检索证明

根据委托方提供的论文目录（2019），经 SCIE ,JCR 网络数据库检索，广州大学**张帆(Zhang, Fan)**发表的论文被 SCIE 收录了 **1** 篇（第一作者，通讯作者）。题录如下：

标题: Efficient community discovery with user engagement and similarity
作者: **Zhang, F (Zhang, Fan)**;Lin, XM (Lin, Xuemin);Zhang, Y (Zhang, Ying);Qin, L (Qin, Lu);Zhang, WJ (Zhang, Wenjie)
来源出版物: VLDB JOURNAL
卷: 28　期: 6　页: 987-1012
出版年: DEC 2019
在线发表:OCT 2019
入藏号: WOS:000492650700002
语言: English
文献类型: Article
地址: [Zhang, Fan] Guangzhou Univ, Guangzhou, Guangdong, Peoples R China
[Lin, Xuemin; Zhang, Wenjie] Univ New South Wales, Sydney, NSW, Australia
[Lin, Xuemin] East China Normal Univ, Shanghai, Peoples R China
[Zhang, Ying; Qin, Lu] Univ Technol Sydney, Ctr AI, Sydney, NSW, Australia
通讯作者地址: [Zhang, Fan] (corresponding author), Guangzhou Univ, Guangzhou, Guangdong, Peoples R China
电子邮件地址:
fanzhang.cs@gmail.com;lxue@cse.unsw.edu.au;ying.zhang@uts.edu.au;lu.qin@uts.edu.au;zhangw@cse.unsw.edu.au
影响因子(2019): 2.904
学科类别、分区及期刊排行百分比 ：

JCR® 类别	类别中的排序	JCR 分区
COMPUTER SCIENCE, INFORMATION SYSTEMS	59/156	Q2
COMPUTER SCIENCE, HARDWARE & ARCHITECTURE	15/53	Q2

特此证明

广州大学图书馆
学科服务部
2022 年 5 月 9 日

The VLDB Journal
https://doi.org/10.1007/s00778-019-00579-4

REGULAR PAPER

Efficient community discovery with user engagement and similarity

Fan Zhang[1] · Xuemin Lin[2,3] · Ying Zhang[4] · Lu Qin[4] · Wenjie Zhang[2]

Received: 7 November 2018 / Revised: 30 August 2019 / Accepted: 5 October 2019

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 the performance of the maximum (k,r)-core computation. We also devise effective search orders for two algorithms with different search priorities for vertices. Besides, we study the diversified (k,r)-core search problem to find l maximal (k,r)-cores which cover the most vertices in total. These maximal (k,r)-cores are distinctive and informationally rich. An efficient algorithm is proposed with a guaranteed approximation ratio. We design a tight upper bound to prune unpromising partial (k,r)-cores. A new search order is designed to speed up the search. Initial candidates with large size are generated to further enhance the pruning power. Comprehensive experiments on real-life data demonstrate that the maximal (k,r)-cores enable us to find interesting cohesive subgraphs, and performance of three mining algorithms is effectively improved by all the proposed techniques.

Keywords Community detection · User engagement · User similarity · Diversification

1 Introduction

Nowadays, data become 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 fundamental graph problem which aims to find groups of well-connected nodes (e.g., people), and a variety of models have been proposed such as clique [14], k-core [45] and k-truss [31]. Most of the existing work only consider the structural cohesiveness of the subgraphs. However, in practice, we usually need to consider both structure and attribute perspectives for cohesive subgraph mining. 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*.

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., [6,22,35,40,55]); that is, ensure there are a considerable num-

✉ Fan Zhang
fanzhang.cs@gmail.com

Xuemin Lin
lxue@cse.unsw.edu.au

Ying Zhang
ying.zhang@uts.edu.au

Lu Qin
lu.qin@uts.edu.au

Wenjie Zhang
zhangw@cse.unsw.edu.au

1 Guangzhou University, Guangzhou, China

2 University of New South Wales, Sydney, Australia

3 East China Normal University, Shanghai, China

4 Centre for AI, University of Technology Sydney, Sydney, Australia

Published online: 26 October 2019

ber of friends for each individual user (vertex) in the group (subgraph). In [6], 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., [7,28,30,47]). The constraint of pairwise similarity is often used in social studies when similarity requirement is nontrivial [25,28,42,61]. 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 [19]. 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 compute the maximal (k,r)-cores.

Example 1 In Fig. 1, we use a set of keywords to describe the interests of each user. Jaccard similarity metric can be employed to measure the user similarity based on user keywords. Suppose $k = 3$ and the similarity threshold $r = 0.5$, the group within red circle is a maximal (k,r)-core. In this group, each user has at least three friends and their interests are similar to others. This group is likely to remain stable, become active and enhance the interactions among users.

Applications Online group activities are already very common in many social networks, e.g., Google has indexed approximately 620 million Facebook groups in 2010 [46].

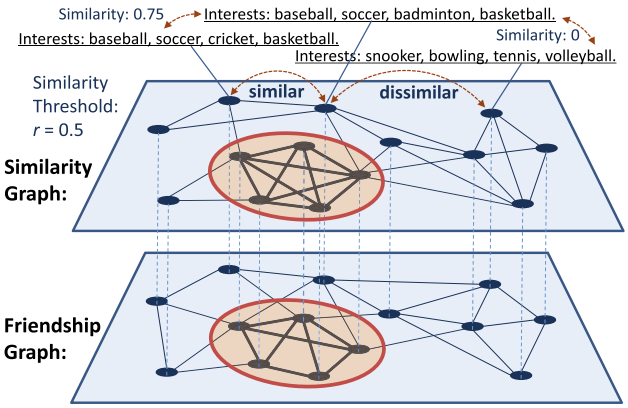


Fig. 1 Group by friendship and interests. When $k = 3$ and $r = 0.5$, the circled group is a maximal (k,r)-core where each vertex has at least 3 neighbors and is similar to each other

The (k,r)-core model can well-serve the discovery of social groups where the users are *highly similar* in terms of interest, location, etc. Some potential applications are as follows.

Interest-based social groups The official information from Facebook [19] shows that the criteria used in their group recommendation system are based on user friendship and high similarity of user attributes. Facebook recommends a group A to a user u if (i) the friends of u are members of the group, and (ii) the attributes of u are highly similar to the group A, i.e., u is highly similar to existing group members. The similarity is generally measured by the pages liked by u, the tags of the group A, the spatial closeness, etc [19]. These requirements well match the modeling of (k,r)-core. The system only recommends existing user groups, while the (k,r)-core algorithms can help find new promising groups.

Gambling groups in mobile payment transactions A large number of payment transactions are now conducted online, including the money involved in gambling. It is common that gamblers in the same group frequently transfer money to a certain number of members, which forms the network structure. Each user in the network has many attributes, e.g., inflow amount, outflow amount, the number of transfers, the time of each transfer, the location of each transfer, etc. The gamblers are connected and highly similar in terms of transfer amount, the time, the location, etc. Our (k,r)-core model can be used to detect the suspicious groups, with a particular similarity function designed for this scenario.

Group buying in social marketing Online group buying services, such as Groupon and Pingduoduo, have become highly popular and successful [38]. The service allows users with *similar* purchase interests to congregate online as a group to obtain group discounts [37]. In addition, the companies are suggested to encourage *social interactions* among users because social influence and group consumption can amplify the deal popularity [39]. The potential buying groups may be

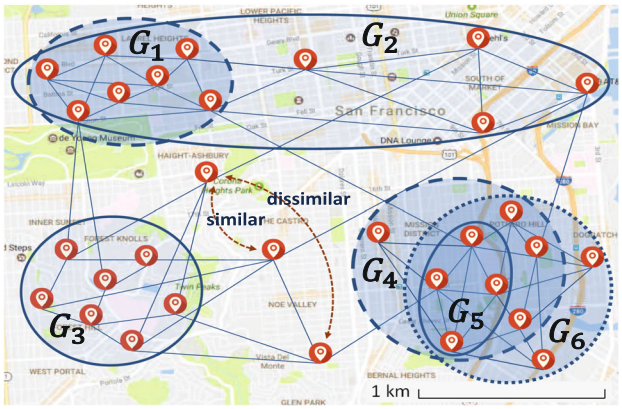


Fig. 2 Group by friendship and locations. Two users are similar if their spatial distance is not larger than r. When $k = 3$ and $r = 1$km, (i) the groups satisfying structure constraint: G_1, G_2, G_4, G_5 and G_6; (ii) the groups satisfying similarity (spatial-closeness) constraint: G_1, G_3, G_4, G_5 and G_6; and (iii) the maximal groups satisfying both constraints (maximal (k,r)-cores): G_1, G_4 and G_6

found by our (k,r)-core model, where proper structure and similarity thresholds should be applied to fit the marketing requirement.

Location-based games Pokemon Go is one popular mobile game with millions of active users worldwide [48], based on physical locations of users. Its social system was launched in 2018 to improve user experience [43]. The developers of Pokemon Go wish to encourage the users to play together and connect meaningfully in-game [43]. The strict spatial closeness is required when users wish to form a real team and play the game nearby. By recommending game teams with potential to become sustainable and active, the companies can greatly enhance user stickiness and improve game experience [12].

Example 2 As illustrated in Fig. 2, we can model the users and their friendship as a graph, where each user is located on the map. We say two users are similar if their spatial distance is not larger than r. Suppose $k = 3$ and the similarity (distance) threshold $r = 1$km, G_2 and G_3 are not good candidate groups. Although each user pair in G_3 has close distance, their friendship is weak. Likewise, although each user in G_2 has at least k friends, some users cannot conveniently play with others because they are far away from others. However, maximal (k,r)-cores (i.e., G_1, G_4 and G_6) can effectively identify good candidate groups satisfying both structure and similarity constraints. Note that although G_5 is a (k,r)-core, it is less interesting because it is fully contained by two larger groups, G_4 and G_6.

The studied problems (a) The enumeration of maximal (k,r)-core is a fundamental procedure to find and analyze all the high-quality groups for network holders, e.g., all the suspicious groups involved in gambling. Our enumeration framework is a base for computing the large or diversified maximal (k,r)-cores. (b) The groups with larger size are more likely to attract attention from the network holders and users, for potentially higher impact and influence. Thus, we compute the maximum (k,r)-core which is the (k,r)-core with the largest number of vertices. The proposed algorithm can also be applied to find the top-l largest maximal (k,r)-cores. In gambling group detection, the large transaction data may lead to frequent computation of (k,r)-cores. The manual verification should examine the large suspicious groups first. (c) Although each top-l maximal (k,r)-core is individually large, there may be many common users who join more than one of the top-l. The overlaps in some results motivate us to search the diversified top-l maximal (k,r)-cores (DivKRCs in short) which cover the largest number of vertices in the graph. DivKRCs are more interesting than the top-l because they are more diverse and informationally rich, e.g., they contain the largest number of users involved in different gambling groups.

Example 3 In Fig. 2, we set $k = 3$. The enumeration returns 3 maximal (k,r)-cores G_1, G_4 and G_6. The maximum (k,r)-core is G_6 with nine vertices. The top-2 largest maximal (k,r)-cores G_4 and G_6 are highly overlapped, with seven common users in both sides. The diversified top-2 maximal (k,r)-cores are G_1 and G_6.

Challenges and contributions Although there is a linear algorithm for k-core computation [5] (i.e., only consider structure constraint), we prove the (k,r)-core problems are all NP-hard because of the similarity constraint involved.

The enumeration 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 Sect. 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 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 largely reducing the search space by proposing effective techniques for pruning, early termination and maximal check. A good search order is designed to speed up the enumeration.

The maximum To find the maximum (k,r)-core, we follow the framework of enumeration and aim to further reduce the search space by pruning unpromising branches. The (k,k')-core based approach is designed to fast compute a tight size upper bound of the maximal (k,r)-core in a search branch. The upper bound significantly prunes the branches which produce small (k,r)-cores. A different search order is designed for finding the maximum.

The diversification In order to find the top-l DivKRCs, a straightforward algorithm is to firstly enumerate all maximal (k,r)-cores and then apply a greedy maximum coverage algorithm [24] to find the top-l DivKRCs. However, it is not worthwhile and necessary to run the complete (k,r)-core enumeration procedure for computing the DivKRCs. Besides, in this solution, the computations of maximal (k,r)-cores and the maximum coverage are isolated. Consequently, this solution is not efficient enough for DivKRC search. We propose a more efficient algorithm to compute the DivKRCs without generating all maximal (k,r)-cores and with a guaranteed approximation ratio. The algorithm maintains at most l candidate maximal (k,r)-cores in the (k,r)-core enumeration procedure. The candidates are updated when better candidate maximal (k,r)-cores are found. 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, finding the maximum (k,r)-core and finding the diversified maximal (k,r)-cores are all NP-hard. (Sect. 2)
- We develop efficient algorithms to enumerate the maximal (k,r)-cores with candidate pruning, early termination and maximal checking techniques. (Sect. 5)
- We 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. (Sect. 6)
- We also develop an efficient algorithm to find the diversified top-l maximal (k,r)-cores, with a guaranteed approximation ratio. Particularly, a tight double k-core based upper bound is proposed to prune unpromising partial (k,r)-cores. Besides, initial candidate generation finds large-size initial maximal (k,r)-cores to enhance the pruning power of the upper bound and the designed search order. (Sect. 7)
- Based on some key observations, we propose five search orders for enumerating maximal (k,r)-cores, checking maximal (k,r)-cores, finding maximum (k,r)-core, finding diversified maximal (k,r)-cores and initial candidate generation, respectively. (Sect. 8)
- Our empirical studies on real-life data demonstrate that interesting cohesive subgraphs can be identified by maximal (k,r)-cores, maximum (k,r)-core and diversified top-l maximal (k,r)-cores. The extensive performance evaluation shows that the techniques proposed in this paper can greatly improve performance of three mining algorithms. (Sect. 9)

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
$\mathrm{sim}(u, v)$	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 $DP(u, C \cup E) = 0$
$G(M)$	The induced subgraph of M on G
$V(S)$	The vertex set of S
$\mathcal{E}(S)$	The edge set of S

2 Preliminaries

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 $\mathrm{sim}(u, v)$, we denote the similarity of two vertices u, v in $V(G)$ which is derived from their corresponding attribute values (e.g., users' geolocations and interests) such as Jaccard similarity or Euclidean distance. For a given similarity threshold r, we say two vertices are dissimilar (resp. similar) if $\mathrm{sim}(u, v) < r$ (resp. $\mathrm{sim}(u, v) \geq r$).[1]

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 $DP(S)$ to 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 $V(S)$. Table 1 summarizes the mathematical notations

[1] 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.

used throughout this paper. Now we formally introduce two constraints.

Definition 1 (*Structure constraint*) Given an 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.

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 $S' \in \mathcal{R}$.

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 fundamental problems: (i) enumerating all maximal (k,r)-cores in G; and (ii) finding the maximum (k,r)-core in G.

For conciseness, we defer the problem definition of diversified maximal (k,r)-core search to Sect. 7.

2.2 Problem complexity

We can compute the k-core in linear time [5], while the two problems studied in this paper are NP-hard due to the involvement of 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 The key idea is that we may construct a specific attributed graph which is a fully connected graph (i.e., structure constraint is always satisfied), and we may come up with the NP hardness based on the similarity graph by reducing to the maximal clique problem.

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., $\text{sim}(u, v) = \frac{|A(u) \cap A(v)|}{|A(u) \cup A(v)|}$ for any pair of vertices u and v in $V(G')$, and let the similarity threshold $r = \frac{1}{2|V(G')|}$. We have $\text{sim}(u, v) \geq r$ if the edge $(u, v) \in \mathcal{E}(G)$, and otherwise $\text{sim}(u, v) = 0 < r$. 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 [27]. □

Theorem 2 *There does not exist a polynomial delay or polynomial total algorithm for the problem of enumerating all maximal (k,r)-cores, unless P=NP.*

Proof First, according to Theorem 1, it is impossible to find a (k,r)-core in polynomial time of input size during enumeration unless P=NP. Consequently, there is no polynomial delay algorithm for maximal (k,r)-core enumeration. Second, suppose on the contrary that there is a polynomial total (total time polynomial to input + output size) algorithm for the maximal (k,r)-core enumeration problem, we can derive that when there is no (k,r)-core, the algorithm can terminate in time polynomial to input size. This contradicts Theorem 1. Therefore, Theorem 2 holds. □

3 The clique-based approach

Let G' denote a new graph named *similarity graph* with $V(G') = V(G)$ and $\mathcal{E}(G') = \{(u, v) \mid \text{sim}(u, v) \geq r \ \& \ u, v \in V(G)\}$, 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'.

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

Input : G : attributed graph, k : degree threshold, r : similarity threshold
Output : $\mathcal{R}$: Maximal (k,r)-cores

1 **for** each edge (u, v) in $\mathcal{E}(G)$ **do**
2 Remove edge (u, v) from G **If** $sim(u, v) < r$;
3 $\mathcal{S} \leftarrow$ **k-core**(G); $\mathcal{R} := \emptyset$;
4 **for** each connected subgraph S in $\mathcal{S}$ **do**
5 $\mathcal{R} := \mathcal{R} \cup$ **NaiveEnum**($\emptyset$, S);
6 **for** each (k,r)-core R in $\mathcal{R}$ **do**
7 **if** there is a (k,r)-core $R' \in \mathcal{R}$ s.t. $R \subset R'$ **then**
8 $\mathcal{R} := \mathcal{R} \setminus R$;
9 **return** $\mathcal{R}$

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

Input : M : chosen vertices, C : candidate vertices
Output : $\mathcal{R}$: (k,r)-cores

1 **if** $C = \emptyset$ and $deg_{min}(M) \geq k$ and $DP(M) = 0$ **then**
2 $\mathcal{R} := \mathcal{R} \cup R$ **for** every connected subgraph $R \in G(M)$;
3 **else**
4 $u \leftarrow$ choose a vertex in C;
5 **NaiveEnum**($M \cup u$, $C \setminus u$); /* **Expand** */;
6 **NaiveEnum**(M, $C \setminus u$); /* **Shrink** */;

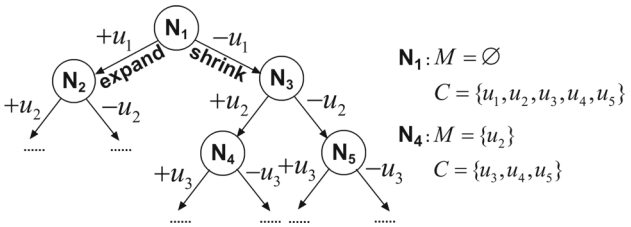


Fig. 3 Example of the search tree

– We only need to compute k-core for each maximal clique because any maximal (k,r)-core in a non-maximal clique can be obtained from a maximal clique.

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 is substantially outperformed by our baseline algorithm (Sect. 9.3), although the state-of-the-art k-core and clique computation methods have been applied [5,51]. This further validates the effectiveness to integrate computation of k-core and clique at each search step.

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. Line 6–8 eliminate the non-maximal (k, r)-cores by checking all (k,r)-cores.

During the NaiveEnum search procedure (Algorithm 2), the vertex set M incrementally retains the chosen vertices, and C retains the candidate vertices. In order to enumerate all possible solutions, for each chosen vertex u from C at Line 4, we will try to extend M with u (Line 5) or explicitly discard u (Line 6). Then, each subset $R \subseteq S$ is validated according to the structure, similarity and connectivity constraints at Line 1–2.

As shown in Fig. 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 1–2 of Algorithm 1, 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.

5 Finding all maximal (k,r)-cores

In this section, we propose pruning techniques for the enumeration. Note that we introduce search orders in Sect. 8.

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 3 (Structure-based pruning) *We can discard a vertex u in C if* $\deg(u, M \cup C) < k$.

Theorem 4 (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 4) 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 unless the search is terminated.

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 5 *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.

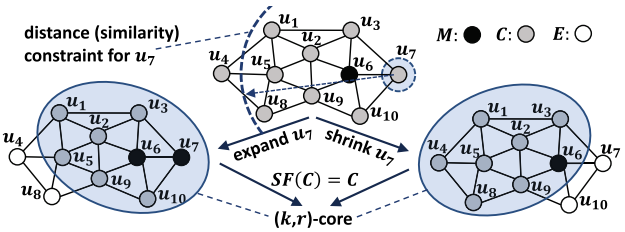


Fig. 4 Pruning and retaining candidates

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 (Eq. 2) of $M \cup C$. So $M \cup C$ 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 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. This validation rule will be used without further mention.

Example 4 In Fig. 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 $SF(C) = C$, 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 Sect. 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 6 (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 6 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 (Line 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.

Theorem 7 (Checking maximal) *Given a* (k,r)*-core* R, R *is a maximal* (k,r)*-core if there does not 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.*

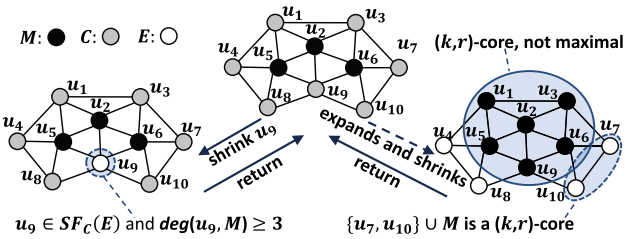


Fig. 5 Early termination and maximal check

Proof We have 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 Fig. 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 are 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 6. 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 is similar to our advanced enumeration algorithm, we defer the details to Sect. 5.4.

5.4 Advanced enumeration method

Algorithm 3 shows our advanced enumeration algorithm which integrates the techniques proposed in the previous sections. We first apply the candidate pruning algorithm outlined in Sect. 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 5, and we can conduct the maximal check (Line 3–5). Otherwise, Line 7–9 choose one vertex from $C \backslash SF(C)$ and continue the search following two branches.

Checking maximal algorithm According to Theorem 7, 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

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

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

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 G(M) is a (k,r)-core then
3     Exit the algorithm with isMax = false If |M*| < |M|;
4 else if |C| > 0 then
5     u ← a vertex in C;
6     CheckMaximal(M ∪ u, C \ u);
7     CheckMaximal(M, C \ u );
```

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 introduce the novel upper bound-based algorithm to find the maximum (k,r)-core.

6.1 Algorithm for finding the maximum

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 (Line 6–12), which is discussed in Sect. 8.

To find the maximum (k,r)-core in G, we need to replace the NaiveEnum procedure (Line 5) 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.

Time complexity As shown in Sect. 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 is same to 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 [26] and k-core-based methods [5] 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 $c_{\min}$ [26].

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 $|R| \leq k_{\max} + 1$.

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 := G(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);
```

Therefore, we may apply the existing k-core decomposition approach [5] to compute the maximal core number (i.e., $k_{\max}$) on the similarity subgraph J'.

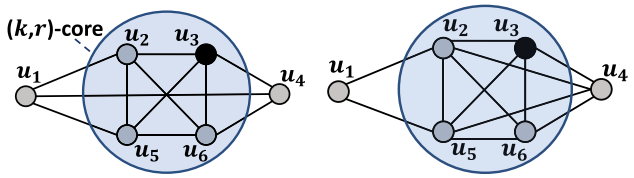
At the first glance, both the structure and similarity constraints are used in the above method because J itself is a k-core (structure constraint) and we consider the $k_{\max}$-core of J' (similarity constraint). The upper bound could be tighter by choosing the smaller one from color-based upper bound 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 8 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.

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 8 *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. □



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

Fig. 6 Upper bound examples, $k = 3$. The color-based upper bound is 5 because we need 5 colors to *color* J'. The k-core-based upper bound is 5 since $k_{\max}$ of J' is 4. The (k, k')-core-based upper bound is 4 because there is a non-empty (3, 3)-core and the (3, 4)-core is empty

If (k,k')-core exists in the graph (J, J'), it is impossible to color graph J' with k' or fewer colors, because there exists some vertex v of degree k' or higher. Thus, (k,k')-core-based approach provides tighter upper bound compared to color-based approach.

Example 6 In Fig. 6, we have $k = 3$, $M = \{u_3\}$, and $C = \{u_1, u_2, u_4, u_5, u_6\}$. Figure 6a shows the induced subgraph J from $M \cup C$ on G, and Fig. 6b 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 [5] on J' with additional update which ensures the corresponding subgraph on J is a k-core. Initially, we use $\deg(u)$ and $\deg_{\text{sim}}(u)$ 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. $NB_{\text{sim}}(u)$) 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 Line 9–11 (resp. Line 12–15). Note that we need to recursively remove vertices with degree smaller than k (Line 15) in the proce-

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 $M \cup C$ with increasing order of their similarity degrees;
2 **for** each $u \in H$ **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 $v \in NB_{sim}(u) \cap H$ **do**
10 **if** $deg_{sim}(v) > k'$ **then**
11 $deg_{sim}(v) := deg_{sim}(v) - 1$;
12 **for** each $v \in NB(u) \cap H$ **do**
13 $deg(v) := deg(v) - 1$;
14 **if** $deg(v) < k$ **then**
15 **KK'coreUpdate**(v, k', H);

dure. At Line 5, we need to reorder the vertices in H since their similarity degree values may be updated. According to Theorem 8, $k' + 1$ is returned at Line 6 as the upper bound of the maximum (k,r)-core size.

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 The top-*l* maximal (*k*,*r*)-cores

Besides the maximum (k,r)-core, social network service providers would also like to see the top-l maximal (k,r)-cores whose activeness reflects the hotness of the network. Users may be interested in these top-l communities which can be some representative and appealing groups in the network. To find the top-l maximal (k,r)-cores, we can record current top-l largest maximal (k,r)-cores in Algorithm 5. At Line 2, the size upper bound is compared with the size of the minimum maximal (k,r)-core recorded. At 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 "$\{R_j \mid j \in N^+ \ \& \ j \leq l\}$: the top-$l$ 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 $\{R_j \mid j \in N^+ \ \& \ j \leq l\}$. Line 4 should be replaced by "$min(R_j) := M \cup C$ If CheckMaximal(M, E);". Since there is no difference for Algorithm 6 toward finding the maximum and the top-l, the algorithm keeps same.

7 Finding diversified maximal (*k*,*r*)-cores

In this section, we first formally define the diversified maximal (k,r)-core (DivKRC) search problem. Then, we introduce a baseline algorithm based on (k,r)-core enumeration, followed by our advanced DivKRC search algorithm, including the basic framework, the double k-core upper bound and initial candidate generation. Note that the search order for DivKRC is introduced in Sect. 8.

7.1 Problem definition

We formally define the diversified top-l maximal (k,r)-core search problem in this section.

Definition 7 (*Coverage*) Given a set of maximal (k,r)-cores $\mathcal{D} = \{R_1, R_2, \ldots\}$ in G, the coverage of $\mathcal{D}$, denoted by $cov(\mathcal{D})$, is the set of vertices covered by the (k,r)-cores in $\mathcal{D}$, i.e., $cov(\mathcal{D}) = \cup_{R \in \mathcal{D}} V(R)$.

Problem statement Given an attributed graph G, an integer k, a similarity threshold r and an integer l, the diversified top-l maximal (k,r)-core search is to find a set $\mathcal{D}$, such that (1) each $R \in \mathcal{D}$ is a maximal (k,r)-core, (2) $|\mathcal{D}| \leq l$ and (3) $|cov(\mathcal{D})|$ is maximized. $\mathcal{D}$ is called diversified top-l maximal (k,r)-cores (DivKRCs).

When $l = 1$, the DivKRC search problem becomes the maximum (k,r)-core search problem which is NP-hard. Therefore, the DivKRC search problem is NP-hard.

7.2 Baseline algorithm

Benefit from the efficient (k,r)-core enumeration algorithm, we can retrieve the top-l DivKRCs with the maximum coverage from all the enumerated maximal (k,r)-cores. Essentially, it is the maximum coverage problem which finds l sets (l maximal (k,r)-cores) to cover the largest number of elements (vertices) from all the given sets (all the maximal (k,r)-cores).

The baseline algorithm is shown in Algorithm 7. At Line 1, it first computes all the maximal (k,r)-cores by the proposed enumeration algorithm in Sect. 5, where the advanced enumeration (Algorithm 3) and maximal check (Algorithm 4) are applied. Then, it computes the top-l DivKRCs using the

Algorithm 7: **CoverEnum**(G, k, r, l)

Input : G : attributed graph, k : degree threshold, r : similarity threshold, l : number limit of DivKRCs
Output : $\mathcal{D}$: the top-l DivKRCs

1 $\mathcal{R} :=$ **EnumerateMKRC**(G, k, r) with advanced enumeration (Algorithm 3) and maximal check (Algorithm 4);
2 $\mathcal{D} := \emptyset$; $V' := V(\mathcal{R})$;
3 **for** $i = 1$ to l **do**
4 $\quad R := \text{argmax}_{R' \in \mathcal{R}-\mathcal{D}}\{|V(R') \cap V'|\}$;
5 $\quad \mathcal{D} := \mathcal{D} \cup R$; $V' := V' \setminus V(R)$;
6 **return** $\mathcal{D}$

greedy algorithm for maximum coverage problem (Line 3–5). The maximum coverage problem is to select l sets from a collection of sets such that the union of l sets contains the largest number of elements. At Line 2, let $\mathcal{D}$ be the selected sets (maximal (k,r)-cores) and V' be the elements (vertices) not covered by $\mathcal{D}$. The algorithm greedily adds a maximal (k,r)-core to $\mathcal{D}$ which covers most vertices in V' (Line 4). Then, $\mathcal{D}$ and V' are updated accordingly (Line 5). The greedy coverage algorithm achieves an approximation ratio of $1 - 1/e$ which is the best possible approximation of a polynomial time algorithm for the maximum coverage problem as shown in [24].

The major limitation of the baseline algorithm is the isolated computations of maximal (k,r)-cores and the top-l DivKRCs. Although the baseline can achieve a good approximation ratio, it requires the complete maximal (k,r)-core generation which indicates that the baseline cannot be faster than (k,r)-core enumeration. Besides, the baseline keeps all the maximal (k,r)-cores in memory. The number might be exponential in some special cases such as the input graph is a complete graph in structure. To overcome above limitations, we can maintain the top-l candidates in the enumeration procedure to prune unpromising search space and greatly speed up the DivKRC computation.

7.3 Advanced DivKRC search

7.3.1 Basic algorithm

We define the private coverage of a maximal (k,r)-core and the min-cover (k,r)-core in $\mathcal{D}$ as follows.

Definition 8 (*Private coverage*) $pcov(R, \mathcal{D})$. Given a set of maximal (k,r)-cores $\mathcal{D} = \{R_1, R_2, \ldots\}$, for each $R \in \mathcal{D}$, the private coverage of R in $\mathcal{D}$, denoted by $pcov(R, \mathcal{D})$, is the set of vertices in R that are not covered by the other maximal (k,r)-cores in $\mathcal{D}$, i.e., $pcov(R, \mathcal{D}) = V(R) \backslash cov(\mathcal{D} \backslash R)$.

Definition 9 (*Min-cover (k,r)-core*) $R_{\min}(\mathcal{D})$. Given a set of maximal (k,r)-cores $\mathcal{D} = \{R_1, R_2, \ldots\}$, the min-cover ($k$,$r$)-core of $\mathcal{D}$, denoted by $R_{\min}(\mathcal{D})$, is the maximal (k,r)-core $R \in \mathcal{D}$ with the smallest $pcov(R, \mathcal{D})$, i.e., $R_{\min}(\mathcal{D}) = \text{argmin}_{R \in \mathcal{D}} |pcov(R, \mathcal{D})|$.

Algorithm 8: **DivBasic**(G, k, r, l)

Input : G : attributed graph, k : degree threshold, r : similarity threshold, l : number limit of DivKRCs
Output : $\mathcal{D}$: the top-l DivKRCs

1 **for** each edge (u, v) in $\mathcal{E}(G)$ **do**
2 $\quad$ Remove edge (u, v) from G **If** $sim(u, v) < r$;
3 $\mathcal{S} \leftarrow$ **k-core**(G); $\mathcal{D} := \emptyset$;
4 **for** each connected subgraph S in $\mathcal{S}$ **do**
5 $\quad$ **DivEnum**($\emptyset, S, \emptyset$);
6 **return** $\mathcal{D}$

7 **DivEnum**(M, C, E)
8 Update C and E based on candidate pruning techniques (Theorem 3 and Theorem 4);
9 **if** $|\mathcal{D}| < l$ or $KRCoreSizeUB(M, C) > |pcov(R_{min}(\mathcal{D}), \mathcal{D})| + \frac{cov(\mathcal{D})}{l}$ **then**
10 $\quad$ **if** $C = SF(C)$ **then**
11 $\quad\quad M := M \cup C$;
12 $\quad\quad$ **CandidateUpdate**(R) **for** every maximal $R \in G(M)$;
13 $\quad$ **else**
14 $\quad\quad u \leftarrow$ choose a vertex in $C \setminus SF(C)$;
15 $\quad\quad$ **if** Expansion is preferred **then**
16 $\quad\quad\quad$ **DivEnum**($M \cup u, C \setminus u, E$);
17 $\quad\quad\quad$ **DivEnum**($M, C \setminus u, E \cup u$);
18 $\quad\quad$ **else**
19 $\quad\quad\quad$ **DivEnum**($M, C \setminus u, E \cup u$);
20 $\quad\quad\quad$ **DivEnum**($M \cup u, C \setminus u, E$);

21 **CandidateUpdate**(R)
22 **if** $|\mathcal{D}| < l$ **then**
23 $\quad \mathcal{D} := \mathcal{D} \cup \{R\}$; **Return**;
24 $\mathcal{D}' := (\mathcal{D} \setminus \{R_{min}(\mathcal{D})\}) \cup \{R\}$;
25 **if** $|pcov(R, \mathcal{D}')| > |pcov(R_{min}(\mathcal{D}), \mathcal{D})| + \frac{cov(\mathcal{D})}{l}$ **then**
26 $\quad \mathcal{D} := \mathcal{D}'$;

The basic algorithm for DivKRC search is shown in Algorithm 8. It firstly conducts graph reduction by edge removing (Line 1–2) and k-core computation (Line 3). Then, it invokes DivEnum procedure (Line 4–5) to find DivKRCs based on the framework of (k,r)-core enumeration.

At Line 8, DivEnum updates candidate vertex set C and excluded vertex set E according to Theorem 3 and Theorem 4. At Line 9, the (k,r)-core size upper bound technique (Algorithm 6) is applied to see whether the search branch is still promising. To continue the recursion, we require the (k,r)-core size upper bound of current $M \cup C$ to be larger than the minimum size of private coverage of a maximal (k,r)-core in $\mathcal{D}$ plus $\frac{cov(\mathcal{D})}{l}$, i.e., $|pcov(R_{\min}(\mathcal{D}), \mathcal{D})| + \frac{cov(\mathcal{D})}{l}$. It is the update threshold for a new (k,r)-core to replace an existing (k,r)-core in $\mathcal{D}$. If the condition in Line 9 is not satisfied, the search branch can be pruned because every (k,r)-core generated by $M \cup C$ cannot change current candidate set $\mathcal{D}$ as shown in CandidateUpdate procedure.

When every vertex in C is similar to every other vertex in C (Line 10), the induced subgraph of $M \cup C$ becomes a (k,r)-core. At Line 12, we check every maximal (k,r)-core from $M \cup C$ in CandidateUpdate procedure. The maximality of a (k,r)-core can be fast checked by Algorithm 4. When there is a vertex in C not similar to another vertex in C (Line 13), we select a vertex u to continue the recursion by expansion or shrinking. At Line 14, the vertex selection in the basic algorithm is same to that of the finding the maximum (k,r)-core (Algorithm 5 and Sect. 8.2). According to the scores of u in vertex selection, we decide the preference of expansion (Line 16–17) or shrinking (Line 19–20).

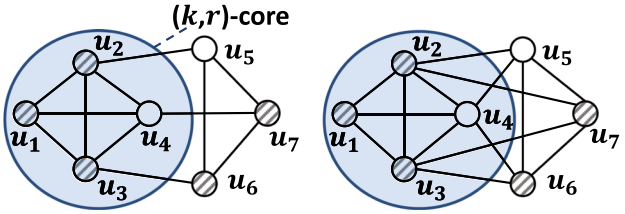
At Line 22–23, CandidateUpdate pushes the (k,r)-core R into the candidate set $\mathcal{D}$ when the candidate number in $\mathcal{D}$ is less than l. The (k,r)-core R can replace a candidate in $\mathcal{D}$ if R satisfies the update condition in Line 25.

Algorithm analysis Because the basic algorithm follows the framework of (k,r)-core enumeration, its time complexity is $O(T_{enum} + |\mathcal{R}| \cdot |R_{\max}|)$ where T_{enum} is the time complexity for (k,r)-core enumeration, $\mathcal{R}$ is the set of all the maximal (k,r)-cores, and $R_{\max}$ is the largest (k,r)-core in $\mathcal{R}$. The candidate update is dominated by (k,r)-core search in both time complexity and runtime. The result quality of Algorithm 8 can be guaranteed as Theorem 9 shows.

Theorem 9 *Given the graph G and an integer l, suppose $\mathcal{D}^*$ is the optimal diversified top-l maximal (k,r)-cores, and $\mathcal{D}$ is the result returned by Algorithm 8, we have* $|cov(\mathcal{D})| \geq 0.25 \times |cov(\mathcal{D}^*)|$.

Proof The correctness is based on a theoretical result from [3], which shows that: Given a stream of sets $\mathcal{A} = \{R_1, R_2, \ldots\}$ and an integer l, let $\mathcal{A}_i = \{R_1, R_2, \ldots, R_i\}$ and $\mathcal{D}_i = \mathcal{A}_i$ for $0 < i \leq l$. For any $i > l$, we construct $\mathcal{D}_i$ from $\mathcal{D}_{i-1}$ as follows: let $\mathcal{D}'_i = (\mathcal{D}_{i-1} \backslash \{R_{\min}(\mathcal{D}_{i-1})\}) \cup \{R_i\}$, we have **(1)** $\mathcal{D}_i = \mathcal{D}'_i$ if $|cov(\mathcal{D}'_i)| > (1 + \frac{1}{l})|cov(\mathcal{D}_{i-1})|$, and **(2)** $\mathcal{D}_i = \mathcal{D}_{i-1}$ otherwise. It is guaranteed that $\mathcal{D}_i$ is a 0.25-approximation solution of the maximum coverage problem of l sets on $\mathcal{A}$.

Then, we prove that the result of Algorithm 8 satisfies the above setting. Firstly, we show the condition in Line 25 of Algorithm 8 is same to $|cov(\mathcal{D}')| > (1 + \frac{1}{l})|cov(\mathcal{D})|$ in [3]. Let $R_{\min} = R_{\min}(\mathcal{D})$ and $\mathcal{D}^- = \mathcal{D} \backslash R_{\min}$, we also have $\mathcal{D}^- = \mathcal{D}' \backslash R$ because $\mathcal{D}' \backslash R = \mathcal{D} \backslash R_{\min}$. Then, $|pcov(R, \mathcal{D}')| = |cov(\mathcal{D}')| - |cov(\mathcal{D}^-)|$, and $|pcov(R_{\min}, \mathcal{D})| = |cov(\mathcal{D})| - |cov(\mathcal{D}^-)|$. So the two conditions are same. Then, we show the upper bound in Line 8 of Algorithm 8 does not violate the condition in [3]. The upper bound prunes the search branches when the (k,r)-cores generated by these branches cannot satisfy rule **(1)** in the condition, and thus all the pruned (k,r)-cores follow rule **(2)** which will not change $\mathcal{D}$. □



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

Fig. 7 Double k-core upper bound examples, $k = 3$. The shaded vertices in the figure are the vertices covered by $\mathcal{D} \backslash R_{\min}$. The (k,k')-core-based upper bound is 5 due to the $(3, 4)$-core induced by $G \backslash \{u_1\}$. The double k-core-based upper bound is the private coverage of the double 3-core (the whole vertex set in this case), which is 2

7.3.2 Upper bound of private coverage

The upper bound used in Line 9 of Algorithm 8 is to prune the search branches where any (k,r)-core produced in $M \cup C$ cannot trigger Line 26 and thus cannot replace the $R_{\min}(\mathcal{D})$ in candidate set $\mathcal{D}$. Currently, we use the upper bound UB_{R^*}, which is the size upper bound of the largest (k,r)-core (R^*) in the candidate set $M \cup C$, obtained by Algorithm 6. UB_{R^*} is a correct upper bound of the largest possible private coverage of a (k,r)-core R in $\mathcal{D}'$, where $\mathcal{D}' = (\mathcal{D} \backslash \{R_{\min}(\mathcal{D})\}) \cup \{R\}$ and R is computed from $M \cup C$. However, UB_{R^*} may not be tight when there are few private vertices in a (k,r)-core of $M \cup C$. The upper bound can be largely tighten if we compute the largest possible number of private vertices for a (k,r)-core R in $M \cup C$ toward $\mathcal{D}''$, where $\mathcal{D}'' = \mathcal{D} \backslash \{R_{\min}(\mathcal{D})\}$. Note that $pcov(R, \mathcal{D}') = pcov(R, \mathcal{D}'')$ for any R.

A straightforward idea is to count the number of private vertices in a set V^* toward $\mathcal{D}''$, where V^* fully contains $V(R^*)$, where R^* is the largest (k,r)-core in $M \cup C$. However, it is infeasible because the number of private vertices in V^* is not certainly the largest among that of every (k,r)-core R in $M \cup C$. So we have to find a vertex set V^{ub} which fully contains every R in $M \cup C$ to count the largest possible number of private vertices.

Note that although UB_{R^*} is at least the size of R^*, the corresponding $(k, k'_{\max})$-core of UB_{R^*} in Algorithm 6 is not a V^{ub} and does not certainly contain every R in $M \cup C$. The UB_{R^*} is the number $k'_{\max} + 1$ from the $(k, k'_{\max})$-core where the $(k, k'_{\max})$-core may not contain the R^*, as shown in Example 7. Thus the $(k, k'_{\max})$-core is not a V^{ub} and its private coverage toward $\mathcal{D}''$ cannot be used to compute an upper bound of private coverage for every R in $M \cup C$.

Example 7 In Fig. 7, we have a friendship graph J and the corresponding similarity graph J'. We can compute the (k,k')-core-based upper bound (i.e., $k'_{\max} + 1$) by running Algorithm 6. When $k = 3$, it is to compute the $(3, k')$-core with the largest k', i.e., $k'_{\max}$. The result value of $k'_{\max}$ is 4 because there is a $(3, 4)$-core $\{u_2, u_3, \ldots, u_7\}$ and there is no

(3, 5)-core. We can see that the (3, 4)-core does not contain the (k,r)-core $\{u_1, u_2, u_3, u_4\}$. Thus the private coverage of (3, 4)-core cannot be utilized in computing a better upper bound, and the (k,k')-core-based upper bound for DivKRC here can only be 5.

Double k-core-based upper bound We define the concept of double k-core which contains every R in $M \cup C$. Theorem 10 shows that we can derive the upper bound of private coverage for any possible (k,r)-core R based on the double k-core. Given a vertex set U and a graph J, let $J(U)$ denote the induced subgraph by U on J.

Definition 10 (*Double k-core*) Given a set of vertices U, the friendship graph J and the corresponding similarity graph J'. U is a double k-core of J and J' if (1) $\deg_{\min}(J(U)) \geq k$, (2) $\deg_{\min}(J'(U)) \geq k$ and (3) there is no double k-core U' of J and J' such that $U \subset U'$.

Theorem 10 *Given a graph J, the corresponding similarity graph J', the double k-core U on J and J', the candidate (k,r)-core set $\mathcal{D}$, let $R_{\min} = R_{\min}(\mathcal{D})$ if $|\mathcal{D}| \geq l$, and $R_{\min} = \emptyset$ otherwise. We have $|pcov(J(U), \mathcal{D}')| \geq |pcov(R, \mathcal{D}')|$ for every (k,r)-core R in J and J', where $\mathcal{D}' = (\mathcal{D} \setminus R_{\min}) \cup \{R\}$.*

Proof Let C be the vertex set of an arbitrary (k,r)-core R in J and J', we prove that $C \subseteq U$. We have $J(C)$ is a k-core and $J'(C)$ is a clique. Since $\deg_{\min}(J(C)) \geq k$, the size of C is at least $k+1$. Then, the $J'(C)$ is also a k-core because $J'(C)$ is a clique containing at least $k+1$ vertices. Suppose there is a non-empty set $W = C \setminus U$, we have $\deg_{\min}(J(U \cup W)) \geq k$ and $\deg_{\min}(J'(U \cup W)) \geq k$ because every vertex $w \in W$ has at least k neighbors in $U \cup W$. Then, $U \cup W$ is the double k-core of J and J', which contradicts with that U is the double k-core of J and J'. So there is no such non-empty set W and $C \subseteq U$. Consequently, we have $|pcov(J(U), \mathcal{D}')| \geq |pcov(R, \mathcal{D}')|$ because $V(R) = C \subseteq U$ for every R in J and J'. □

Example 8 In Fig. 7, we have a friendship graph J and the corresponding similarity graph J'. We have $R_{\min} = R_{\min}(\mathcal{D})$ if $|\mathcal{D}| \geq l$, and $R_{\min} = \emptyset$ otherwise. The shaded vertices in the figure are the vertices covered by $\mathcal{D} \setminus R_{\min}$. When $k = 3$, the full vertex set in the figure is already the double 3-core. Since the double k-core-based upper bound is the private coverage of the double 3-core, the upper bound is 2. As shown in Example 7, the (k,k')-core-based upper bound is 5. So the double k-core-based upper bound is better than (k,k')-core-based upper bound in this example.

When the private vertices in current candidate (k,r)-core set $\mathcal{D}$ are very few or none, the double k-core upper bound may be less effective than the (k,k')-core-based upper bound.

Algorithm 9: **DoubleKcoreBound**(M, C, D)

Input : M : vertices chosen, C : candidate vertices, $\mathcal{D}$: l candidate (k,r)-cores
Output : UB_{pri} : the upper bound for private coverage

```
1  U := M ∪ C;
2  G' is the similarity graph of G;
3  deg(u) := deg(u, G(U)) for every u ∈ U;
   deg_sim(u) := deg_sim(u, G'(U)) for every u ∈ U;
4  for each u ∈ U with deg(u) < k or deg_sim(u) < k do
5      Remove u from U;
6      for each v ∈ NB(u) ∩ U do
7          deg(v) := deg(v) − 1;
8      for each v ∈ NB_sim(u) ∩ U do
9          deg_sim(v) := deg(v)_sim − 1;
10 if |D| ≥ l then R_min := R_min(D); else R_min := ∅;
11 D' := (D \ {R_min}) ∪ {G(U)};
12 return |pcov(G(U), D')|
```

With the update of candidate (k,r)-core set, the private vertices may largely increase and then the double k-core upper bound becomes more effective.

7.3.3 Algorithm for double *k*-core-based upper bound

Algorithm 9 shows the details of double k-core- based upper bound computation. Initially, we use $\deg(u)$ and $\deg_{\text{sim}}(u)$ to denote the degree and similarity degree (i.e., the number of similar pairs containing u) of u w.r.t $M \cup C$, respectively. Meanwhile, $NB(u)$ (resp. $NB_{\text{sim}}(u)$) denotes the set of adjacent (resp. similar) vertices of u. We delete every vertex which violates the definition of double k-core at Line 5 and update the degrees of its neighbors in G (Line 6–7) and G' (Line 8–9). The upper bound is the private coverage of $G(U)$ in $\mathcal{D}'$ which replaces the min-cover (k,r)-core in $\mathcal{D}$ by $G(U)$ (Line 10–12).

Algorithm analysis The time complexity of Algorithm 9 is $O(m + m' + n + |U| + |R_{\min}(\mathcal{D})|)$ where m (resp. m') is the number of edges in $G(M \cup C)$ (resp. $G'(M \cup C)$), n is the number of vertices in $M \cup C$, and $|U| + |R_{\min}(\mathcal{D})|$ is the complexity of private coverage computation. The correctness is guaranteed by Theorem 10. Algorithm 9 can be immediately applied by replacing the basic upper bound $KRCoreSizeUB(M, C)$ at Line 9 of Algorithm 8.

7.3.4 Initial candidate generation

In this section, we generate initial candidate (k,r)-cores $\mathcal{D}$ to enhance the pruning power of the proposed upper bound of private coverage. Good initial candidates can also enhance the effectiveness of search order which will be introduced in Sect. 8.5. Intuitively, each (k,r)-core in a good initial $\mathcal{D}$ should be large and has low overlap with other (k,r)-cores in $\mathcal{D}$.

Algorithm 10: **InitCandidate**(G, k, r, l)

```
Input  : G : a connected subgraph, k : degree threshold, r :
         similarity threshold, l : number limit of DivKRCs
Output : D : the initial l DivKRCs
1  D := ∅;
2  for each connected subgraph S in the reduced G do
3      D := EnumL(∅, S, ∅);
4  return top-l (k,r)-cores in D with maximum size
5  EnumL(M, C, E)
6  Return if M ∩ D ≠ ∅ or D update time ≥ α × l;
7  Update C and E based on candidate pruning techniques
   (Theorem 3 and Theorem 4);
8  if D < l or KRCoreSizeUB(M, C) > |R_min(D)| then
9      if C = SF(C) then
10         M := M ∪ C;
11         D := D \ {R_min(D)} ∪ {R} if |R| > |R_min(D)| and R is a
           maximal (k,r)-core in G(M);
12     else
13         u ← choose a vertex in C \ SF(C);
14         if Expansion is preferred then
15             EnumL(M ∪ u, C \ {u ∪ D}, E);
16             EnumL(M, C \ {u ∪ D}, E ∪ u);
17         else
18             EnumL(M, C \ {u ∪ D}, E ∪ u);
19             EnumL(M ∪ u, C \ {u ∪ D}, E);
```

The initial candidate computation is shown in Algorithm 10 which maintains l candidate maximal (k,r)-cores where any two (k,r)-cores do not overlap. It follows the framework of (k,r)-core enumeration on the graph after pruning dissimilar edges and non-k-core vertices (Line 2–3). The search branch is returned once $\mathcal{D}$ contains some vertices in M, or $\mathcal{D}$ is updated for not less than $\alpha \times l$ times (Line 6). The upper bound in Line 8 is computed by Algorithm 6 because all the vertices in $\mathcal{D}$ are private vertices. The smallest (k,r)-core in $\mathcal{D}$ is replaced by a new maximal (k,r)-core R if R is larger than it (Line 11).

Algorithm analysis Compared with the diversified maximal (k,r)-core search (Algorithm 8), Algorithm 10 generates only one maximal (k,r)-core for each vertex in the reduced graph. Thus the time complexity is not larger than Algorithm 8. By pruning all the branches when M has overlap with $\mathcal{D}$ and limiting the update time of $\mathcal{D}$, Algorithm 10 runs much faster than Algorithm 8. The initial candidate computation can be easily applied in the basic DivKRC search by inserting it between Line 3 and 4 in Algorithm 8.

8 Search order

Section 8.1 briefly introduces some important measurements that should be considered for an appropriate visiting order. Then, we investigate the visiting orders in five algorithms: finding the maximum (k,r)-core (Algorithm 5), advanced maximal (k,r)-core enumeration (Algorithm 3), maximal check (Algorithm 4), advanced diversified top-l maximal (k,r)-core search (Algorithm 8 equipped with Algorithm 9 and 10) and initial candidate generation (Algorithm 10).

8.1 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 the three problems studied 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 favor 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 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

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

  Note that we have $DP(u, M \cup C) = 0$ for every $u \in M$ according to the similarity invariant (Eq. 1).
- Δ_2: the change of the number of edges, where

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

  Recall that $|\mathcal{E}(V)|$ denotes the number of edges in the induced graph from the vertex 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., $M = \emptyset$).

8.2 Finding the maximum (*k,r*)-core

In the search process, 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 $C \backslash SF(C)$. 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 $C \backslash SF(C)$, and d_1 (resp. d_2) stands for the average degree of the vertices in J (resp. J').

8.3 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 Sect. 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 Sect. 8.2.

8.4 Checking maximal

The search order for checking maximals is rather different than the enumeration and maximum algorithms. Toward 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.

8.5 Finding diversified maximal (*k*,*r*)-cores

In the diversified top-l maximal (k,r)-core search, we aim to fast identify maximal (k,r)-cores with large overall coverage, which requires the identified (k,r)-cores to contain a large number of private vertices. As discussed in Sect. 8.2, choosing a vertex with the largest $\lambda\Delta_1 - \Delta_2$ score helps to quickly find a large size (k,r)-core. But this score fails to ensure a large size of private vertices in the searched (k,r)-core where the majority of vertices may be shared vertices with other (k,r)-cores in the candidate set $\mathcal{D}$.

In our implementation, to encourage the preference on large private vertices, we give incentives to the scores of a vertex where the updated C' excludes the $\mathcal{D}$ vertices from C. We propose the following incentive measurement to fulfill the above requirement.

– Δ_3: the change of number of shared vertices with $\mathcal{D}$, where

$$\Delta_3 = \frac{(C \backslash C') \cap \mathcal{D}}{C \backslash C'}. \tag{5}$$

Specifically, we use $\lambda\Delta_1 - \Delta_2 + \Delta_3$ to measure the potential of a branch for each vertex in $C \backslash SF(C)$, where λ is used to make the trade-off. In this way, each candidate vertex has two scores. The vertex with the highest score is chosen, and its branch with higher score is explored first. For time efficiency, we only explore vertices within two hops from the candidate vertex when we compute its Δ_1, Δ_2 and Δ_3 values. It takes $O(n_c \times (d_1^2 + d_2^2))$ time where n_c denote the number of vertices in $C \backslash SF(C)$ and d_1 (resp. d_2) stands for the average degree of the vertices in J (resp. J').

Toward the search order in initial candidate generation (Algorithm 10), since we require each candidate has no overlap with any other candidate, Δ_3 is always 0 for each vertex. We apply the score $\lambda\Delta_1 - \Delta_2$ as in Sect. 8.2 because the initial candidate generation has the same objective with finding the maximum (k,r)-core, that is to fast identify a (k,r)-core with large size.

9 Performance evaluation

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

9.1 Experimental setting

Algorithms In this paper, we implement and evaluate 7 baseline algorithms and 3 advanced algorithms, as described in Table 3. Since the naive method in Sect. 4 is extremely slow even on a small graph, we employ `BasEnum` and `BasMax`

Table 2 Summary of techniques

Technique	Description
CR	The candidate retaining technique (Theorem 5)
ET	The early termination technique (Theorem 6)
CM	The checking maximal technique (Theorem 7)
CK	A tighter upper bound from the color-based and the k-core-based upper bound. (Sect. 6.2)
UB	The (k,k')-core upper bound technique (Theorem 8)
UBd	The double k-core upper bound technique for DivKRC search (Theorem 10)
IN	The initial candidate generation technique for DivKRC search (Sect. 7.3.4)
SO	The best search order is applied (Sect. 8)

Table 3 Summary of algorithms

Algorithm	Description
SimCore	Given G and r, the algorithm firstly removes all the dissimilar edges (similarity less than r) in G and then computes the k-core on the new graph G
Clique+	The advanced clique-based algorithm proposed in Sect. 3, using the clique and k-core computation algorithms in [51] and [5], 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 3 and 4 in Sect. 5.1). The best search order (Δ_1-then-Δ_2, in Sect. 8.3) is applied
AdvEnum	AdvEnum = BasEnum+CR+ET+CM. The advanced enumeration algorithm proposed in Sect. 5.4 that applies all advanced pruning techniques including: candidate size reduction (Theorems 3, 4 and 5 in Sect. 5.1), early termination (Theorem 6 in Sect. 5.2) and checking maximals (Theorem 7 in Sect. 5.3). Moreover, the best search order is used (Δ_1-then-Δ_2, in Sect. 8.3)
BasMax	The algorithm proposed in Sect. 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 Sect. 8.2)
AdvMax	AdvMax = BasMax+UB. The advanced finding maximum (k,r)-core algorithm proposed in Sect. 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 Sect. 8.2)
TopLMax	The algorithm for finding the top-l maximal (k,r)-core with the largest sizes, proposed in Sect. 6.4 by utilizing the AdvMax algorithm
DivEnum	The baseline algorithm for DivKRC search proposed in Algorithm 7 which applies the greedy maximum coverage algorithm on all the enumerated maximal (k,r)-cores from AdvEnum
BasDiv	The basic DivKRC search proposed in Algorithm 8 where candidate retaining CR, early termination ET, maximal check CM, upper bound UB and the best search order for AdvMax ($\lambda\Delta_1 - \Delta_2$) are applied
AdvDiv	The advanced DivKRC search which equips Algorithm 8 with the double k-core upper bound UBd (Theorem 10 in Sect. 7.3.3), initial candidate generation IN (Sect. 7.3.4) and the best search order SO (Sect. 8.5)

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. We employ the top-l algorithm TopLMax and two baseline algorithms DivEnum and BasDiv for the problem of diversified top-l maximal (k,r)-core search.

In Table 2, we show the name for each technique. Table 3 summaries all the evaluated algorithms.

Datasets Five real datasets are used in our experiments. The original data of DBLP are from http://dblp.uni-trier.de, DBpedia is from [22], and the remaining three datasets are from http://snap.stanford.edu. For non-spatial data, we use *Weighted Jaccard Similarity* of the attribute sets of two vertices to measure their similarity. For spatial data, we use *Euclidean Distance* between the locations of two users to measure their similarity.

In DBLP, we consider each author as a vertex with attribute of counted "publication venues" list. There is an edge between an author pair iff they have at least one co-authored paper. In Pokec, we consider each user to be a vertex with personal interests. There is an edge between two users iff they are friends. In Gowalla and Brightkite, we consider each user as a vertex along with his/her check-in information. The graph structure is based on user friendship. An edge is removed if an incident user has no check-in. In DBpedia, each vertex represents an entity, and each edge represents the relationship between two entities. The attribute of each entity is a keyword set, extracted by the Stanford Analyzer and Lemmatizer. Table 4 shows the statistics of the datasets.

Parameters We use different settings of k and r, where k varies from 3 to 25. In Gowalla and Brightkite, the distance threshold r ranges from 1 km to 5000 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 is from top

Table 4 Statistics of datasets

Dataset	Nodes	Edges	d_{avg}	k_{max}
Brightkite	58,228	194,090	6.67	52
Gowalla	196,591	456,830	4.65	43
DBLP	1,566,919	6,461,300	8.25	118
Pokec	1,632,803	8,320,605	10.19	27
DBpedia	8,099,955	71,527,515	17.66	95

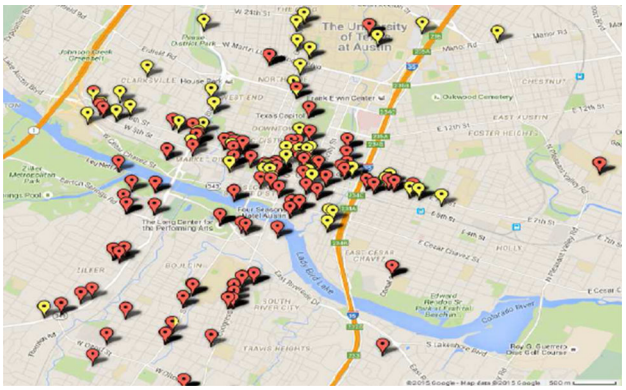


Fig. 8 Case study on Gowalla ($k=10$, $r=10$ km). Two groups (maximal (k,r)-cores) of users are marked by different colors

1‰ to top 15‰ (the similarity threshold value drops). In diversified top-l maximal (k,r)-core search, the default l is 10. Regarding the search orders of the `AdvMax`, `BasMax` and `AdvDiv` algorithms, we set λ to 5 by default. The α in initial candidate generation (`IN`) is set to 1 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.

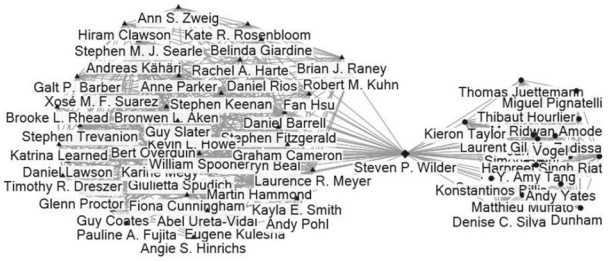
9.2 Effectiveness of AdvEnum and AdvMax

Compared to k-core, (k,r)-core enables us to find more valuable information with the additional similarity constraint.

Gowalla Figure 8 illustrates a set of `Gowalla` users who are from the same k-core when $k = 10$. By setting r to 10 km, two groups of users emerge, each of which is a maximal (k,r)-core, and we cannot identify them by structure constraint or similarity constraint alone. We observe that the maximum (k,r)-core in `Gowalla` always appears at Austin when $k \geq 6$. Then, we realize that this is because the headquarters of `Gowalla` is located in Austin.

DBLP Figure 9 shows cases of `DBLP` when $k = 15$ and r= 3‰.[2] In Fig. 9a, all authors come from the same k-core based on their co-authorship information alone (their structure constraint). While there are two (k,r)-cores with one common author Steven, 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 Steven from both sides. Figure 9b depicts the maximum (k,r)-core of `DBLP` with 49 authors. We find that they have intensively co-authored many papers related

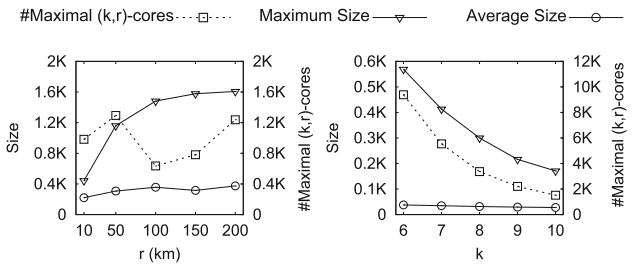
[2] To avoid the noise, we enforce that there are at least three co-authored papers between two connected authors in the case study.



(a) Two groups of users (two maximal (k,r)-cores) with one common user Steven from both groups.

(b) The largest group (the maximum (k,r)-core) where all the users engage in the Ensembl project.

Fig. 9 Case study on DBLP ($k=15$, $r=$top 3‰)



(a) Gowalla, k=5 **(b)** DBLP, r=top 3‰

Fig. 10 (k,r)-core statistics

to a project named *Ensembl* (http://www.ensembl.org/index.html), which is one of the well-known genome browsers.

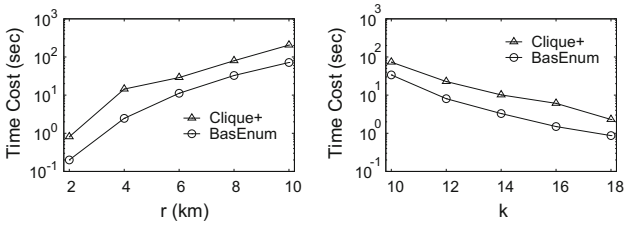
Statistics We also report the number of maximal (k,r)-cores, the average size and maximum size of (k,r)-cores on `Gowalla` and `DBLP`. Figure 10a and b shows that both maximum size of (k,r)-cores and the number of maximal (k,r)-cores are much more sensitive to the change of r or k on the two datasets, compared to the average size.

Let R denote a connected k-core subgraph returned by `SimCore`. Table 5 shows the average percentage of vertices in R similar to a vertex u in R, on different settings. We find the vertices in R may be dissimilar to many other vertices in R, because here we only enforce two users to be similar if they are structurally connected. Thus, the tightness of the similarity constraint in `SimCore` is dependent on the graph structure. Note that this percentage for any vertex in a (k,r)-core is always 100% due to the pairwise vertex similarity.

Table 5 Average percentage of similar vertices in the subgraph for 100 randomly selected vertices of top-5 largest maximal k-core subgraphs returned by SimCore

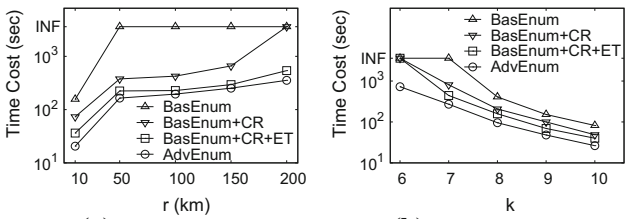
Setting	top-1	top-2	top-3	top-4	top-5
DBLP ($k=10$, $r=$top 3‰)	0.28 (857)	0.54 (245)	0.63 (189)	0.31 (126)	0.86 (115)
DBLP ($k=10$, $r=$top 5‰)	0.32 (1182)	0.61 (248)	0.65 (198)	0.28 (152)	0.54 (118)
Gowalla ($k=5$, $r=150$)	0.47 (2139)	0.98 (1671)	0.22 (1154)	0.7 (1113)	1 (840)
Gowalla ($k=5$, $r=300$)	0.38 (4043)	0.42 (2963)	0.28 (2062)	0.24 (984)	0.99 (877)

The sizes of subgraphs are shown in brackets



(a) Gowalla, k=5 **(b)** DBLP, r=top 3‰

Fig. 11 Evaluation of clique-based method



(a) Gowalla, k=5 **(b)** DBLP, r=top 3‰

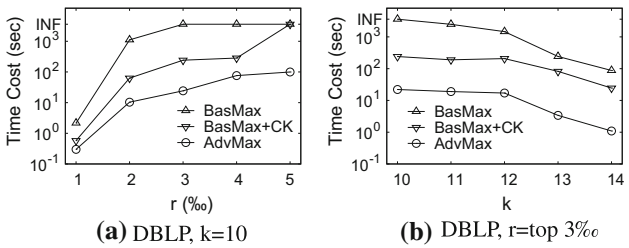
Fig. 12 Evaluation of pruning techniques

9.3 Efficiency of AdvEnum and AdvMax

In this section, we evaluate the efficiency of the techniques.

Evaluating the clique-based method In Fig. 11, 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 outperforms Clique+ by a stable margin because we apply pruning rules in BasEnum with the best search order and a large number 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 Figure 12 shows 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). Particularly, BasEnum + CR represents the BasEnum algorithm with candidate retaining



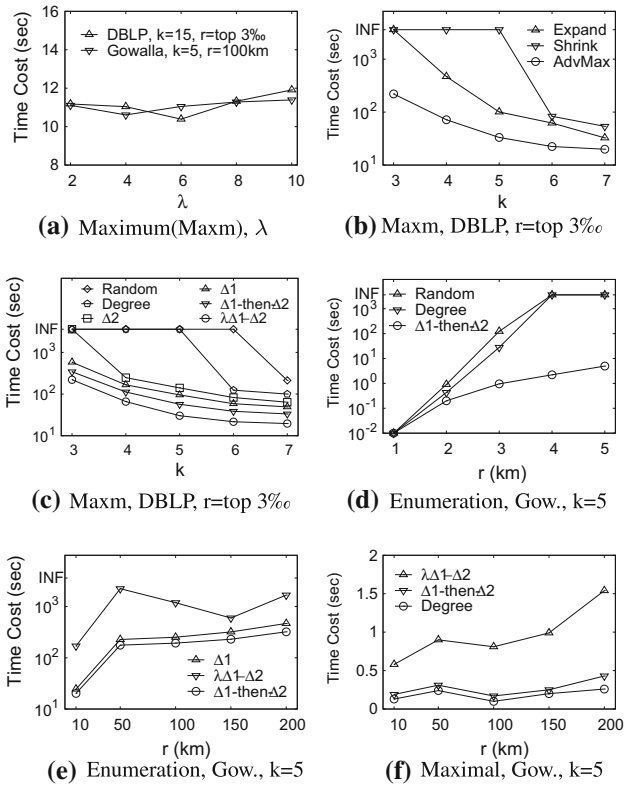
(a) DBLP, k=10 **(b)** DBLP, r=top 3‰

Fig. 13 Evaluation of upper bounds

technique (Theorem 5). Then BasEnum + CR + ET further includes the early termination technique (Theorem 6). By integrating the checking maximal technique (Theorem 7), it turns to be our AdvEnum algorithm. Note that the best search order is used for all algorithms. Among these techniques, Theorem 5 achieves the best speedup because the search on $SF(C)$ is skipped and $SF(C)$ may be large. The results in Fig. 12 confirm that all techniques contribute to enhancing the performance of AdvEnum.

Evaluating the upper bound techniques Figure 13 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 (Sect. 6.2) [5,26]. 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 (Sect. 8.2, Fig. 14a–c), enumeration algorithm (Sect. 8.3, Fig. 14d, e) and the checking maximal algorithm (Sect. 8.4, Fig. 14f). We first tune λ value for the search order of AdvMax in Fig. 14a against DBLP and Gowalla. In the following experiments, we set λ to 5 for maximum algorithms. Fig. 14b 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 Fig. 14c, we investigate a set of possible order strategies for AdvMax. As expected, the $\lambda\Delta_1 - \Delta_2$ order proposed in Sect. 8.2 outperforms

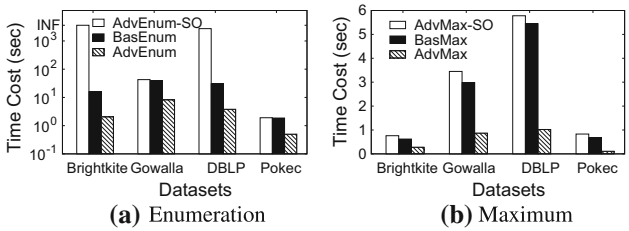


(a) Maximum(Maxm), λ (b) Maxm, DBLP, r=top 3‰

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

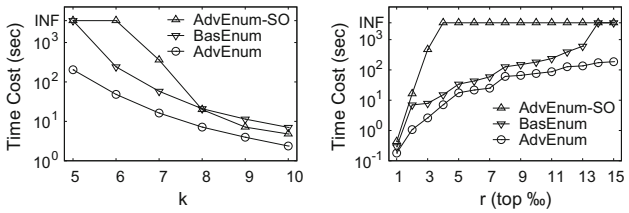
(e) Enumeration, Gow., k=5 (f) Maximal, Gow., k=5

Fig. 14 Evaluation of search orders



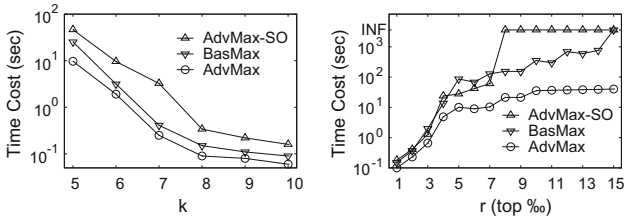
(a) Enumeration (b) Maximum

Fig. 15 Performance on four datasets



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

Fig. 16 Effect of k and r for enumeration



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

Fig. 17 Effect of k and r for maximum

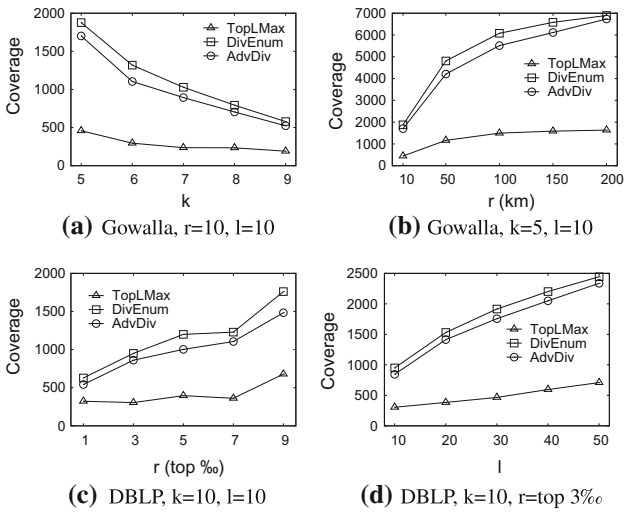
the other alternatives including random order, degree-based order (Sect. 8.4, used for checking maximals), Δ_1 order, Δ_2 order and Δ_1-then-Δ_2 order (Sect. 8.3, used by `AdvEnum`). Similarly, Fig. 14d and e confirms that the Δ_1-then-Δ_2 order is the best choice for `AdvEnum` compared to the alternatives. Figure 14f 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 15 evaluates the performance of the enumeration and maximum algorithms on four datasets with $k = 10$. We set r to 500 km, 300 km, 3‰ and 5‰ in `Brightkite`, `Gowalla`, `DBLP` and `Pokec`, respectively. 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 15a 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 Fig. 15b, where three algorithms are evaluated (`AdvMax-SO`, `BasMax`, and `AdvMax`).

Effect of k and r Figure 16 studies the impact of k and r for the three enumeration algorithms on `Gowalla` and `DBLP`. As expected, Fig. 16a shows that the time cost drops when k grows because many more vertices are pruned by the structure constraint. In Fig. 16b, 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 Fig. 17 for three maximum algorithms. Moreover, Figs. 16 and 17 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.

9.4 Effectiveness of AdvDiv

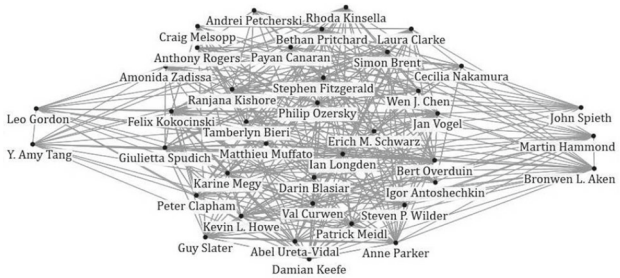
Coverage In Fig. 18, we report the number of vertices covered (coverage) by the result of `TopLMax`, `DivEnum` and `AdvDiv`, respectively, by varying the values of k, r and l. Figure 18a shows the coverage when we vary k, where the coverage basically becomes smaller with a larger k, because the structural constraint becomes tighter with a larger k. Figure 18b and c shows the coverage increases with a larger input of r. In Fig. 18d, the coverage increases when l becomes larger. We observe that the trends of `TopLMax` are not as good as that of `DivEnum` and `AdvDiv` with the larger



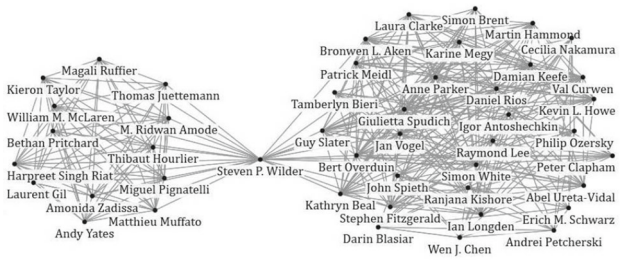
(a) Gowalla, r=10, l=10 **(b)** Gowalla, k=5, l=10

(c) DBLP, k=10, l=10 **(d)** DBLP, k=10, r=top 3‰

Fig. 18 Evaluation of vertex coverage



(a) TopLMax. The top-2 largest maximal (k,r)-cores are highly overlapped, where the common users are shown in the middle.

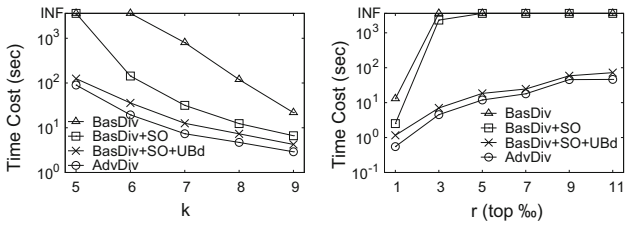


(b) AdvDiv. The diversified top-2 maximal (k,r)-cores have only one common user Steven from both side.

Fig. 19 Case study on DBLP ($k=10$, $r=$ top 1‰, $l=2$)

input values, which further validates the effectiveness of the DivKRC search. Although the coverage results of AdvDiv are slightly smaller than that of DivEnum, AdvDiv can outperform DivEnum on efficiency by more than 1 order of magnitude.

DBLP We conduct case studies on DBLP to show the effectiveness of diversified top-l maximal (k,r)-cores. Fig. 19a, b depicts the top-2 largest maximal (k,r)-cores by TopLMax



(a) Gowalla, r=100, l=10 **(b)** DBLP, k=10, l=10

Fig. 20 Evaluation of individual techniques

and the diversified top-2 maximal (k,r)-cores by AdvDiv, respectively, when $k = 10$ and $r =$ top 1‰.[3]

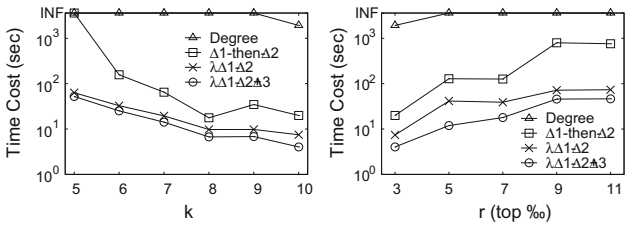
Figure 19a shows the two (k,r)-cores from TopLMax highly overlap where all the authors located at the middle are shared by the two (k,r)-cores. Besides the middle authors, one (k,r)-core further contains two authors (Leo and Y. Amy), and the other further contains three authors (John, Martin and Bronwen). The top-2 maximal (k,r)-cores from TopLMax are less interesting than the diversified top-2 from AdvDiv depicted in Fig. 19b. There is only one common author, Steven, in the diversified top-2 maximal (k,r)-cores. Specifically, one diversified (k,r)-core, denoted by $\mathcal{R}_l$, (resp. the other, denoted by $\mathcal{R}_r$) contains the left group of authors (resp. the right group) and Steven. We can see the $\mathcal{R}_r$ is similar to the (k,r)-cores in Fig. 19a, but the $\mathcal{R}_l$ is quite different to the result in Fig. 19a. The number of authors in Fig. 19b is also larger than that in Fig. 19a. Consequently, the result of AdvDiv is more informative and valuable than that of TopLMax.

9.5 Efficiency of AdvDiv

Evaluating the individual techniques Figure 20 shows the efficiency gain by incrementally equipping each proposed technique on BasDiv. Note that the best search order for AdvMax and the (k,k')-core upper bound from AdvMax are employed in BasDiv initially. BasDiv + SO denotes the BasDiv equipped with best search order (SO) for DivKRC search proposed in Sect. 8.5. BasDiv + SO + UBd further equips the advanced double k-core upper bound (UBd) designed in Sect. 7.3.3. After employing the initial candidate generation (IN), the algorithm becomes the advanced DivKRC search algorithm (AdvDiv). Figure 20 validates that all the proposed techniques improve the search efficiency, especially the double k-core upper bound and best search order techniques.

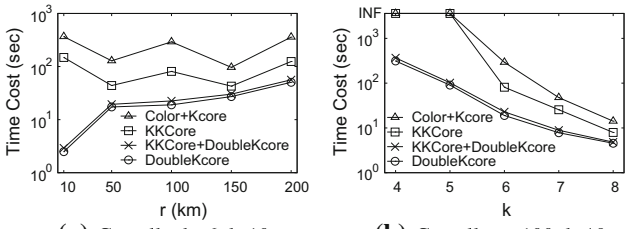
Evaluating the search orders Figure 21 shows the effect of employing different search orders in AdvDiv, includ-

[3] To make the figure clear, in this case study, one edge represents that there are at least three co-authored papers between two corresponding authors.



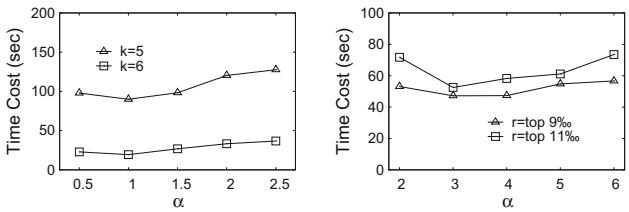
(a) DBLP, r=top 3‰, l=10 **(b)** DBLP, k=10, l=10

Fig. 21 Evaluation of search orders



(a) Gowalla, k=6, l=10 **(b)** Gowalla, r=100, l=10

Fig. 22 Evaluation of upper bounds



(a) Gowalla, r=100, l=10 **(b)** DBLP, k=10, l=10

Fig. 23 Evaluation of α in initial candidate generation

ing the degree in $M \cup C$-based order (Degree), the best order in AdvEnum (Δ_1-then-Δ_2), the best order in AdvMax ($\lambda\Delta_1 - \Delta_2$) and the best order for AdvDiv ($\lambda\Delta_1 - \Delta_2 + \Delta_3$). The other techniques for AdvDiv, such as the double k-core upper bound and initial candidate generation, are all equipped when comparing the search orders. In Fig. 21, the degree-based order is significantly outperformed by the other 3 orders. The AdvMax order is more effective than the AdvEnum order. The best order designed for AdvDiv further outperforms the other two best orders in AdvEnum and AdvMax, respectively, on different k and r values. It validates the effectiveness of encouraging the exclusion of shared vertices in $C \cap \mathcal{D}$ when we design the best search order for AdvDiv.

Evaluating the upper bounds Figure 22 reports the effect of employing different upper bound techniques in AdvDiv, including the color and k-core-based upper bound (Color+Kcore), the (k,k')-core-based upper bound (KKcore), the (k,k')-core and double k-core integrated upper bound (KKcore+DoubleKcore) and the double k-core upper bound (DoubleKcore) adopted for AdvDiv. The other techniques for AdvDiv, such as the best search order and initial candidate generation, are all equipped when comparing the upper bounds. Color+Kcore adopts the smaller one from the color-based upper bound and k-core-based upper bound (Sect. 6.2) [5,26]. KKcore is also proposed in Sect. 6.2 which performs better than Color+Kcore when evaluating the upper bounds in finding the maximum (k,r)-core. DoubleKcore is proposed in Sect. 7.3.3 which considers the private coverage status in the top-l candidates to further improve the upper bound technique for AdvDiv. KKcore+DoubleKcore adopts the smaller one from KKcore upper bound and DoubleKcore upper bound, to see whether KKcore can help to improve DoubleKcore by outperformance in some cases.

In Fig. 22a, we notice that the KKcore and Color+Kcore perform better when $r = 50$ and $r = 150$ with the same k and l, which is due to the different overlap status of candidate (k,r)-cores with different r values. The overlaps are heavier for some r such that KKcore and Color+Kcore upper bounds sometimes fail to prune unpromising branches while our DoubleKcore cuts a lot of these unpromising branches. Fig. 22a and b shows that DoubleKcore and KKcore+DoubleKcore significantly outperform the other upper bounds for different k and r. DoubleKcore runs even slightly faster than KKcore+DoubleKcore, which indicates KKcore upper bound is outperformed by DoubleKcore in most cases, and the time cost of computing KKcore cannot be paid-off when integrating DoubleKcore and KKcore. It further validates the superior of our DoubleKcore upper bound.

Evaluating α in initial candidate generation Figure 23 shows the effect of varying α in AdvDiv, where all the other techniques are equipped. α is used in initial candidate generation (IN) to limit the number of candidate update. A good α can achieve a balance between time cost of IN and the benefit from IN. Fig. 23a shows we can set α to 1 for Gowalla. Figure 23b shows performance of AdvDiv on DBLP is better when $\alpha = 3$. Fig. 23a and b suggests that the different values of α do not have a significant impact on the runtime of AdvDiv.

Evaluating performance of AdvDiv Figure 24 reports the overall performance of AdvDiv compared with the two baselines DivEnum and BasDiv, with different k, r, l and on different datasets. Fig. 24a shows the runtime of 3 algorithms on all datasets with $k = 5$ and $l = 10$, where $r = 100$ km on location-based datasets and $r =$ top 3‰ for other datasets. We can see the AdvDiv significantly outperform the baseline algorithms. Figure 24b reports the runtime of 3 algorithms on Brightkite with different l values. AdvDiv can outperform the baselines by several times, where the margin can be several orders of magnitude when l is relatively small. Figure 24c shows the runtime of 3 algorithms on Gowalla with different k values. AdvDiv still largely out-

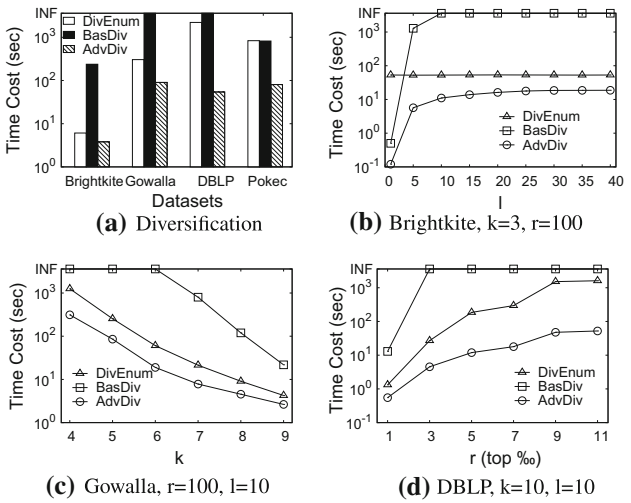


Fig. 24 Evaluation of performance of AdvDiv

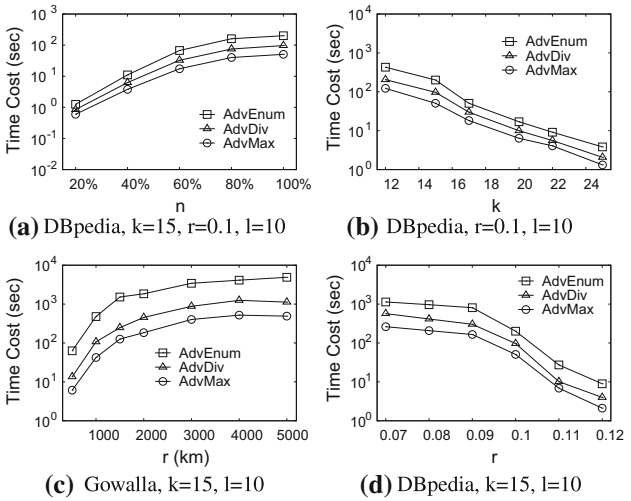


Fig. 25 Scalability evaluation

performs the other algorithms. Figure 24d shows the runtime of 3 algorithms on DBLP with different r values, where the AdvDiv can outperform DivEnum and BasDiv by more than one order of magnitude.

9.6 Scalability of AdvEnum, AdvMax and AdvDiv

Figure 25a shows that the 3 proposed algorithms are scalable with increasing size of DBpedia data, where $k = 15$, $l = 10$ and $r = 0.1$. The graphs are generated by random node sampling. To show a comprehensive result, we also report the trends of runtime with large ranges of similarity constraint r and degree constraint k, respectively. Figure 25b shows the performance on different values of k. Figure 25c and d varies r from 500 km to 5000 km on Gowalla, and from 0.07 to 0.12 on DBpedia, respectively. Note that the result is empty for $r > 0.12$ on DBpedia. We can observe that the runtime is relatively large when extreme values of k or r are used, e.g, $r > 500$ km, while the algorithms can return in 10 s for many reasonable settings, e.g., $r = 100$ km which may be preferred in real-life scenarios.

10 Related work

Mining cohesive subgraphs on graphs A variety of cohesive subgraph models have been widely studied [67], e.g., clique [14], k-core [45] and k-truss [66]. Below, we introduce k-core and clique used in this paper.

k-core The model of k-core, introduced in [45], has many applications such as social contagion [49], influence study [34], user engagement [6,40], and network robustness [63,64, 68]. Batagelj and Zaversnik present a liner in-memory algorithm for core decomposition [5]. An I/O efficient algorithm is proposed for core decomposition on graphs that cannot fit in the main memory [54]. There are some extended models based on k-core, e.g., (k,d)-core [35] and (k,s)-core [62] which are different from our model because they do not consider user similarity in the models.

Clique As a fundamental graph problem, finding cliques has been extensively studied in the literature, e.g., [9,53]. Many algorithms are designed for computing maximal cliques, e.g., [10,11,18,51]. Finding a maximum clique in a unit disk graph (UDG) is polynomially solvable [15], while the algorithm cannot be applied to finding the maximum (k,r)-core because of the structure constraint involved. An algorithm for maximal clique enumeration on UDG is proposed in [29]. The approximate enumeration of maximal cliques [33] may inspire the approximate algorithm for (k,r)-core computation.

Mining attributed graphs It is common to use attributed graphs under various scenarios for real-world social network studies on both research and industry [23,32]. A large amount of classical graph queries have been investigated on attributed graphs such as clustering [57], community detection [58] and network modeling [32]. None of these works combine k-core and pairwise similarity computation on attributed graphs.

There are some investigations on the problem of cohesive subgraph computation on attributed graphs. Wu et al. [56] develop 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. [69] study the computation of a k-core containing a query vertex and within a given spatial region. Chen et al. [13] propose algorithms to find the maximum k-truss where users are co-located. The above two works study community search on location-based social networks while we detect communities considering similarity of

various attributes instead of geographic distances. Fang et al. [22] propose 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. They also propose algorithms [21] 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. Wang et al. [52] develop algorithms to find the k-cores with bounded radius r which contains a query vertex. Their experimental results report that the radius-bounded k-cores are dissimilar to the (k,r)-cores, which implies the difference of radius constraint in [52] and pairwise similarity constraint in the (k,r)-core model.

To the best of our knowledge, (k,r)-core is the first cohesive subgraph model which considers both user engagement and similarity on various types of attributes. Note that the conference version of this paper can be found in [65].

Computing maximum coverage As shown in [24], the greedy algorithm for computing maximum l-coverage can achieve an approximation ratio of $1-1/e$ which cannot be improved by any algorithm in polynomial time unless P=NP. Some works study the maximum l-coverage problem in streaming, e.g., [3,4]. In [4], Badanidiyuru et al. design an algorithm which maintains multiple lists of sets $\cup_{1\leq i\leq l}\mathcal{D}_i$. Let $d_i = (i/2-|cov(\mathcal{D}_i)|)/(l-|\mathcal{D}_i|)$, a new set C is added to $\mathcal{D}_i$ if C covers at least d_i vertices which are not in $\mathcal{D}_i$. The $\mathcal{D}_i$ which covers the most vertices is returned after processing all sets. In [59], Yu and Yuan propose an algorithm which retains a set with high potential to cover some new vertices and removes an existing set if it does not cover any new vertices. The l retained sets with the most covered vertices are returned after processing all sets. Although these two algorithms have better approximation ratios than [3] theoretically, they are not suitable for our DivKRC search because neither of them can lead to the effective pruning techniques proposed in this paper.

Diversified top-l search This search problem aims to find the top-l answers that are most relevant to a query in consideration of diversity. As an extensively studied topic, most existing works focus on answering the query for a specific problem. For example, Lin et al. study the l most representative skyline problem [36]. Some works study the diversified top-k document retrieval [1,2]. Fan et al. study the diversified top-l graph pattern matching [20]. Yuan et al. study the diversified top-l clique search [60]. Nevertheless, the clique structure on a single graph does not consider the user engagement and similarity constraints at the same time. Drosou and Pitoura present a survey on diversifying query results under different scenarios [17]. Some other works study the general framework of diversified top-l search [44,50]. Borodin et al. [8] and Minack et al. [41] study the top-l result diversification on a dynamic environment. Deng and Fan analyze the complexity of query result diversification [16]. Nevertheless, the diversity in above frameworks is established on pairwise dissimilarity of query results. Consequently, these frameworks cannot be applied to solve the DivKRC search efficiently.

11 Discussion

In many real-life networks, it is rather natural to consider both structure and attribute values in many graph problems. we can expect a variety of extensions of (k,r)-core model to be used for different scenarios. The techniques developed in this paper can shed light on the computation of the extensions. 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 graph structure 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. The double k-core-based upper bound and initial candidate generation can be used to speed up the diversified (k,r)-truss search. With similar rationale, other proposed techniques, e.g., candidate retaining, early termination, maximal check and search orders, can also be extended or provide insights into the counterparts of new cohesive subgraph models on attributed graphs.

12 Conclusion

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

Acknowledgements Xuemin Lin is supported by 2018YFB1003504, NSFC61232006, ARC DP180103096 and DP170101628. Ying Zhang is supported by ARC DP180103096 and FT170100128. Lu Qin is supported by ARC DP160101513. Wenjie Zhang is supported by ARC DP180103096.

References

1. Agrawal, R., Gollapudi, S., Halverson, A., Ieong, S.: Diversifying search results. In: WSDM, pp. 5–14 (2009)
2. Angel, A., Koudas, N.: Efficient diversity-aware search. In: SIGMOD, pp. 781–792 (2011)
3. Ausiello, G., Boria, N., Giannakos, A., Lucarelli, G., Paschos, V.T.: Online maximum k-coverage. Discrete Appl. Math. **160**(13–14), 1901–1913 (2012)
4. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: massive data summarization on the fly. In: KDD, pp. 671–680 (2014)
5. Batagelj, V., Zaversnik, M.: An o(m) algorithm for cores decomposition of networks. In: CoRR, cs.DS/0310049 (2003)
6. Bhawalkar, K., Kleinberg, J.M., Lewi, K., Roughgarden, T., Sharma, A.: Preventing unraveling in social networks: the anchored k-core problem. SIAM J. Discrete Math. **29**(3), 1452–1475 (2015)
7. Bird, C., Gourley, A., Devanbu, P. T., Gertz, M., Swaminathan, A.: Mining email social networks. In: MSR, pp. 137–143 (2006)
8. Borodin, A., Lee, H.C., Ye, Y.: Max-sum diversification, monotone submodular functions and dynamic updates. In: PODS, pp. 155–166 (2012)
9. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph (algorithm 457). Commun. ACM **16**(9), 575–576 (1973)
10. Chang, L.: Efficient maximum clique computation over large sparse graphs. In: SIGKDD, pp. 529–538 (2019)
11. Chang, L., Yu, J.X., Qin, L.: Fast maximal cliques enumeration in sparse graphs. Algorithmica **66**(1), 173–186 (2013)
12. Chen, K., Lei, C.: Network game design: hints and implications of player interaction. In: NETGAMES, p. 17 (2006)
13. Chen, L., Liu, C., Zhou, R., Li, J., Yang, X., Wang, B.: Maximum co-located community search in large scale social networks. PVLDB **11**(10), 1233–1246 (2018)
14. Cheng, J., Zhu, L., Ke, Y., Chu, S.: Fast algorithms for maximal clique enumeration with limited memory. In: KDD, pp. 1240–1248 (2012)
15. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Math. **86**(1–3), 165–177 (1990)
16. Deng, T., Fan, W.: On the complexity of query result diversification. PVLDB **6**(8), 577–588 (2013)
17. Drosou, M., Pitoura, E.: Search result diversification. SIGMOD Rec. **39**(1), 41–47 (2010)
18. Eppstein, D., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. In: SEA, pp. 364–375 (2011)
19. Facebook. How does facebook suggest groups for me? https://www.facebook.com/help/382485908586472?helpref=uf_permalink. Accessed 16 Sep 2019
20. Fan, W., Wang, X., Wu, Y.: Diversified top-k graph pattern matching. PVLDB **6**(13), 1510–1521 (2013)
21. Fang, Y., Cheng, R., Li, X., Luo, S., Hu, J.: Effective community search over large spatial graphs. PVLDB **10**(6), 709–720 (2017)
22. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. PVLDB **9**(12), 1233–1244 (2016)
23. Fang, Y., Zhang, H., Ye, Y., Li, X.: Detecting hot topics from twitter: a multiview approach. J. Inf. Sci. **40**(5), 578–593 (2014)
24. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652 (1998)
25. Ferrara, E., JafariAsbagh, M., Varol, O., Qazvinian, V., Menczer, F., Flammini, A.: Clustering memes in social media. In: ASONAM, pp. 548–555 (2013)
26. Garey, M.R., Johnson, D.S.: The complexity of near-optimal graph coloring. JACM **23**(1), 43–49 (1976)
27. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H Freeman, New York (1979)
28. Goldberg, M.K., Kelley, S., Magdon-Ismail, M., Mertsalov, K., Wallace, A.: Finding overlapping communities in social networks. In: SocialCom/PASSAT, pp. 104–113 (2010)
29. Gupta, R., Walrand, J., Goldschmidt, O.: Maximal cliques in unit disk graphs: polynomial approximation. In: Proceedings INOC, vol. 2005. Citeseer (2005)
30. Hristova, D., Musolesi, M., Mascolo, C.: 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)
31. Huang, X., Lu, W., Lakshmanan, L.V.S.: Truss decomposition of probabilistic graphs: Semantics and algorithms. In: SIGMOD, pp. 77–90 (2016)
32. Pfeiffer, J.J III., Moreno, S., Fond, T.L., Neville, J., Gallagher, B.: Attributed graph models: modeling network structure with correlated attributes. In: WWW, pp. 831–842 (2014)
33. Izumi, T., Suzuki, D.: Faster enumeration of all maximal cliques in unit disk graphs using geometric structure. IEICE Trans. **98–D**(3), 490–496 (2015)
34. Kitsak, M., Gallos, L.K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H.E., Makse, H.A.: Identification of influential spreaders in complex networks. Nat. Phys. **6**(11), 888–893 (2010)
35. Lee, P., Lakshmanan, L.V.S., Milios, E.E.: CAST: a context-aware story-teller for streaming social content. In: CIKM, pp. 789–798 (2014)
36. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representative skyline operator. In: ICDE, pp. 86–95 (2007)
37. Liu, Y., Sutanto, J.: Buyers purchasing time and herd behavior on deal-of-the-day group-buying websites. Electron. Mark. **22**(2), 83–93 (2012)
38. Luo, M.M., Chea, S.: The effect of social rewards and perceived effectiveness of e-commerce institutional mechanisms on intention to group buying. In: Advances in Human Factors, Business Management, Training and Education, pp. 833–840. Springer, Berlin (2017)
39. Luo, X., Andrews, M., Song, Y., Aspara, J.: Group-buying deal popularity. J. Mark. **78**(2), 20–33 (2014)
40. Malliaros, F.D., Vazirgiannis, M.: To stay or not to stay: modeling engagement dynamics in social graphs. In: CIKM, pp. 469–478 (2013)
41. Minack, E., Siberski, W., Nejdl, W.: Incremental diversification for very large sets: a streaming-based approach. In: SIGIR, pp. 585–594 (2011)
42. Mitzlaff, F., Atzmüller, M., Hotho, A., Stumme, G.: The social distributional hypothesis: a pragmatic proxy for homophily in online social networks. Soc. Netw. Anal. Min. **4**(1), 216 (2014)
43. PokemonGo. Developer insights: Inside the philosophy of friends and trading. https://pokemongolive.com/en/post/jundevupdate-trading/. Accessed 16 Sep 2019
44. Qin, L., Yu, J.X., Chang, L.: Diversifying top-k results. PVLDB **5**(11), 1124–1135 (2012)
45. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
46. Sharma, P., Govindan, S.: Information seeking behavior of expats in asia on facebook open groups. Singap. J. Libr. Inf. Manag. **44**, 35 (2016)
47. Singla, P., Richardson, M.: Yes, there is a correlation—from social networks to personal behavior on the web. In: WWW, pp. 655–664 (2008)
48. Statista. Number of active users of pokemon go worldwide from 2016 to 2020, by region (in millions). https://www.statista.com/statistics/665640. Accessed 16 Sep 2019
49. Ugander, J., Backstrom, L., Marlow, C., Kleinberg, J.: Structural diversity in social contagion. PNAS **109**(16), 5962–5966 (2012)
50. Vieira, M.R., Razente, H.L., Barioni, M.C.N., Hadjieleftheriou, M., Srivastava, D., Traina, Jr. C., Tsotras, V.J.: On query result diversification. In: ICDE, pp. 1163–1174 (2011)

51. Wang, J., Cheng, J., Fu, A.W.: Redundancy-aware maximal cliques. In: KDD, pp. 122–130 (2013)
52. Wang, K., Cao, X., Lin, X., Zhang, W., Qin, L.: Efficient computing of radius-bounded k-cores. In: ICDE, pp. 233–244 (2018)
53. Wang, K., Lin, X., Qin, L., Zhang, W., Zhang, Y.: Vertex priority based butterfly counting for large-scale bipartite networks. PVLDB **12**(10), 1139–1152 (2019)
54. Wen, D., Qin, L., Zhang, Y., Lin, X., Yu, J.X.: I/O efficient core graph decomposition at web scale. In: ICDE, pp. 133–144 (2016)
55. Wu, S., Sarma, A.D., Fabrikant, A., Lattanzi, S., Tomkins, A.: Arrival and departure dynamics in social networks. In: WSDM, pp. 233–242 (2013)
56. Wu, Y., Jin, R., Zhu, X., Zhang, X.: Finding dense and connected subgraphs in dual networks. In: ICDE, pp. 915–926 (2015)
57. Xu, Z., Ke, Y., Wang, Y., Cheng, H., Cheng, J.: A model-based approach to attributed graph clustering. In: SIGMOD, pp. 505–516 (2012)
58. Yang, J., McAuley, J.J., Leskovec, J.: Community detection in networks with node attributes. In: ICDM, pp. 1151–1156 (2013)
59. Yu, H., Yuan, D.: Set coverage problems in a one-pass data stream. In: SDM, pp. 758–766 (2013)
60. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: Diversified top-k clique search. In: ICDE, pp. 387–398 (2015)
61. Yuan, Q., Zhao, S., Chen, L., Liu, Y., Ding, S., Zhang, X., Zheng, W.: Augmenting collaborative recommender by fusing explicit social relationships. In: Recsys Workshop (2009)
62. Zhang, F., Yuan, L., Zhang, Y., Qin, L., Lin, X., Zhou, A.: Discovering strong communities with user engagement and tie strength. In: DASFAA, pp. 425–441 (2018)
63. Zhang, F., Zhang, W., Zhang, Y., Qin, L., Lin, X.: OLAK: an efficient algorithm to prevent unraveling in social networks. PVLDB **10**(6), 649–660 (2017)
64. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: Finding critical users for social network engagement: the collapsed k-core problem. In: AAAI, pp. 245–251 (2017)
65. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: efficient (k, r)-core computation on social networks. PVLDB **10**(10), 998–1009 (2017)
66. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: Efficiently reinforcing social networks over user engagement and tie strength. In: ICDE, pp. 557–568 (2018)
67. Zhang, Y., Qin, L., Zhang, F., Zhang, W.: Hierarchical decomposition of big graphs. In: ICDE, pp. 2064–2067 (2019)
68. Zhou, Z., Zhang, F., Lin, X., Zhang, W., Chen, C.: K-core maximization: An edge addition approach. In: IJCAI, pp. 4867–4873 (2019)
69. Zhu, Q., Hu, H., Xu, C., Xu, J., Lee, W.: Geo-social group queries with minimum acquaintance constraints. VLDB J. **26**(5), 709–727 (2017)