methods substantially improve the performance of the clique-based approach. Nevertheless, our experiments later will demonstrate that the improved clique-based approach are substantially outperformed by our baseline algorithm (Section 8.3), although the state-of-the-art k-core and clique computation methods have been applied [2, 27]. This further validates the effectiveness of the following techniques, which integrates computation of k-core and clique at each search step with effective optimizations.

4. WARMING UP FOR OUR APPROACH

For ease of understanding, we start with a straightforward set enumeration approach . The pseudo-code is given in Algorithm 1. At the initial stage (Line 1-2), we remove the edges in $\mathcal{E}(G)$ whose corresponding vertices are dissimilar, and then compute the k-core $\mathcal{S}$ of the graph G. For each connected subgraph $S \in \mathcal{S}$, the procedure NaiveEnum (Line 5) identifies all possible (k,r)-cores by enumerating and validating all the induced subgraphs of S. By $\mathcal{R}$, we record the (k,r)-cores seen so far. Lines 6-8 eliminate the non-maximal (k, r)-cores by checking all (k,r)-cores.

Algorithm 1: **EnumerateMKRC**(G, k, r)

```
   Input  : G : attributed graph, k : degree threshold, r : similarity threshold
   Output : M : Maximal (k,r)-cores
1  for each edge (u, v) in E(G) do
2      Remove edge (u, v) from G If sim(u, v) < r;
3  S ← k-core(G); R := ∅;
4  for each connected subgraph S in S do
5      R := R ∪ NaiveEnum(∅, S);
6  for each (k,r)-core R in R do
7      if there is a (k,r)-core R' ∈ R s.t. R ⊂ R' then
8          R := R \ R;
9  return R
```

During the NaiveEnum search procedure (Algorithm 2), the vertex set M incrementally retains the chosen vertices, and C retains the candidate vertices. As shown in Figure 3, the enumeration process corresponds to a *binary search tree* in which each leaf node represents a subset of S. In each non-leaf node, there are two branches. The chosen vertex will be moved to M from C in *expand branch*, and will be deleted from C in *shrink branch*, respectively.

Algorithm Correctness. We can safely remove dissimilar edges (i.e., edges whose corresponding vertices are dissimilar) at Line 1 and 2, since they will not be considered in any (k,r)-core due to the similarity constraint. For every (k,r)-core R in G, there is one and only one connected k-core subgraph S from $\mathcal{S}$ with $R \subseteq S$. Since all possible subsets of S (i.e., $2^{|S|}$ leaf nodes) are enumerated in the corresponding search tree, every (k,r)-core R can be accessed *exactly once* during the search. Together with the structure/similarity constraints and maximal property validation, we can output all maximal (k,r)-cores.

Algorithm 1 immediately finds the maximum (k,r)-core by returning the maximal (k,r)-core with the largest size.

Algorithm 2: **NaiveEnum**(M, C)

```
   Input  : M : chosen vertices, C : candidate vertices
   Output : R : (k,r)-cores
1  if C = ∅ and deg_min(M) ≥ k and DP(M) = 0 then
2      R := R ∪ R for every connected subgraph R ∈ M;
3  else
4      u ← choose a vertex in C;
5      NaiveEnum(M ∪ u, C \ u);        /* Expand */;
6      NaiveEnum(M, C \ u );           /* Shrink */;
```

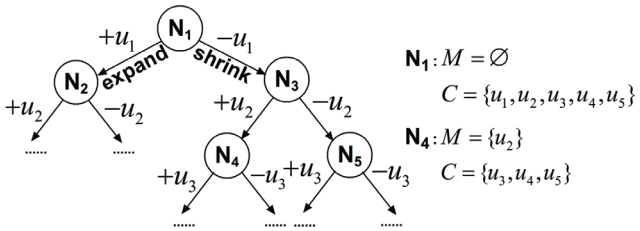


Figure 3: Example of the Search Tree

Algorithm 1 immediately finds the maximum (k,r)-core by returning the maximal (k,r)-core with the largest size.

5. FINDING ALL MAXIMAL (K,R)-CORES

In this section, we propose pruning techniques for the enumeration algorithm including candidate reduction, early termination, and maximal check techniques, respectively. Note that we defer discussion on search orders to Section 7.

5.1 Reducing Candidate Size

We present pruning techniques to explicitly/implicitly exclude some vertices from C.

5.1.1 Eliminating Candidates

Intuitively, when a vertex in C is assigned to (i.e., expand branch) M or discarded (i.e., shrink branch), we shall recursively remove some non-promising vertices from C due to structure and similarity constraints. The following two pruning rules are based on the definition of (k,r)-core.

THEOREM 2. ***Structure based Pruning***. *We can discard a vertex u in C if $deg(u, M \cup C) < k$.*

THEOREM 3. ***Similarity based Pruning***. *We can discard a vertex u in C if $DP(u, M) > 0$.*

Candidate Pruning Algorithm. If a chosen vertex u is extended to M (i.e., to the expand branch), we first apply the similarity pruning rule (Theorem 3) to exclude vertices in C which are dissimilar to u. Otherwise, none of the vertices will be discarded by the similarity constraint when we follow the shrink branch. Due to the removal of the vertices from C (expand branch) or u (shrink branch), we conduct structure based pruning by computing the k-core for vertices in $M \cup C$. Note that the search terminates if any vertex in M is discarded.

It takes at most $O(|C|)$ time to find dissimilar vertices of u from C. Due to the k-core computation, the structure based pruning takes linear time to the number of edges in the induced graph of $M \cup C$.

After applying the candidate pruning, following two important invariants always hold at each search node.

Similarity Invariant. We have

$$DP(u, M \cup C) = 0 \text{ for every vertex } u \in M \quad (1)$$

That is, M satisfies similarity constraint regarding $M \cup C$.

Degree Invariant. We have

$$deg_{min}(M \cup C) \geq k \quad (2)$$

That is, M and C together satisfy the structure constraint.

5.1.2 Retaining Candidates

In addition to explicitly pruning some non-promising vertices, we may implicitly reduce the candidate size by not choosing some vertices from C. In this paper, we say a vertex u is *similarity free* w.r.t C if u is similar to all vertices in C, i.e., $DP(u, C) = 0$. By $SF(C)$ we denote the set of similarity free vertices in C.

THEOREM 4. *Given that the pruning techniques are applied in each search step, we do not need to choose vertices from $SF(C)$ on both expand and shrink branches. Moreover, $M \cup C$ is a (k,r)-core if we have $C = SF(C)$.*

PROOF. For every vertex $u \in SF(C)$, we have $DP(u, M \cup C) = 0$ due to the similarity invariant of M (Equation 1) and the definition of $SF(C)$. Let M_1 and C_1 denote the corresponding chosen set and candidate after u is chosen for expansion. Similarly, we have M_2 and C_2 if u is moved to the shrink branch. We have $M_2 \subset M_1$ and $C_2 \subseteq C_1$, because there are no discarded vertices when u is extended to M while some vertices may be eliminated due to the removal of u in the shrink branch. This implies that $\mathcal{R}(M_2, C_2) \subseteq \mathcal{R}(M_1, C_1)$. Consequently, we do not need to explicitly discard u as the shrink branch of u is useless. Hence, we can simply retain u in C in the following computation.

However, $C = SF(C)$ implies every vertex u in $M \cup C$ satisfies the similarity constraint. Moreover, u also satisfies the structure constraint due to the degree invariant (Equation 2) of $M \cup C$. Consequently, $M \cup C$ is a (k,r)-core. □

Note that a vertex $u \in SF(C)$ may be discarded in the following search due to the structural constraint. Otherwise, it is moved to M when the condition $SF(C) = C$ holds. For each vertex u in C, we can update $DP(u, C)$ in a passive way when its dissimilar vertices are eliminated from the computation. Thus, it takes $O(n_d)$ time in the worst case where n_d denotes the number of dissimilar vertex pairs in C.

REMARK 1. *With similar rationale, we can move a vertex u directly from C to M if it is similarity free (i.e., $u \in SF(C)$) and is adjacent to at least k vertices in M. As this validation rule is trivial, it will be used in this paper without further mention.*

EXAMPLE 4. *In Figure 4, initially we have $M = \{u_6\}$ and $C = \{u_1, \ldots, u_5, u_7, \ldots, u_{10}\}$. We use the spatial distance of two vertices as their similarity, and the only dissimilar pair is u_4 and u_7. Suppose u_7 is chosen from C, following the expand branch, u_7 will be moved from C to M and then u_4 will be pruned due to the similarity constraint. Then we need to prune u_8 as $deg(u_8, M \cup C) < 3$. Thus, $M = \{u_6, u_7\}, E = \{u_4, u_8\}$. Since we have*

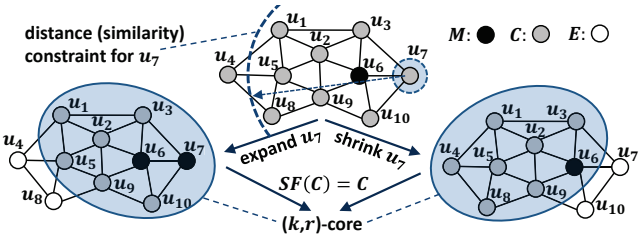


Figure 4: Pruning and Retaining Candidates

$SF(C) = C$, this search branch is terminated and we get a (k,r)-core of $M \cup C$. Regarding the shrink branch, u_7 is moved from C to E, which leads to the deletion of u_{10} due to structure constraint. Thus, $M = \{u_6\}, E = \{u_7, u_{10}\}$. Since we have $SF(C) = C$, this search branch is terminated and we get a (k,r)-core of $M \cup C$.

5.2 Early Termination

Trivial Early Termination. There are two trivial early termination rules. As discussed in Section 5.1, we immediately terminate the search if any vertex in M is discarded due to the structure constraint. We also terminate the search if M is disconnected to C. Both these stipulations will be applied in the remainder of this paper without further mention.

In addition to identifying the subtree that cannot derive any (k,r)-core, we further reduce the search space by identifying the subtrees that cannot lead to any *maximal* (k,r)-core. By E, we denote the related excluded vertices set for a search node of the tree, where the discarded vertices during the search are retained if they are similar to M, i.e., $DP(v, M) = 0$ for every $v \in E$ and $E \cap (M \cup C) = \emptyset$. We use $SF_C(E)$ to denote the similarity free vertices in E w.r.t the set C; that is, $DP(u, C) = 0$ for every $u \in SF_C(E)$. Similarly, by $SF_{C \cup E}(E)$ we denote the similarity free vertices in E w.r.t the set $E \cup C$.

THEOREM 5. ***Early Termination.*** *We can safely terminate the current search if one of the following two conditions hold:*
(i) there is a vertex $u \in SF_C(E)$ with $deg(u, M) \geq k$;
(ii) there is a set $U \subseteq SF_{C \cup E}(E)$, such that $deg(u, M \cup U) \geq k$ for every vertex $u \in U$.

PROOF. (i) We show that every (k,r)-core R derived from current M and C (i.e., $R \subseteq \mathcal{R}(M, C)$) can reveal a larger ($k$,$r$)-core by attaching the vertex u. For any $R \in \mathcal{R}$, we have $deg(u, R) \geq k$ because $deg(u, M) \geq k$ and $M \subseteq V(R)$. u also satisfies the similarity constraint based on the facts that $u \in SF_C(E)$ and $R \subseteq M \cup C$. Consequently, $V(R) \cup \{u\}$ forms a (k,r)-core. (ii) The correctness of condition (ii) has a similar rationale. The key idea is that for every $u \in U$, u satisfies the structure constraint because $deg(u, M \cup U) \geq k$; and u also satisfies the similarity constraint because $U \subseteq SF_{E \cup C}(E)$ implies that $DP(u, U \cup R) = 0$. □

Early Termination Check. It takes $O(|E|)$ time to check the condition (i) of Theorem 5 with one scan of the vertices in $SF_C(E)$. Regarding condition (ii), we may conduct k-core computation on $M \cup SF_{C \cup E}(E)$ to see if a subset of $SF_{C \cup E}(E)$ is included in the k-core. The time complexity is $O(n_e)$ where n_e is the number of edges in the induced graph of $M \cup C \cup E$.

5.3 Checking Maximal

In Algorithm 1 (Lines 6-8), we need to validate the maximal property based on all (k,r)-cores of G. The cost increases with both the number and the average size of the (k,r)-cores. Similar to the early termination technique, we use the following rule to check the maximal property.

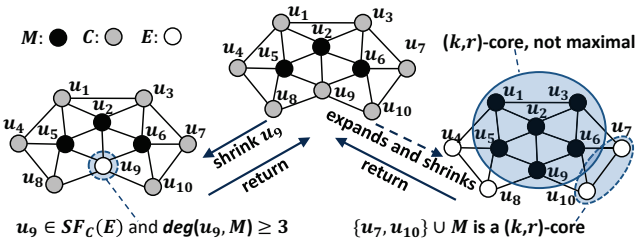


Figure 5: Early Termination and Check Maximal

THEOREM 6. ***Checking Maximal.*** *Given a (k,r)-core R, R is a maximal (k,r)-core if there doesn't exist a non-empty set $U \subseteq E$ such that $R \cup U$ is a (k,r)-core, where E is the excluded vertices set when R is generated.*

PROOF. E contains all discarded vertices that are similar to M according to the definition of the excluded vertices set. For any (k,r)-core R' which fully contains R, we have $R' \subseteq E \cup R$ because $R = M$ and $C = \emptyset$, i.e., the vertices outside of $E \cup R$ cannot contribute to R'. Therefore, we can safely claim that R is maximal if we cannot find R' among $E \cup R$. □

EXAMPLE 5. *In Figure 5, we have $M = \{u_2, u_5, u_6\}$, $C = \{u_1, u_3, u_4, u_7, u_8, u_9, u_{10}\}$. u_4 and u_7 is the only dissimilar pair. If u_9 is chosen from C on the shrink branch, u_9 is moved from C to E. Because u_9 is similar to every vertex in C and has 3 neighbors in M, the search is terminated for $u_9 \in SF_C(E)$ and $deg(u_9, M) \geq 3$ according to Theorem 5. If we expand and shrink the initial graph several times, the graph becomes $M = \{u_1, u_2, u_3, u_5, u_6, u_9\}$ and $E = \{u_4, u_7, u_8, u_{10}\}$. Here M is a (k,r)-core, but we can further extend u_7 and u_{10} to M and get a larger (k,r)-core. So M is not a maximal (k,r)-core.*

Since the maximal check algorithm is similar to our advanced enumeration algorithm, we delay providing the details of this algorithm to Section 5.4.

REMARK 2. *The early termination technique can be regarded as a lightweight version of the maximal check, which attempts to terminate the search before a (k,r)-core is constructed.*

5.4 Advanced Enumeration Method

Algorithm 3: **AdvancedEnum**(M, C, E)

Input : M : chosen vertices set, C : candidate vertices set, E : relevant excluded vertices set
Output : $\mathcal{R}$: maximal (k,r)-cores
1 Update C and E based on candidate pruning techniques (Theorem 2 and Theorem 3);
2 **Return If** current search can be terminated (Theorem 5);
3 **if** $C = SF(C)$ (Theorem 4) **then**
4 $M := M \cup C$;
5 $\mathcal{R} := \mathcal{R} \cup M$ *If* **CheckMaximal**(M, E) (Theorem 6);
6 **else**
7 $u \leftarrow$ a vertex in $C \setminus SF(C)$ (Theorem 4);
8 **AdvancedEnum**($M \cup u$, $C \setminus u$, E);
9 **AdvancedEnum**(M, $C \setminus u$, $E \cup u$);

In Algorithm 3, we present the pseudo code for our advanced enumeration algorithm which integrates the techniques proposed in previous sections. We first apply the candidate pruning algorithm outlined in Section 5.1 to eliminate some vertices based on structure/similarity constraints. Along with C, we also update E by including discarded vertices and removing the ones that are not similar to M. Line 2 may then terminate the search based on our early termination rules. If the condition $C = SF(C)$ holds, $M \cup C$ is a (k,r)-core according to Theorem 4, and we can conduct the maximal check (Lines 3-5). Otherwise, Lines 7-9 choose one vertex from $C \setminus SF(C)$ and continue the search following two branches.

Algorithm 4: **CheckMaximal**(M, C)

Input : M : chosen vertices, C : candidate vertices
Output : $isMax$: true if M is a maximal (k,r)-core
1 Update C based on similarity and structure constraint;
2 **if** M is a (k,r)-core **then**
3 Exit the algorithm with $isMax = false$ *If* $|\mathbf{M}^*| < |M|$;
4 **else if** $|C| > 0$ **then**
5 $u \leftarrow$ a vertex in C;
6 **CheckMaximal**($M \cup u$, $C \setminus u$);
7 **CheckMaximal**(M, $C \setminus u$);

Checking Maximal Algorithm. According to Theorem 6, we need to check whether some of the vertices in E can be included in the current (k,r)-core, denoted by M^*.This can be regarded as the process of further exploring the search tree by treating E as candidate C (Line 5 of Algorithm 3). Algorithm 4 presents the pseudo code for our maximal check algorithm.

To enumerate all the maximal (k,r)-cores of G, we need to replace the NaiveEnum procedure (Line 5) in Algorithm 1 using our advanced enumeration method (Algorithm 3). Moreover, the naive checking maximals process (Line 6-8) is not necessary since checking maximals is already conducted by our enumeration procedure (Algorithm 3). Since the search order for vertices does not affect the correctness, the algorithm correctness can be immediately guaranteed based on above analyses. It takes $O(n_e + n_d)$ times for each search node in the worst case, where n_e and n_d denote the total number of edges and dissimilar pairs in $M \cup C \cup E$.

6. FINDING THE MAXIMUM (K,R)-CORE

In this section, we first introduce the upper bound based algorithm to find the maximum (k,r)-core. Then a novel (k,k')-core approach is proposed to derive tight upper bound of the (k,r)-core size. Finally, we show the proposed upper bound and algorithm can be easily applied to finding the top-m maximal (k,r)-cores.

6.1 Algorithm for Finding the Maximum One

Algorithm 5 presents the pseudo code for finding the maximum (k,r)-core, where R denotes the largest (k,r)-core seen so far. There are three main differences compared to the enumeration algorithm (Algorithm 3). (i) Line 2 terminates the search if we find the current search is non-promising based on the upper bound of the core size, denoted by *KRCoreSizeUB*(M,C). (ii) We do not need to validate the maximal property. (iii) Along with the order of visiting the vertices, the order of the two branches also matters for quickly identifying large (k,r)-cores (Lines 6-12), which is discussed in Section 7.

To find the maximum (k,r)-core in G, we need to replace the NaiveEnum procedure (Line 2) in Algorithm 1 with the method in Algorithm 5, and remove the naive maximal check section of Algorithm 1 (Line 6-8). To quickly find a (k,r)-core with a large size, we start the algorithm from the subgraph S which holds the vertex with the highest degree. The maximum (k,r)-core is identified when Algorithm 1 terminates.

Algorithm Correctness. Since Algorithm 5 is essentially an enumeration algorithm with an upper bound based pruning technique, the correctness of this algorithm is clear if the $KRCoreSizeUB(M, C)$ at Line 2 is calculated correctly.

Algorithm 5: **FindMaximum**(M, C, E)

```
Input  : M : chosen vertices set , C : candidate vertices set, E :
         relevant excluded vertices set
Output : R : the largest (k,r)-core seen so far
1  Update C and E; Early terminate if possible;
2  if KRCoreSizeUB(M, C) > |R| then
3      if C = SF(C) then
4          R := M ∪ C;
5      else
6          u ← choose a vertex in C \ SF(C);
7          if Expansion is preferred then
8              FindMaximum(M ∪ u, C \ u, E);
9              FindMaximum(M, C \ u, E ∪ u);
10         else
11             FindMaximum(M, C \ u, E ∪ u);
12             FindMaximum(M ∪ u, C \ u, E);
```

Time Complexity. As shown in Section 6.2, we can efficiently compute the upper bound of core size in $O(n_e + n_s)$ time where n_s is the number of similar pairs w.r.t $M \cup C \cup E$. For each search node the time complexity of the maximum algorithm is same as that of the enumeration algorithm.

6.2 Size Upper Bound of (k,r)-Core

We use R to denote the (k,r)-core derived from $M \cup C$. In this way, $|M| + |C|$ is clearly an upper bound of $|R|$. However, it is very loose because it does not consider the similarity constraint.

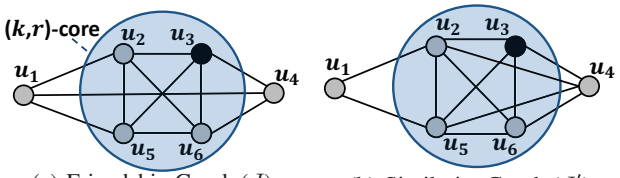
Recall that G' denotes a new graph that connects the similar vertices of $V(G)$, called *similarity graph*. By J and J', we denote the induced subgraph of vertices $M \cup C$ from graph G and the similarity graph G', respectively. Clearly, we have $V(J) = V(J')$. Because R is a *clique* on the similarity graph J' and the size of a k-clique is k, we can apply the maximum clique size estimation techniques to J' to derive the upper bound of $|R|$. Color [13] and k-core based methods [2] are two state-of-the-art techniques for maximum clique size estimation.

Color based Upper Bound. Let c_{min} denote the minimum number of colors to *color* the vertices in the similarity graph J' such that every two adjacent vertices in J' have different colors. Since a k-clique needs k number of colors to be *colored*, we have $|R| \leq c_{min}$. Therefore, we can apply graph coloring algorithms to estimate a small c_{min} [13].

***k*-Core based Upper Bound.** Let k_{max} denote the maximum k value such that k-core of J' is not empty. Since a k-clique is also a (k-1)-core, this implies that we have $|R| \leq k_{max} + 1$. Therefore, we may apply the existing k-core decomposition approach [2] to compute the maximal core number (i.e., k_{max}) on the similarity subgraph J'.

At the first glance, both the structure and similarity constraints are used in the above method because J itself is a k-core (structure constraint) and we consider the k_{max}-core of J' (similarity constraint). The upper bound could be tighter by choosing the smaller one from color based upper bound and k-core based upper bound. Nevertheless, we observe that the vertices in k_{max}-core of J' may not form a k-core on J since we only have J itself as a k-core. If so, we can consider k_{max}-1 as a tighter upper bound of R. Repeatedly, we have the largest k_{max}-i as the upper bound such that the corresponding vertices form a k-core on J and a (k_{max}-i)-core on J'. We formally introduce this (k, k')-core based upper bound in the following.

***(k,k')*-Core based Upper Bound.** We first introduce the concept of (k,k')-core to produce a tight upper bound of $|R|$. Theorem 7 shows that we can derive the upper bound for any possible (k,r)-core R based on the largest possible k' value, denoted by k'_{max}, from the corresponding (k,k')-core.



(a) Friendship Graph (J) (b) Similarity Graph (J')

Figure 6: Upper Bound Examples

DEFINITION 6. ***(k, k')-core.*** *Given a set of vertices U, the graph J and the corresponding similarity graph J', let J_U and J'_U denote the induced subgraph by U on J and J', respectively. If $deg_{min}(J_U) \geq k$ and $deg_{min}(J'_U) = k'$, U is a (k,k')-core of J and J'.*

THEOREM 7. *Given the graph J, the corresponding similarity graph J', and the maximum (k,r)-core R derived from J and J', if there is a (k,k')-core on J and J' with the largest k', i.e., k'_{max}, we have $|R| \leq k'_{max} + 1$.*

PROOF. Based on the fact that a (k,r)-core R is also a (k,k')-core with $k' = |R| - 1$ according to the definition of (k,r)-core, the theorem is proven immediately. □

EXAMPLE 6. *In Figure 6, we have $k = 3$, $M = \{u_3\}$ and $C = \{u_1, u_2, u_4, u_5, u_6\}$. Figure 6($a$) shows the induced subgraph J from $M \cup C$ on G and Figure 6(b) shows the similarity graph J' from $M \cup C$ on the similarity graph G'. We need at least 5 colors to color J', so the color based upper bound is 5. By core decomposition on similarity graph J', we get that the k-core based upper bound is 5 since $k_{max} = 4$ with 4-core $\{u_2, u_3, u_4, u_5, u_6\}$. Note that the vertices of this 4-core do not form a 3-core on J. Regarding the (k, k')-core based upper bound, we can find $k'_{max} = 3$ because there is a $(3, 3)$-core on J and J' with four vertices $\{u_2, u_3, u_5, u_6\}$, and there is no other (k, k')-core with a larger k' than k'_{max}. Consequently, the (k, k')-core based upper bound is 4, which is tighter than 5.*

6.3 Algorithm for (k,k')-Core Upper Bound

Algorithm 6 shows the details of the (k,k')-core based upper bound (i.e.,k'_{max}) computation, which conducts core decomposition [2] on J' with additional update which ensures the corresponding subgraph on J is a k-core. We use $deg[u]$ and $deg_{sim}[u]$ to denote the degree and similarity degree (i.e., the number of similar pairs from u) of u w.r.t $M \cup C$, respectively. Meanwhile, $NB[u]$ (resp. $NB_{sim}[u]$) denotes the set of adjacent (resp. similar) vertices of u. The key idea is to recursively mark the k' value of the vertices until we reach the maximal possible value. Line 1 sorts all vertices based on the increasing order of their similarity degrees. In each iteration, the vertex u with the lowest similarity degree has already reached its maximal possible k' (Line 3). Then Line 4 invokes the procedure KK'coreUpdate to remove u and decrease the degree (resp. similarity degree) of its neighbors (resp. similarity neighbors) at Lines 9-11 (resp. Lines 12-15). Note that we need to recursively remove vertices with degree smaller than k (Line 15) in the procedure. At Line 5, we need to reorder the vertices in H since their similarity degree values may be updated. According to Theorem 7, $k' + 1$ is returned at Line 6 as the upper bound of the maximum (k,r)-core size.

Algorithm 6: **KK'coreBound**(M, C)

Input : M : vertices chosen, C : candidate vertices
Output : k'_{max} : the upper bound for the size of the maximum (k,r)-core in $M \cup C$

```
1  H := vertices in M ∪ C with increasing order of their similarity degrees;
2  for each u ∈ H do
3      k' := deg_sim(u);
4      KK'coreUpdate(u, k', H);
5      reorder H accordingly;
6  return k' + 1

7  KK'coreUpdate(u, k', H)
8  Remove u from H;
9  for each v ∈ NB_sim[u] ∩ H do
10     if deg_sim[v] > k' then
11         deg_sim[v] := deg_sim[v] − 1;
12 for each v ∈ NB[u] ∩ H do
13     deg[v] := deg[v] − 1;
14     if deg[v] < k then
15         KK'coreUpdate(v, k', H);
```

Time Complexity. We can use an array H to maintain the vertices where $H[i]$ keeps the vertices with similarity degree i. Then the sorting of the vertices can be done in $O(|J|)$ time. The time complexity of the algorithm is $O(n_e + n_s)$, where n_e and n_s denote the number of edges in the graph J and the similarity graph J', respectively.

Algorithm Correctness. Let $k'_{max}(u)$ denote the largest k' value u can contribute to (k,k')-core of J. By H_j, we represent the vertices $\{u\}$ with $k'_{max}(u) \geq j$ according to the definition of (k,k')-core. We then have $H_j \subseteq H_i$ for any $i < j$. This implies that a vertex u on H_i with $k'_{max}(u) = i$ will not contribute to H_j with $i < j$. Thus, we can prove correctness by induction.

6.4 Finding the Top-m Maximal (k,r)-Cores

Besides the maximum (k,r)-core, social network service providers would also like to see the top-m maximal (k,r)-cores whose activeness reflects the hotness of the network. Users may be interested in these top-m communities which are most representative and appealing groups in the network. To find the top-m maximal (k,r)-cores, we can record current top-m largest maximal (k,r)-cores in Algorithm 5. In Line 2, the size upper bound is compared with the size of the minimum maximal (k,r)-core recorded. And in Line 4, we replace the minimum maximal (k,r)-core with current larger one if it is maximal. More specifically, the output of Algorithm 5 should be "$\{R_j \mid j \in N^+ \ \& \ j \leq m\}$: the top-$m$ largest maximal (k,r)-cores seen so far". Line 2 should be replaced by "If $\{KRCoreSizeUB(M, C) > |min(R_j)|\}$ then" where $min(R_j)$ is the minimum maximal (k,r)-core in $\{R_j \mid j \in N^+ \ \& \ j \leq m\}$. Line 4 should be replaced by "$min(R_j) := M \cup C$ If CheckMaximal(M, E);". Since there is no difference for Algorithm 6 towards finding the maximum and the top-m, the algorithm keeps same.

7. SEARCH ORDER

Section 7.1 briefly introduces some important measurements that should be considered for an appropriate visiting order. Then we investigate the visiting orders in three algorithms: finding the maximum (k,r)-core (Algorithm 5), advanced maximal (k,r)-core enumeration (Algorithm 3) and maximal check (Algorithm 4) at Section 7.2, Section 7.3, and Section 7.4, respectively.

7.1 Important Measurements

In this paper, we need to consider two kinds of search orders: (i) the vertex visiting order: the order of which vertex is chosen from candidate set C and (ii) the branch visiting order: the order of which branch goes first (expand first or shrink first). It is difficult to find simple heuristics or cost functions for two problems studied in this paper because, generally speaking, finding a maximal/maximum (k,r)-core can be regarded as an optimization problem with two constraints. On one hand, we need to reduce the number of dissimilar pairs to satisfy the similarity constraint, which implies eliminating a considerable number of vertices from C. On the other hand, the structure constraint and the maximal/maximum property favors a larger number of edges (vertices) in $M \cup C$; that is, we prefer to eliminate fewer vertices from C.

To accommodate this, we propose three measurements where M' and $\mathbf{C}'$ denote the updated M and C after a chosen vertex is extended to M or discarded.

- Δ_1 : the change of number of dissimilar pairs, where

$$\Delta_1 = \frac{DP(C) - DP(\mathbf{C}')}{DP(C)} \tag{3}$$

  Note that we have $DP(u, M \cup C) = 0$ for every $u \in M$ according to the similarity invariant (Equation 1).

- Δ_2 : the change of the number of edges, where

$$\Delta_2 = \frac{|\mathcal{E}(M \cup C)| - |\mathcal{E}(M' \cup \mathbf{C}')|}{|\mathcal{E}(M \cup C)|} \tag{4}$$

  Recall that $|\mathcal{E}(V)|$ denote the number of edges in the induced graph from the vertices set V.

- $deg(u, M \cup C)$: Degree. We also consider the degree of the vertex as it may reflect its importance. In our implementation, we choose the vertex with highest degree at the initial stage (i.e., $M = \emptyset$).

7.2 Finding the Maximum (k,r)-Core

Since the size of the largest (k,r)-core seen so far is critical to reduce the search space, we aim to quickly identify the (k,r)-core with larger size. One may choose to carefully discard vertices such that the number of edges in M is reduced slowly (i.e., only prefer smaller Δ_2 value). However, as shown in our empirical study, this may result in poor performance because it usually takes many search steps to satisfy the structure constraint. Conversely, we may easily fall into the trap of finding (k,r)-cores with small size if we only insist on removing dissimilar pairs (i.e., only favor larger Δ_1 value).

In our implementation, we use a cautious greedy strategy where a parameter λ is used to make the trade-off. In particular, we use $\lambda\Delta_1 - \Delta_2$ to measure the suitability of a branch for each vertex in $C \setminus SF(C)$. In this way, each candidate has two scores. The vertex with the highest score is then chosen and its branch with higher score is explored first (Line 6-12 in Algorithm 5).

For time efficiency, we only explore vertices within two hops from the candidate vertex when we compute its Δ_1 and Δ_2 values. It takes $O(n_c \times (d_1^2 + d_2^2))$ time where n_c denote the number of vertices in $C \setminus SF(C)$, and d_1 (resp. d_2) stands for the average degree of the vertices in J (resp. J').

7.3 Enumerating All Maximal (k,r)-Core

The ordering strategy in this section differs from finding the maximum in two ways.

(i) We observe that Δ_1 has much higher impact than Δ_2 in the enumeration problem, so we adopt the Δ_1-then-Δ_2 strategy; that is, we prefer the larger Δ_1, and the smaller Δ_2 is considered if there is a tie. This is because the enumeration algorithm does not prefer (k,r)-core with very large size since it eventually needs to enumerate all maximal (k,r)-cores. Moreover, by the early termination technique proposed in Section 5.2, we can avoid exploring many non-promising subtrees that were misled by the greedy heuristic.

(ii) We do not need to consider the search order of two branches because both must be explored eventually. Thus, we use the score summation of the two branches to evaluate the suitability of a vertex. The complexity of this ordering strategy is the same as that in Section 7.2.

7.4 Checking Maximal

The search order for checking maximals is rather different than the enumeration and maximum algorithms. Towards the checking maximals algorithm, it is cost-effective to find a *small* (k,r)-core which fully contains the candidate (k,r)-core. To this end, we adopt a short-sighted greedy heuristic. In particular, we choose the vertex with the largest degree and the expand branch is always preferred as shown in Algorithm 4. By continuously maintaining a priority queue, we fetch the vertex with the highest degree in $O(\log |C|)$ time.

8. PERFORMANCE EVALUATION

This section evaluates the effectiveness and efficiency of our algorithms through comprehensive experiments.

8.1 Experimental Setting

Algorithms. To the best of our knowledge, there are no existing works that investigate the problem of (k,r)-core. In this paper, we implement and evaluate 2 baseline algorithms, 2 advanced algorithms and the clique-based algorithm which are described in Table 3. Since the naive method in Section 4 is extremely slow even on a small graph, we employ `BasEnum` and `BasMax` as the baseline algorithms in the empirical study for the problem of enumerating all maximal (k,r)-cores and finding the maximum (k,r)-core, respectively. In Table 3, we also show the name for each technique, which may be equipped or unloaded for the mentioned algorithms.

Datasets. Four real datasets are used in our experiments. The original data of `DBLP` was downloaded from `http://dblp.uni-trier.de` and the remaining three datasets were downloaded from `http://snap.stanford.edu`. In `DBLP`, we consider each author as a vertex with attribute of counted "attended conferences" and "published journals" list. There is an edge for a pair of authors if they have at least one co-authored paper. We use *Weighted Jaccard Similarity* between the corresponding attributes (counted conferences and journals) to measure the similarity between two authors. In `Pokec`, we consider each user to be a vertex with personal interests. We use *Weighted Jaccard Similarity* as the similarity metric. And there is an edge between two users if they are friends. In `Gowalla` and `Brightkite`, we consider each user as a vertex along with his/her location information. The graph is constructed based on friendship information. We use *Euclidean Distance* between two locations to measure the similarity between two users. Table 2 shows the statistics of the four datasets.

Table 2: Statistics of Datasets

Dataset	Nodes	Edges	d_{avg}	d_{max}
`Brightkite`	58,228	194,090	6.67	1098
`Gowalla`	196,591	456,830	4.65	9967
`DBLP`	1,566,919	6,461,300	8.25	2023
`Pokec`	1,632,803	8,320,605	10.19	7266

Table 3: Summary of Algorithms

Technique	Description
`CR`	The candidate retaining technique (Theorem 4).
`ET`	The early termination technique (Theorem 5).
`CM`	The checking maximal technique (Theorem 6).
`CK`	A tighter upper bound from the color based and the k-core based upper bound. (Section 6.2).
`UB`	The (k,k')-core upper bound technique (Theorem 7).
`SO`	The best search order is applied (Section 7).
Algorithm	**Description**
`Clique+`	The advanced clique-based algorithm proposed in Section 3, using the clique and k-core computation algorithms in [27] and [2], respectively. The source code for maximal clique enumeration was downloaded from `http://www.cse.cuhk.edu.hk/~jcheng/publications.html`.
`BasEnum`	The basic enumeration method proposed in Algorithm 1 including the structure and similarity constraints based pruning techniques (Theorems 2 and 3 in Section 5.1). The best search order (Δ_1-then-Δ_2, in Section 7.3) is applied.
`AdvEnum`	`AdvEnum` = `BasEnum+CR+ET+CM`. The advanced enumeration algorithm proposed in Section 5.4 that applies all advanced pruning techniques including: candidate size reduction (Theorems 2, 3 and 4 in Section 5.1), early termination (Theorem 5 in Section 5.2) and checking maximals (Theorem 6 in Section 5.3). Moreover, the best search order is used (Δ_1-then-Δ_2, in Section 7.3).
`BasMax`	The algorithm proposed in Section 6.1 with the upper bound replaced by a naive one: $\|M\| + \|C\|$. The best search order is applied ($\lambda\Delta_1 - \Delta_2$, in Section 7.2).
`AdvMax`	`AdvMax` = `BasMax+UB`. The advanced finding maximum (k,r)-core algorithm proposed in Section 6.1 including (k,k')-core based upper bound technique (Algorithm 6). Again, the best search order is applied ($\lambda\Delta_1 - \Delta_2$, in Section 7.2).

Parameters. We conducted experiments using different settings of k and r. We set reasonable positive integers for k, which varied from 3 to 15. In `Gowalla` and `Brightkite`, we used Euclidean distance as the distance threshold r, ranging from 1 km to 500 km. The pairwise similarity distributions are highly skewed in `DBLP` and `Pokec`. Thus, we used the thousandth of the pairwise similarity distribution in decreasing order which grows from top 1‰ to top 15‰ (i.e., the similarity threshold value drops). Regarding the search orders of the `AdvMax` and `BasMax` algorithms, we set λ to 5 by default.

All programs were implemented in standard C++ and compiled with G++ in Linux. All experiments were performed with Intel Xeon 2.3GHz CPUs and a Redhat Linux system. The time cost is set to INF if an algorithm did not terminate within one hour. The source code is available at `https://sites.google.com/view/fanzhang`.

8.2 Effectiveness

We conducted a case study on `DBLP` to demonstrate the effectiveness of our (k,r)-core model. Compared to k-core, (k,r)-core enables us to find more valuable information with the additional similarity constraint on the vertex attribute. Figure 7 shows a case of `DBLP` with $k = 15$ and r= 3‰[3].

In Figure 7, all authors come from the same k-core based on their co-authorship information alone (their structure constraint). While

[3]To avoid the noise, we enforce that there are at least three co-authored papers between two connected authors in the case study.

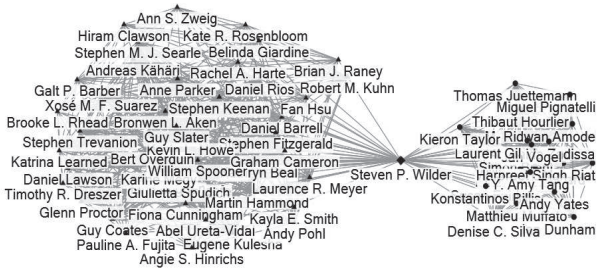
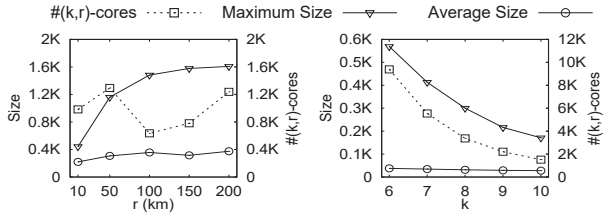


Figure 7: Case Study on DBLP (k=15, r=top 3‰)



(a) Gowalla, k=5 (b) DBLP, r=top 3‰

Figure 8: (k,r)-core Statistics

there are two (k,r)-cores with one common author named Steven P. Wilder, if we also consider their research background (their similarity constraint). We find the result of (k,r)-cores is consistent with reality that there are two groups of people with Dr. Wilder from both side.

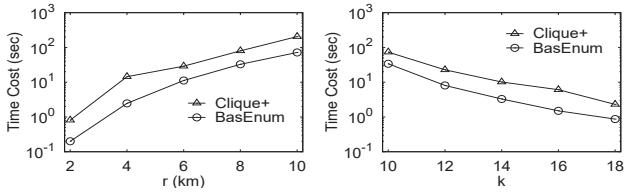
We also report the number of (k,r)-cores, the average size and maximum size of (k,r)-cores on `Gowalla` and `DBLP`. Figure 8(a) and (b) show that both maximum size of (k,r)-cores and the number of (k,r)-cores are much more sensitive to the change of r or k on the two datasets, compared to the average size.

8.3 Efficiency

In this section, we evaluate the efficiency of the techniques proposed in this paper and report the time costs of the algorithms.

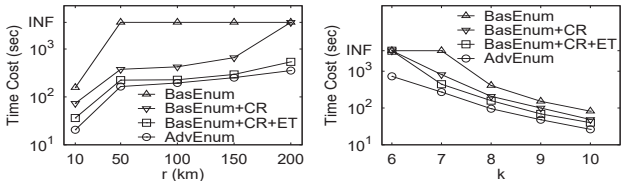
Evaluating the Clique-based Method. In Figure 9, we evaluate the time cost of the maximal (k,r)-core enumeration for `Clique+` and `BasEnum` on the `Gowalla` and `DBLP` datasets. In the experiments, `BasEnum` always outperform `Clique+` by a stable margin because we apply pruning rules in `BasEnum` with the best search order and a large number of of cliques are materialized in the similarity graphs for `Clique+`. Consequently, we exclude `Clique+` from the following experiments. This supports the insight that for a problem of computing cohesive subgraphs on dual graphs, a careful integration of existing cohesive subgraph computations at each search step (e.g., `BasEnum`) is better than computing the two kinds of cohesive subgraphs sequentially (e.g., `Clique+`).

Evaluating the Pruning Techniques. In Figure 10, we evaluate the efficiency of our pruning techniques on `Gowalla` and `DBLP` by incrementally integrating these techniques from `BasEnum`, `BasEnum+CR`, `BasEnum+CR+ET` to `AdvEnum` (`BasEnum+CR+ET+CM`). Note that the best search order is used



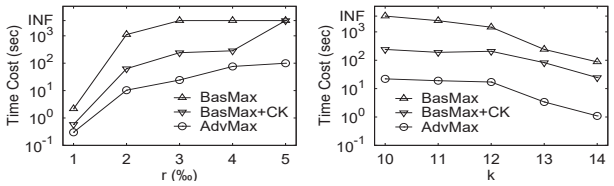
(a) Gowalla, k=5 (b) DBLP, r=top 3‰

Figure 9: Evaluate Clique-based Method



(a) Gowalla, k=5 (b) DBLP, r=top 3‰

Figure 10: Evaluate Pruning Techniques



(a) DBLP, k=10 (b) DBLP, r=top 3‰

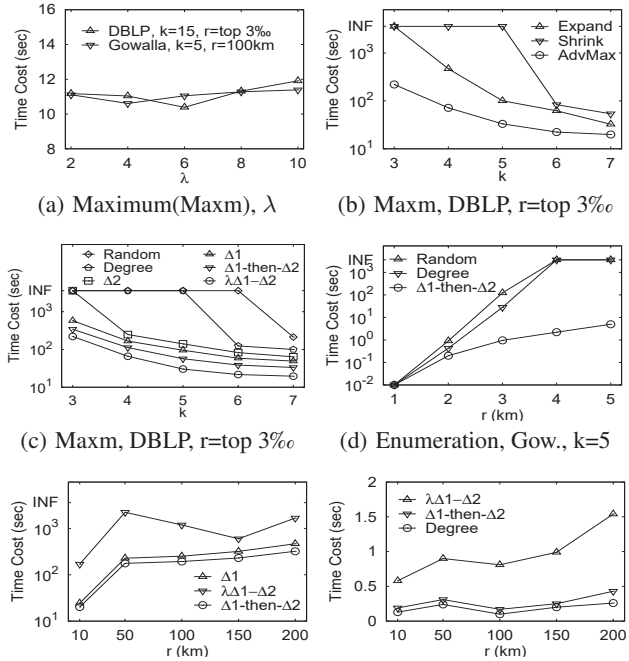
Figure 11: Evaluate Upper Bounds

for all algorithms. Among these techniques, Theorem 4 achieves the best speedup because the search on $SF(C)$ is skipped and $SF(C)$ may be large. The results in Figure 10 confirm that all techniques contribute to enhance the performance of `AdvEnum`.

Evaluating the Upper Bound Technique. Figure 11 demonstrates the effectiveness of the (k,k')-core based upper bound technique (Algorithm 6) on DBLP by varying the values of r and k. In `BasMax+CK`, we used the better upper bound from color and k-core based upper bound techniques (Section 6.2) [13, 2]. Studies show that `BasMax+CK` greatly enhances performance compared to the naive upper bound $|M| + |C|$ (used in `BasMax`). Nevertheless, our (k,k')-core based upper bound technique (`AdvMax`) outperforms `BasMax+CK` by a large margin because it can better exploit the structure/similarity constraints.

Evaluating the Search Orders. In this experiment, we evaluate the effectiveness of the three search orders proposed for the maximum algorithm (Section 7.2, Figure 12(a)-(c)), enumeration algorithm (Section 7.3, Figure 12(d)-(e)) and the checking maximal algorithm (Section 7.4, Figure 12(f)). We first tune λ value for the search order of `AdvMax` in Figure 12(a) against `DBLP` and `Gowalla`. In the following experiments, we set λ to 5 for maximum algorithms. Figure 12(b) verifies the importance of the adaptive order for the two branches on `DBLP` where Expand (resp. Shrink) means the expand (resp. shrink) branch is always preferred in `AdvMax`. In Figure 12(c), we investigate a set of possible order strategies for `AdvMax`. As expected, the $\lambda\Delta_1 - \Delta_2$ order proposed in Section 7.2 outperforms the other alternatives including random order, degree based order (Section 7.4, used for checking maximals), Δ_1 order, Δ_2 order and Δ_1-then-Δ_2 order (Section 7.3, used by `AdvEnum`). Similarly, Figure 12(d) and (e) confirm that the Δ_1-then-Δ_2 order is the best choice for `AdvEnum` compared to the alternatives. Figure 12(f) shows that the degree order achieves the best performance for the checking maximal algorithm (Algorithm 4) compared to the two orders used by `AdvEnum` and `AdvMax`.

Effect of Different Datasets. Figure 13 evaluates the performance of the enumeration and maximum algorithms on four datasets with $k = 10$. We use `AdvEnum-SO` to denote the `AdvEnum` algorithm *without* the best search order while all other advanced techniques applied (degree order is used instead). Figure 13(a) demonstrates the efficiency of those techniques and search orders on four datasets. We also demonstrate the efficiency of the upper bound and search order for the maximum algorithm in Figure 13(b), where three algorithms are evaluated (`AdvMax-SO`, `BasMax`, and `AdvMax`).

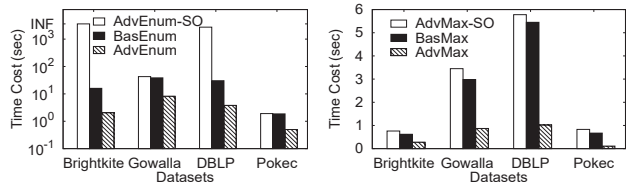


(a) Maximum(Maxm), λ (b) Maxm, DBLP, r=top 3‰

(c) Maxm, DBLP, r=top 3‰ (d) Enumeration, Gow., k=5

(e) Enumeration, Gow., k=5 (f) Maximal, Gow., k=5

Figure 12: Evaluate Search Orders



(a) Enumeration (b) Maximum

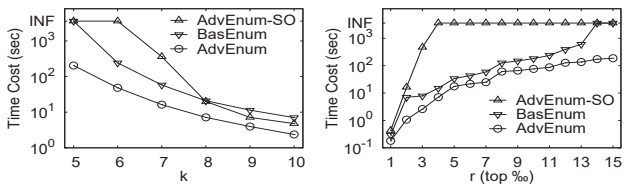
Figure 13: Performance on Four Datasets

Effect of k and r. Figure 14 studies the impact of k and r for the three enumeration algorithms on `Gowalla` and `DBLP`. As expected, Figure 14(a) shows that the time cost drops when k grows because many more vertices are pruned by the structure constraint. In Figure 14(b), the time costs grow when r increases because more vertices will be included in (k,r)-cores when the similarity threshold drops. Similar trends are also observed in Figure 15 for three maximum algorithms. Moreover, Figure 14 and 15 further confirm the effectiveness of proposed techniques. `AdvMax` greatly outperforms `AdvEnum` under the same setting because `AdvMax` can further cut-off the search tree based on the derived upper bound of the (k,r)-core size and does not need maximal check.

9. RELATED WORK

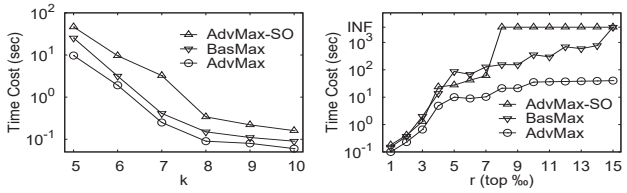
Mining Cohesive Subgraphs on Graphs. A variety of cohesive subgraph models have been proposed such as clique [8], k-core [24], and k-truss [17]. Below, we introduce k-core and clique, which are used in this paper to capture user engagement and attribute similarity, respectively.

k-core. The model of k-core, proposed by [24], has been widely used in many applications such as social contagion [26], influence study [19], and user engagement [3, 22, 33, 34]. Batagelj and Zaversnik presented a liner in-memory algorithm for core decomposition [2]. I/O efficient algorithms [28] were proposed for core decomposition on graphs that cannot fit in the main memory. Lee *et* al. [21] proposed an extension of k-core, namely (k,d)-core, which



(a) Gowalla, r=100 (b) DBLP, k=15

Figure 14: Effect of k and r for Enumeration



(a) Gowalla, r=100 (b) DBLP, k=15

Figure 15: Effect of k and r for Maximum

is different with our model because they enforce both k-core and d-truss structures on the same graph.

Clique. As a fundamental graph problem, maximal clique enumeration has been extensively studied. Most clique algorithms (e.g., [5]) are based on backtracking search. Eppstein and Strash [9] further speedup maximal clique enumeration by selecting pivots with good potential to reduce the search space in backtracking. Chang *et* al. study maximal clique enumeration in a sparse graph [6]. Recently, Wang *et* al. [27] utilize the overlaps among cliques to speedup the maximal cliques enumeration.

None of the above algorithms considered the computation of k-core and clique at the same time. In this paper, we show that the computation of (k,r)-core is much more challenging than the individual computation of k-core and clique. Moreover, Our empirical study shows that it is less efficient to compute (k,r)-core by sequentially applying the k-core and clique techniques.

Mining Attributed Graphs. It is common to use attributed graphs under various scenarios for real-world social network studies on both research and industry [12, 18]. A large amount of classical graph queries have been investigated on attributed graphs such as clustering [31], community detection [32], and network modeling [18]. None of these work combine k-core and pairwise similarity computation on attributed graphs.

Recently, there are some investigations on the problem of cohesive subgraph computation on attributed graphs. Wu *et* al. [30] developed efficient algorithms to find a set of nodes which are connected in structural graph and the corresponding subgraph on conceptual graph is the densest. Their result excludes some cohesive subgraphs where vertices have high engagement and similarity. Zhu *et* al. [35] studied finding the k-core within a given spatial region and contains a query point. Their approach is designed for community search on geo-social networks while we detect communities considering similarity of various attributes instead of a specific region. Fang *et* al. [11] proposed algorithms to find a subgraph related to a query point considering cohesiveness on both structure and keyword similarity, while it focuses on maximizing the number of common keywords of the vertices in the subgraph. Their recent work [10] aims to find a connected cohesive subgraph which satisfies a minimum degree of k for every vertex in the subgraph and has the minimum spatial radius. To the best of our knowledge, this paper is the first work to advocate a general cohesive subgraph model to consider both user engagement and attribute similarity on various kinds of attributed graphs. The techniques developed in related papers have not been found applicable to solving our problem.

10. DISCUSSION

In many real-life networks, it is rather natural to consider both structure and attribute values in many graph problems. As cohesive subgraph mining is one of the most fundamental problems, we can expect a variety of extensions of (k,r)-core model can be used for different scenarios by (i) applying possible combinations of existing cohesive subgraph models (e.g., k-core, k-truss, clique, quasi-clique, and dense subgraph); and (ii) considering the cohesive subgraph computation on multi-dimensional networks [20].

The techniques developed in this paper can shed light on the computation of these models. For instance, our study suggests that it is less efficient to sequentially apply the state-of-the-art techniques for each constraint (i.e., cohesive subgraph model). Instead, we need to carefully integrate the computation of the multiple constraints at each search step. Our study also indicates that, to deal with the multiple constraints, it is crucial to develop advanced early termination and maximal check techniques as well as design good visiting orders. When finding the (k,r)-trusses whose vertices form k-truss on the graph and clique on the similarity graph, the (k,k')-core upper bound technique can be directly applied to this problem because a k-truss is also a $(k$-1$)$-core. As to the problem of (k, λ)-core in which vertices form k-core on the graph and λ-quasi-clique on the similarity graph, the (k,k')-core upper bound is also applicable. With similar rationale, other proposed techniques can also be extended or provide insights to the counterparts of new cohesive subgraph models on attributed graphs.

11. CONCLUSION

In this paper, we propose a novel cohesive subgraph model, called (k,r)-core, which considers the cohesiveness of a subgraph from the perspective of both graph structure and vertex attribute. We show that the problem of enumerating the maximal (k,r)-cores and finding the maximum (k,r)-core are both NP-hard. Several novel pruning techniques are proposed to improve algorithm efficiency. We also devise effective search orders for enumeration, maximum and maximal check algorithms. Extensive experiments on real-life networks demonstrate the effectiveness of the (k,r)-core model, as well as the efficiency of our techniques.

12. ACKNOWLEDGMENTS

Ying Zhang is supported by ARC DE140100679 and DP170103710. Lu Qin is supported by ARC DE140100999 and DP160101513. Wenjie Zhang is supported by ARC DP150103071 and DP150102728. Xuemin Lin is supported by NSFC61232006, ARC DP150102728, DP140103578 and DP170101628.

13. REFERENCES

[1] How does facebook suggest groups for me to join? `https://www.facebook.com/help/382485908586472?helpref=uf_permalink`. Accessed: 14 Mar. 2017.

[2] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[3] K. Bhawalkar, J. M. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. *SIAM J. Discrete Math.*, 29(3):1452–1475, 2015.

[4] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *MSR*, pages 137–143, 2006.

[5] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[6] L. Chang, J. X. Yu, and L. Qin. Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1):173–186, 2013.

[7] K. Chen and C. Lei. Network game design: hints and implications of player interaction. In *NETGAMES*, page 17, 2006.

[8] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *KDD*, pages 1240–1248, 2012.

[9] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In *SEA*, pages 364–375, 2011.

[10] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[11] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[12] Y. Fang, H. Zhang, Y. Ye, and X. Li. Detecting hot topics from twitter: A multiview approach. *J. Information Science*, 40(5):578–593, 2014.

[13] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *JACM*, 23(1):43–49, 1976.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[15] M. K. Goldberg, S. Kelley, M. Magdon-Ismail, K. Mertsalov, and A. Wallace. Finding overlapping communities in social networks. In *SocialCom/PASSAT*, pages 104–113, 2010.

[16] D. Hristova, M. Musolesi, and C. Mascolo. Keep your friends close and your facebook friends closer: A multiplex network approach to the analysis of offline and online social ties. In *ICWSM*, 2014.

[17] X. Huang, W. Lu, and L. V. S. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*, pages 77–90, 2016.

[18] J. J. P. III, S. Moreno, T. L. Fond, J. Neville, and B. Gallagher. Attributed graph models: modeling network structure with correlated attributes. In *WWW*, pages 831–842, 2014.

[19] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888–893, 2010.

[20] M. Kivelä and e. Arenas. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 2014.

[21] 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.

[22] F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In *CIKM*, pages 469–478, 2013.

[23] A. Nanopoulos, H. Gabriel, and M. Spiliopoulou. Spectral clustering in social-tagging systems. In *WISE*, pages 87–100, 2009.

[24] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[25] P. Singla and M. Richardson. Yes, there is a correlation - from social networks to personal behavior on the web. In *WWW*, pages 655–664, 2008.

[26] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 109(16):5962–5966, 2012.

[27] J. Wang, J. Cheng, and A. W. Fu. Redundancy-aware maximal cliques. In *KDD*, pages 122–130, 2013.

[28] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu. I/O efficient core graph decomposition at web scale. In *ICDE*, pages 133–144, 2016.

[29] S. Wu, A. D. Sarma, A. Fabrikant, S. Lattanzi, and A. Tomkins. Arrival and departure dynamics in social networks. In *WSDM*, pages 233–242, 2013.

[30] Y. Wu, R. Jin, X. Zhu, and X. Zhang. Finding dense and connected subgraphs in dual networks. In *ICDE*, pages 915–926, 2015.

[31] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516, 2012.

[32] J. Yang, J. J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *ICDM*, pages 1151–1156, 2013.

[33] 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.

[34] 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.

[35] Q. Zhu, H. Hu, J. Xu, and W. Lee. Geo-social group queries with minimum acquaintance constraint. *CoRR*, abs/1406.7367, 2014.