Knowledge and Information Systems (2021) 63:3217–3250
https://doi.org/10.1007/s10115-021-01624-x

REGULAR PAPER

Discovering fortress-like cohesive subgraphs

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

Received: 1 June 2020 / Revised: 28 October 2021 / Accepted: 7 November 2021 /
Published online: 24 November 2021

Abstract
Morris (Rev Econ Stud 67:57–78, 2000) defines the p-cohesion by a connected subgraph in which every vertex has at least a fraction p of its neighbors in the subgraph, i.e., at most a fraction $(1 - p)$ of its neighbors outside. We can find that a p-cohesion ensures not only inner-cohesiveness but also outer-sparseness. The textbook on networks by Easley and Kleinberg (Networks, Crowds, and Markets - Reasoning About a Highly Connected World, Cambridge University Press, 2010) shows that p-cohesions are fortress-like cohesive subgraphs which can hamper the entry of the cascade, following the contagion model. Despite the elegant definition and promising properties, to our best knowledge, there is no existing study on p-cohesion regarding problem complexity and efficient computing algorithms. In this paper, we fill this gap by conducting a comprehensive theoretical analysis on the complexity of the problem and developing efficient computing algorithms. We focus on the minimal p-cohesion because they are elementary units of p-cohesions and the combination of multiple minimal p-cohesions is a larger p-cohesion. We demonstrate that the discovered minimal p-cohesions can be utilized to solve the MinSeed problem: finding a smallest set of initial adopters (seeds) such that all the network users are eventually influenced. Extensive experiments on 8 real-life social networks verify the effectiveness of this model and the efficiency of our algorithms.

Keywords Cohesive subgraph · Information barrier · Social network · MinSeed

1 Introduction

Graphs have been widely used in many real-life applications. A variety of cohesive subgraph models are proposed for graph analysis, including k-core, k-truss, clique, p-cohesion and so on. Given a graph and a real number $p \in (0, 1)$, a p-cohesion is defined as a connected subgraph where every vertex has, at least, a fraction p of its neighbors in the subgraph [29]. This definition implies that every vertex in a p-cohesion has, at most, a fraction $(1 - p)$ of its neighbors outside of the p-cohesion. The advantages of the p-cohesion model over other existing cohesive subgraph models are twofold: First, with a large p value, we can find a p-cohesion ensures not only inner-cohesiveness, as the vertices inside a p-cohesion are cohesive; but also outer-sparseness, as the outside neighbors of the p-cohesion have a sparse connection to the p-cohesion; second, in many applications, it is more natural to consider

Extended author information available on the last page of the article

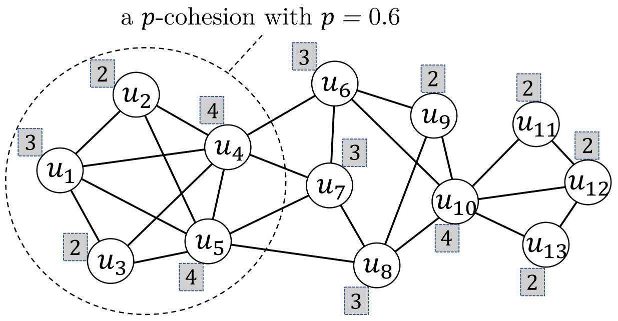


Fig. 1 A Graph, $p = 0.6$

the percentage of the neighbors instead of the same number of neighbors (such as the k value in a k-core) within the cohesive subgraph. For instance, in real-life social networks, a large-degree user may need more neighbors than a small-degree user to encourage her/him to adopt a behavior [4].

The p-cohesion is also related to the contagion model, which is introduced in [29] to study the interaction of large populations: Given a graph with some initial adopters of a behavior "A" and a cascading threshold $r \in (0, 1)$, a user will adopt "A" if at least a fraction r of his/her neighbors already adopted "A." Clearly, with the contagion model, for any p-cohesion S with $p > (1 - r)$, none of the users in S will adopt "A" if S does not contain any initial adopters; in other words, the influence coming from outside of S alone cannot affect any user in S. In this sense, a p-cohesion is a fortress regarding the contagion model.

Example 1 Figure 1 shows a small graph. Suppose $p = 0.6$, we use the gray-filled rectangles to label every user u with the smallest number of neighbors required for u to stay in a p-cohesion. A minimal p-cohesion S is marked in the dashed circle, which is the subgraph induced by vertices u_1, u_2, u_3, u_4 and u_5. Suppose $r = 0.5$ in the contagion model and there is no initial adopter in S for product "A," any user in S will not adopt "A" even if all the other users (not in S) adopted "A."

The fortress-like cohesive subgraphs (i.e., p-cohesions) are critical for information diffusion related applications. They may be information islands in social networks, focalization in viral marketing, etc. We may benefit from exploiting the fortress property of p-cohesions. For instance, in viral marketing, it is usually hard to peddle a new product to homogeneous users in social groups (e.g., p-cohesions) who use another competing product, while these users form a business focalization [12]. Offering incentives such as discounts or free product trials to users in p-cohesions may help in the successful marketing of a new product.

In this paper, we are interested in the minimal p-cohesion problem where we say a p-cohesion S is minimal if there does not exist a proper subgraph S' of S ($S' \subset S$) which is also a p-cohesion. This is because: (1) they are elementary units of p-cohesions, and the union of multiple minimal p-cohesions forms a larger p-cohesion; (2) we may avoid enumerating an overwhelming number of p-cohesions; and (3) it is more useful to find small fortresses in influence-related applications. For instance, it is immediate that the whole graph or a connected component is a p-cohesion for any p value, but this is not interesting.

Minimal p-cohesions also enable us to find good heuristics for the MinSeed problem under the contagion model; that is, given a target graph and a cascading threshold r, find a minimum set of seeds such that the whole graph is eventually influenced. The minimal p-cohesions can hinder the entry of cascades to it, and the vertices inside a p-cohesion are relatively isolated from the outside vertices. Therefore, by giving certain priorities to seed the vertices inside the minimal p-cohesions, we may break the entry barriers of the p-cohesions by the great influence power of the selected seeds. The p-cohesion can thus benefit the applications of influence studies, e.g., viral marketing and online fraud control [18].

In this paper, we study two representative problems with regard to the p-cohesion model: minimum p-cohesion search and diversified p-cohesion enumeration.

- *Minimum* p**-cohesion search** aims to find the smallest p-cohesion containing the given query vertex, i.e., the p-cohesion with the smallest number of vertices to which a user (query vertex) belongs. We show this problem is NP-hard, and some heuristics are proposed to efficiently identify a p-cohesion with a small size for the given query vertex.
- *Diversified* p**-cohesion enumeration** aims to find a set of disjoint p-cohesions which can cover as many vertices as possible. Here, we consider diversity because, in practice, the user may be overwhelmed by the exponential number of minimal p-cohesions. Thus, in this paper, we design efficient algorithms to find a set of disjoint minimal p-cohesions.

1.1 Contributions

Despite the elegant definition and promising properties of the p-cohesion model, to our best knowledge, there is no study on the problem hardness or the efficient computing algorithms. The major contributions of this paper are as follows.

On the theoretical side, we prove that (1) the problem of finding the smallest p-cohesion is NP-hard and it does not admit any constant-factor approximation, unless P=NP; and (2) the number of minimal p-cohesions can be exponential in the graph size. On the practical side, we propose efficient algorithms to find a small p-cohesion containing a query vertex, as the basic computing unit for p-cohesion related computations. Due to the huge number of p-cohesions, we also propose efficient algorithms based on the computing unit to identify a set of disjoint minimal p-cohesions. As an application example, the algorithms are applied in the MinSeed problem to reduce the seeding cost, by carefully utilizing the fortress property of p-cohesions. Comprehensive experiments are conducted to demonstrate the effectiveness of p-cohesion on modeling cohesive subgraphs, compared with other classical cohesive subgraph models. Due to the fortress property, the proposed algorithms are also effective on solving influence problems such as influence maximization, under different influence cascade models. Extensive efficiency reports show that our algorithms can fast return on large real-life graphs.

2 Related work

2.1 Cohesive subgraph models

Various subgraph models are proposed to accommodate different scenarios, including p-cohesion studied in this paper. Clique [11,26] is the most cohesive subgraph where every two vertices are adjacent. Because of the over-restriction of the clique model, some clique relaxation models are proposed such as k-plex [34], k-core [21,25,33], k-truss [9,44], k-fami [43] and dense subgraph [13,31], to name a few. It is intractable to compute some models such as clique enumeration and finding the densest k subgraph, as there is no polynomial-time algorithm existed.

Some cohesive subgraph works aim to discover small-size subgraphs. Amini *e*t al. [2] study some degree-constrained subgraph problems including the minimum-size k-core search. Cui *e*t al. [10] propose efficient algorithms to locally search the k-core. Barbieri *e*t al. [5] propose greedy algorithms to search the minimum k-core for one or multiple query vertices. Wood *e*t al. [39] study the k-assemblies based on minimal k-core computation. The above cohesive subgraph models only consider the cohesiveness inside the subgraphs and ignore the interactions from outside. The work most-related with p-cohesion is the k-defensive alliance [15,32,41] which finds a subgraph S where each vertex v in S has at least k more neighbors in S than out of S. When $k = 0$, the model corresponds to the p-cohesion with $p = 0.5$. However, there is no such mapping between the two models for an arbitrary p value. We stress that none of the above subgraph models possess the fortress property of p-cohesion to defend the influence of information cascades.

According to the definition of p-cohesion [29], different vertices in a p-cohesion subgraph need different number of neighbors inside the subgraph to remain engaged. Therefore, it is infeasible to apply the methods with fixed threshold for each unit (vertex or edge), e.g., k-core and k-truss computations. The framework of clique enumeration [7] can be utilized to find minimal p-cohesions, while this exact solution can only handle very small graphs, e.g., it cannot finish in one week for a graph with 80 vertices in our experiments. It is consistent with the hardness results of p-cohesion computation proved in this paper: Finding the minimum p-cohesion is NP-hard to approximate for any constant factor, and there are graphs with an exponential number of minimal p-cohesions.

Influence cascade models Morris [29] introduces the contagion model and p-cohesion to characterize social choices in local interaction systems. The paper also studies the diffusion of a behavior from a finite set of initial adopters to all network users. Ugander *e*t al. [37] show that the contagion probability of a user is strongly influenced by his local neighbors, e.g., neighbors in cohesive subgraphs. Zarezade *e*t al. [42] study correlated cascades based on the fact that the adoption of a behavior by a user is influenced by the aggregation of the behaviors of his/her neighbors. Easley and Kleinberg [12] make the same observation as the above in various applications. They further emphasize the contagion model and the p-cohesion that it is difficult for new innovations to enter tightly knit social groups (i.e., p-cohesions), because people tend to interact with their friends or acquaintances.

In addition to the contagion model, there are some other information cascade models, such as independent cascade model (IC) and linear threshold (LT) model [19], where the influence maximization problem and seed minimization problem have been extensively studied, e.g., [3,16,35,45]. It uses polynomial time to compute the influence spread of given seeds in the contagion model, while this influence computation is NP-hard for both IC and LT models [27]. For a set of seeds, the spread area of their influence is certain in contagion model, while the influence spread is uncertain and complex in IC and LT models due to the possible world assumption.

3 Preliminaries

Assume there is an unweighted and undirected graph $G = (V, E)$ with n vertices and m edges, where V (resp. E) represents the set of vertices (resp. edges). S denotes a subgraph of G. Let $N(u, S)$ denote the set of adjacent vertices (i.e., neighbors) of u in S. Let $deg(u, S)$ denote the number of vertices in S which are adjacent to u. We may omit the target graph in

Table 1 Summary of Notations

Notation	Definition
G	An unweighted and undirected graph
S	A subgraph of G
$V(S)$	The vertex set of S
$E(S)$	The edge set of S
$G(V)$	The induced subgraph of a vertex set V on G
$u, v; e$	A vertex in G; an edge in G
n, m	The number of vertices and edges in G
$N(u, S)$	The adjacent vertices of u in S
$deg(u, S)$	The number of adjacent vertices of u in S
$E(u)$	The set of incident edges to u in G
D	A set of seed vertices

notations when the context is clear, e.g., using $deg(u)$ instead of $deg(u, G)$. In this paper, if a vertex is deleted, its incident edges are also deleted accordingly.

The contagion model in [12,29] defines the following cascading condition when some vertices are activated and the others are not.

Definition 1 **Cascading Condition**. Given a graph G and a cascading threshold $r \in (0, 1)$, an inactivated vertex $u \in V(G)$ will be activated iff the number of activated neighbors of u is at least $\lceil r \times deg(u, G) \rceil$.

The cascades in a graph may imply some interesting subgraphs which are named p-cohesions.

Definition 2 **p-Cohesion**. Given a graph G and a real number $p \in (0, 1)$, a connected subgraph S is a p-cohesion of G, denoted by $C_p(G)$, where $deg(u, S) \geq \lceil p \times deg(u, G) \rceil$ for every vertex $u \in S$.

Easley and Kleinberg [12] prove that the p-cohesion subgraphs have the "fortress" property to hamper the progression of information cascades.

Property 1 **"Fortress."** Given a graph G and a real number $p \in (0, 1)$, for an arbitrary p-cohesion S of G, according to the *Cascading Condition* (Definition 1) with threshold $r > 1 - p$, no vertex in S can be activated when all vertices in S are inactivated initially, even if all the vertices in $V(G)\backslash V(S)$ are activated.

Definition 3 **Minimal p-cohesion**. Given a graph G and a real number $p \in (0, 1)$, a p-cohesion S of G is called minimal if it is an elementary unit of p-cohesion, i.e., every proper subgraph $S' \subset S$ is not a p-cohesion.

Problem statement Given an undirected and unweighted graph G, and a real number $p \in (0, 1)$, we aim to develop algorithms for the following two representative problems: (1) $\underline{\text{M}}$inimum $\underline{p}$-$\underline{\text{C}}$ohesion $\underline{\text{S}}$earch (MPCS): Given a query vertex q, find the smallest p-cohesion containing q in G; and (2) $\underline{\text{D}}$iversified $\underline{p}$-$\underline{\text{C}}$ohesion $\underline{\text{E}}$numeration (DPCE): Enumerate a set of disjoint minimal p-cohesions.

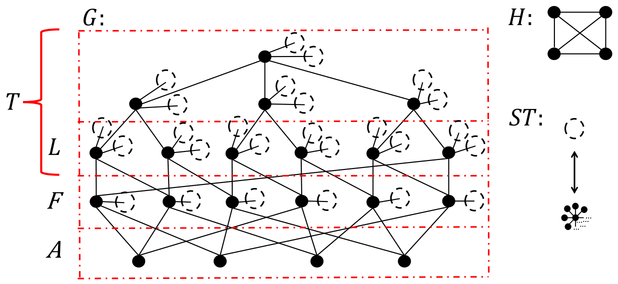


Fig. 2 Construction Example of Theorem 1, $p = \frac{1}{2}$

4 Minimum *p*-cohesion search

In this section, we study the minimum p-cohesion search problem, i.e., MPCS. Firstly, we analyze the complexity of the problem, and then present the solution.

4.1 Problem analysis

In this section, we prove that the MPCS problem is NP-hard to approximate for any constant factor. Let MinPC denote the computation to find the smallest p-cohesion in G (without query vertices).

Theorem 1 *The MPCS problem is NP-hard, for any fixed* $p \in (0, 1)$.

Proof We prove the hardness of MPCS by the hardness of finding a minimum p-cohesion in G without query vertices, i.e., MinPC. If there is a polynomial solution for MPCS, we can immediately come up with a polynomial solution to find the minimum p-cohesion in G (MinPC) by conducting MPCS on every vertex.

For MinPC, we show a reduction from Vertex Cover in cubic graphs which proved to be NP-hard in cubic graphs [1,17]. Given an arbitrary cubic graph H as an instance of vertex cover, with $|V(H)| = n$, we construct an instance G of MinPC as follows.

We may assume that $|E(H)| = \frac{3n}{2} = 3 \times 2^l$ for some integer l, without loss of generality [2]. As shown in Fig. 2, we construct a rooted tree T with a height of $l + 1$, where the root vertex has 3 child vertices. Except for the root and leaves, every vertex in T has 1 parent vertex and 2 child vertices. So T contains 3×2^l leaf vertices, where the leaf set is denoted by L. Then, we add a copy of L, denoted by F, and add a Hamiltonian cycle on L and F as in the figure. The elements in F are identified with the elements in $E(H)$. Now, we add a set A which is a copy of $V(H)$ with the identifications. Then, we add an edge between a vertex $u \in A$ and a vertex $e \in F$ (i.e., corresponding to an edge in H) if and only if u is incident to e in H. Let $\widetilde{G}$ be the graph constructed in this way.

Let ST denote a *star-like* subgraph which is induced by a center vertex and its $\lceil \frac{|\widetilde{G}|}{p} \rceil$ neighbors, where $|\widetilde{G}| = |V(T)| + |V(F)| + |V(A)|$. Then, for every vertex u in T (resp. F), we connect u to every center vertex in $k - 3$ (resp. $max(k - 4, 0)$) copies of ST, where $k = \lfloor \frac{2}{p} \rfloor + 1$. The construction is completed.

Every vertex in the star-like subgraph has a degree one except the center vertices. When $\frac{2}{3} < p < 1$, we have that every vertex in F has a degree of 4 in G and every vertex in $V(G) \setminus F$ has a degree of 3 in G. When $0 < p \leq \frac{2}{3}$, every vertex in T and F has a degree of

k in G. If a p-cohesion C_p contains a vertex in one of the star-like subgraphs, then C_p has to contain more than $|V(T)| + |V(F)| + |V(A)|$ vertices which makes C_p not a minimum p-cohesion, as shown in the following.

According to the definition of p-cohesion, if a vertex $u \in T \cup F$ is in a minimum p-cohesion (C_p) of G, u has at least 3 neighbors in C_p. Based on the construction, if a minimum p-cohesion contains a vertex in $T \cup F$, it has to contain every vertex in $T \cup F$. Furthermore, a minimum p-cohesion cannot only contain the vertices in $T \cup F$ or A. So a minimum p-cohesion is induced by all the vertices in $T \cup F$ and a smallest subset of vertices in A such that each vertex in F has at least a degree of 3 in the p-cohesion. Clearly, such a p-cohesion contains at most $|V(T)| + |V(F)| + |V(A)|$ number of vertices. Then, the minimum p-cohesion search problem on G is exactly the Vertex Cover problem in H. □

Theorem 2 *The MPCS problem does not admit a PTAS, unless P = NP.*

Proof We still use the reduction in the proof of Theorem 1, and let G be the graph constructed in the reduction from an arbitrary cubic graph H (as in Fig. 2), where $|V(H)| = n$, and $|E(H)| = \frac{3n}{2} = 3 \times 2^l$ for some integer l. As proved, the minimum p-cohesion search problem on G is exactly the Vertex Cover problem in H. The p-cohesion of the constructed graph G contains at most $|V(T)| + |V(F)| + |V(A)|$ number of vertices, where $|V(T)| = 1 + 3 \times 2^0 + 3 \times 2^1 + \cdots + 3 \times 2^l = 3 \times 2^{l+1} - 2 = \frac{6n}{2} - 2$, $|V(F)| = L = 3 \times 2^l = \frac{3n}{2}$, and $|V(A)| = n$. Thus, we have

$$OPT_{MinPC}(G) = OPT_{VC}(H) + |V(T)| + |V(F)| = OPT_{VC} + \frac{9n}{2} - 2 \tag{1}$$

where $OPT_{MinPC}(G)$ (resp. $OPT_{VC}(H)$) is the size of the optimal solution for the MinPC in the constructed graph G (resp. Vertex Cover in cubic graph H). We omit the target graph in notations when the context is clear, e.g., using OPT_{MinPC} instead of $OPT_{MinPC}(G)$. Note that any solution of MinPC in G of size $SOLN_{MinPC}$ induces a solution of Vertex Cover problem in H of size $SOLN_{VC} = SOLN_{MinPC} - \frac{9n}{2} + 2$. Suppose that MinPC admits a PTAS, i.e., for any $\epsilon > 0$ we can find a solution for MinPC in polynomial time in graph G of size $SOLN_{MinPC} \leq (1+\epsilon) \cdot OPT_{MinPC}$. Therefore, we can find a solution of Vertex Cover in H in polynomial time with size

$$SOLN_{VC} = SOLN_{MinPC} - \frac{9n}{2} + 2 \leq (1+\epsilon) \cdot OPT_{MinPC} - \frac{9n}{2} + 2 \tag{2}$$

Based on Eqs. 1 and 2, we have

$$SOLN_{VC} \leq (1+\epsilon) \cdot OPT_{VC} + \epsilon \cdot \left(\frac{9n}{2} - 2\right) \tag{3}$$

Since H is a cubic graph, any solution of Vertex Cover in H has at least $\frac{|E(H)|}{3} = \frac{n}{2}$ nodes, i.e., $\frac{n}{2} \leq OPT_{VC}$. Using this in Eq. 3, we have

$$SOLN_{VC} \leq (1+\epsilon) \cdot OPT_{VC} + \epsilon \cdot \left(\frac{9n}{2} - 2\right) \leq (1 + 10 \cdot \epsilon) \cdot OPT_{VC} \tag{4}$$

Thus, the existence of a PTAS for MinPC would imply the existence of a PTAS for Vertex Cover in the cubic graphs, which is impossible unless P = NP [1]. □

Theorem 3 *The MPCS problem does not admit any constant-factor approximation, unless P = NP.*

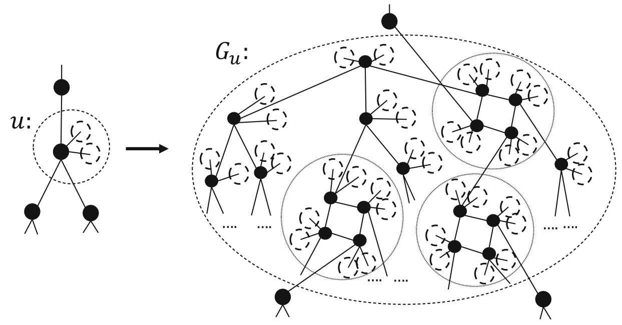


Fig. 3 Error Amplification in the Proof of Theorem 3, $p = \frac{1}{2}$

Proof Following the construction in Theorem 2, the theorem is proved by applying the standard error amplification technique [2]. Let $\mathcal{G}_1 = \{G\}$ be the family of graphs constructed from the instances H of Vertex Cover (as in Fig. 2), G being a typical member of this family, and let $\alpha > 1$ be the factor of approximation of MinPC that exists by Theorem 1.

We construct a sequence of families of graph $\mathcal{G}_s$, such that MinPC problem is hard to approximate within a factor $\theta(\alpha^s)$ in the family $\mathcal{G}_s$. This proves that MinPC does not have any constant-factor approximation. In the following, G_s will denote a typical element of $\mathcal{G}_s$ constructed from the element $G \in \mathcal{G}_1$. We describe the construction of $\mathcal{G}_2$ and obtain the result by repeating the same construction inductively to obtain $\mathcal{G}_s$.

We take a copy of the subgraph $\widetilde{G} \subset G$ constructed in the proof of Theorem 1 and denote it as G'. It is proved that a p-cohesion C_p does contain any vertex in ST in the construction. For every vertex $u \in G'$, let $d_u = deg(u, G')$. For each u, we construct a graph G_u as follows. First, take another copy of the subgraph $\widetilde{G} \subset G$, denoted as G'', and choose d_u arbitrary vertices $x_1, x_2, \dots, x_{d_u}$ of degree three in $T \subset G''$, i.e., $x_i \in T$, $i = 1, 2, \dots, d_u$. Then, replace each of these vertices x_i with a cycle of length four, and join three vertices of the cycle to the three neighbors of x_i, $i = 1, \dots, d_u$. Then, we connect the d_u edges incident to u to the d_u vertices of degree two in the cycles. Let G_u be the graph constructed in this way. An example is shown in Fig. 3.

Now, replace every vertex $u \in G'$ with G_u mentioned above, denoted the constructed graph as $\widetilde{\mathcal{G}_2}$. Similar to Theorem 1, let ST denote a *star-like* subgraph which is induced by a center vertex and its $\lceil \frac{|\widetilde{\mathcal{G}_2}|}{p} \rceil$ neighbors. Then for every vertex $u \in \widetilde{\mathcal{G}_2}$ with degree 3 (resp. with degree 4), we join u to every center vertex in $k - 3$ (resp. $max(k - 4, 0)$) copies of ST, where $k = \lfloor \frac{2}{p} \rfloor + 1$. This completes the construction of the graph $\mathcal{G}_2$.

We have that $|V(\mathcal{G}_2)| = |V(\widetilde{G})|^2 + o(|V(\widetilde{G})|^2) + o(\lceil \frac{|\widetilde{\mathcal{G}_2}|}{p} \rceil)$, because each vertex of $\widetilde{G}$ is replaced with a copy of $\widetilde{G}$ where we had replaced some of the vertices with a cycle of length four, and we add some copies of ST to vertices in $\widetilde{\mathcal{G}_2}$, i.e., $o(\lceil \frac{|\widetilde{\mathcal{G}_2}|}{p} \rceil)$. To find a solution of the MinPC problem in $\mathcal{G}_2$, for any $u \in V(G)$, once a vertex in G_u is chosen, we have to solve MinPC in G, which is hard up to a constant factor α. But approximating the number of u's for which we should touch G_u is also solving MinPC in G, which is hard up to the same factor α. This proves that approximating MinPC in $\mathcal{G}_2$ is hard up to a factor α^2. The proof of the theorem is completed by repeating this procedure, applying the same construction to obtain $\mathcal{G}_3$, and inductively $\mathcal{G}_s$. Note that in each construction of $\mathcal{G}_i$, the size of the ST is different, which is $\lceil \frac{|\widetilde{\mathcal{G}_i}|}{p} \rceil$, where $i = 1, 2, \dots, s$. Note that $\mathcal{G}_i$ is a supergraph of $\widetilde{\mathcal{G}_i}$ by adding some

Algorithm 1: **SubPCExact**(R, P, X, p, c)

```
Input  : $R, P, X$ : three data sets, $p$ : a real number in (0, 1), $c$ : a positive integer
Output : $MinC$ : a p-cohesion containing $q$
1 if all $v \in R$ are with $deg(v, G(R)) \geq \lceil p \times deg(v, G) \rceil$ then
2     if $|R| < c$ then
3         $MinC = G(R), c = |R|$;
4         return;
5 for each $v \in P$ do
6     $R \leftarrow R \cup \{v\}, X \leftarrow X \cup \{v\}$;
7     SubPCExact($R, (P \cup \{N(v) - R\} - X), X, p, c$);
8     $R \leftarrow R - \{v\}$;
9 return $MinC$
```

Algorithm 2: **ExactPC**(S, p, q)

```
Input  : $S$ : a graph, $p$ : a real number in (0,1), $q$ : a query vertex
Output : $MinC_p$ : the minimum $p$-cohesion containing $q$
1 $MinC_p \leftarrow \emptyset, c = |V(S)|, R \leftarrow \emptyset, P \leftarrow \emptyset, X \leftarrow \emptyset$;
2 $MinC_p \leftarrow$ SubPCExact($R \cup \{q\}, P \cup N(q), X, p, c$);
3 return $MinC_p$
```

copies of ST to some vertices in $\widetilde{\mathcal{G}_i}$. In order to build $\mathcal{G}_s$ from $\mathcal{G}_{s-1}$, we replace each vertex $u \in V(\mathcal{G}_{s-1})$ with a copy of $\mathcal{G}_{s-1}$ in which $deg(u, \mathcal{G}_{s-1})$ arbitrary vertices of degree three have been replaced with a cycle of length four. □

4.2 Exact search algorithms

Algorithms 1 and 2 show the pseudo-code for finding the exact minimum p-cohesions containing a query vertex, based on the framework proposed by Bron and Kerbosch [7] which is to enumerate the maximal cliques in a backtracking manner. In Algorithm 1, we use R to denote the intermediate vertex set of a p-cohesion containing the query vertex q. $G(R)$ is a p-cohesion that every vertex in R satisfies the definition of p-cohesion, i.e., $deg(v, G(R)) \geq \lceil p \times deg(v, G) \rceil$. Set P is the candidate set that the combining of P and R may be a p-cohesion. By X, we denote the vertices that have been processed. p is the parameter for p-cohesion, and c is the minimum size of all p-cohesions we found. Within each recursive call, the algorithm considers vertices in P in turn. If there are no such vertices, it backtracks. For each vertex v chosen from P, it makes a recursive call in which v is added to R and X, respectively, and in which P is $P \cup (N(v) - R) - X$, which finds and reports all p-cohesions containing vertex v. Then, it removes v from R and P to exclude it in future p-cohesions and continues with the next vertex in P.

In Algorithm 2, we invoke Algorithm 1 with $R = \{q\}$, $P = N(q)$, $X = \{q\}$ and $c = |V(S)|$, where S is the input graph. The p-cohesion returned by Algorithm 1 is the minimum p-cohesion containing query vertex q. Note that we can also enumerate all the p-cohesions of a graph, i.e., execute Algorithm 1 with "*SubPCExact*($\emptyset$, $V(G)$, $\emptyset$, p, $|V(G)|$)."

As our experiments on real-life graphs find that the size of a minimal p-cohesion is close to the minimum p-cohesion, and the number of minimal p-cohesions is enormous, considering the hardness results of p-cohesion computation, we design heuristic algorithms to fast retrieve a minimal p-cohesions or a set of disjoint minimal p-cohesions.

```
Algorithm 3: Collapse($S$, $p$, $u$)
   Input   : $S$ : a graph, $p$ : a real number in (0,1), $u$ : a vertex
   Output  : $C_p$ : a smaller $p$-cohesion or an empty set
1  $S \leftarrow S \setminus \{u \cup E(u)\}$;
2  while $\exists v \in V(S)$ with $deg(v, S) < \lceil p \times deg(v, G) \rceil$ do
3      $S \leftarrow S \setminus \{v \cup E(v)\}$;
4  return $S$
```

4.3 Heuristic search algorithms

Global search algorithm. As previously mentioned, the whole graph or a connected component is a p-cohesion. By removing a vertex from a p-cohesion, it may cause the `collapse` of the p-cohesion, resulting in a smaller p-cohesion or an empty set. In Algorithm 3, we show the removal of a vertex may lead to the shrink of the p-cohesion.

Let u be a vertex that should be removed from a graph S. After removing u and the edges incident to u, if there is a vertex v in S violating the p-cohesion constraint (Line 2), we remove v and its incident edges from S at Line 3. The algorithm returns a smaller p-cohesion or an empty set.

Time complexity. If a vertex u is deleted at Line 1 or 3, only the neighbors of u may violate the definition of p-cohesion (Line 2). Thus, each vertex is visited once for deletion and each edge is visited once for degree update and vertex marking. The time complexity of Algorithm 3 is $\mathcal{O}(m + n)$.

Space complexity. The subgraph S and the neighbor set take $\mathcal{O}(m + n)$ space, respectively. The degree set, to-delete set and vertex index take $\mathcal{O}(n)$ respectively. The space complexity of Algorithm 3 is $\mathcal{O}(m + n)$.

Example 2 Figure 1 shows a graph with the label of the smallest number of neighbors required for every vertex in a p-cohesion, when $p = 0.6$. If the input graph S is induced by $\{u_1, \ldots, u_{13}\}$, and $u = u_{12}$, in Algorithm 3, the deletion of u leads to the removal of u_{11}, u_{13} and u_{10} (Lines 2-3) according to the label (p-cohesion constraint). The returned S is induced by $\{u_1, \ldots, u_9\}$. In Algorithm 3, if the input graph S is induced by $\{u_1, u_2, u_3, u_4, u_5\}$, and $u = u_4$, the deletion of u leads to the removal of u_5, u_2, u_3 and u_1, such that the returned S is an empty set.

Based on the above `collapse` procedure, we propose a global search algorithm to compute a minimal p-cohesion containing q in a top-down manner. Let S be the connected component containing the query vertex q, which is a p-cohesion, we iteratively remove a vertex from S without violating the p-cohesion constraint until no such vertex exists.

Algorithm 4 shows the details of the global search algorithm. Let S be a copy of the connected component in G which contains q (Line 1). At Line 2, T ensures each vertex in S is visited once only. Let S' be a copy of S (Line 5). We select an unvisited vertex u with the largest degree in S (Line 4) and compute the p-cohesion on S after deleting u by invoking Algorithm 3 (Line 6). If the returned subgraph is empty or q is deleted from S, we recover S (Line 7). When every vertex in current S is visited, S is a minimal p-cohesion. In this paper, the vertex with the largest degree in current S will be chosen first at Line 4, because a large degree vertex may fast reduce the size of current p-cohesion (S).

```
Algorithm 4: GlobalSearch(G, p, q)
  Input  : $G$ : a graph, $p$ : a real number in (0,1), $q$ : a vertex
  Output : $C_{min}$ : a $p$-cohesion containing $q$
1 $S \leftarrow$ the connected component containing $q$ in $G$;
2 $T \leftarrow \emptyset$; $T \leftarrow T \cup \{q\}$;
3 while $\exists u \in V(S) \setminus T$ //$u$ is the vertex with the largest degree in current $S$;
4 do
5     $S' \leftarrow S$; $T \leftarrow T \cup \{u\}$;
6     $S \leftarrow$ Collapse($S, p, u$);
7     if $S = \emptyset$ or $q \notin S$ then $S \leftarrow S'$;
8 return $S$
```

```
Algorithm 5: LocalSearch($G, p, q$)
  Input  : $G$ : a graph, $p$ : a real number in (0,1), $q$ : a vertex
  Output : $C_{min}$ : a $p$-cohesion containing $q$
1 $D \leftarrow \{q\}$; $T \leftarrow \emptyset$;
2 $f(u) \leftarrow$ compute score for every vertex in $G$;
3 while $\exists v \in D \setminus T$ do
4     $T \leftarrow T \cup \{v\}$; $b \leftarrow |N(v, G) \cap D|$;
5     $P \leftarrow \{max(\lceil p \times deg(v, G) \rceil - b, 0)$ vertices in $N(v, G) \setminus D$ with the largest $f(u)\}$;
6     Update score $f(u)$;    $D \leftarrow D \cup P$;
7 $S \leftarrow$ GlobalSearch($G(D), p, q$);
8 return $S$
```

Time complexity. The visit of every vertex in $V(S)$ takes $\mathcal{O}(n)$ (Line 4). Algorithm 3 (Line 6) and the recover of S (Line 7) take $\mathcal{O}(m+n)$ for one iteration, respectively. Thus, the time complexity of Algorithm 4 is $\mathcal{O}(n(m+n))$.
Space complexity. The subgraph S, S' and the neighbor set take $\mathcal{O}(m+n)$ space, respectively. The set T and $deg(\cdot)$ take $\mathcal{O}(n)$ space, respectively. The space complexity of Algorithm 4 is $\mathcal{O}(m+n)$.

Example 3 Figure 1 shows a graph with the label of the smallest number of neighbors required for every vertex in a p-cohesion, when $p = 0.6$. If $q = u_1$, Algorithm 4 may firstly choose u_{10} at Line 4, and then, delete u_{11}, u_{12} and u_{13} by the collapse procedure (Algorithm 3). At the next loop, it deletes u_4 which leads to the deletion of all the vertices according to the constraint of p-cohesion, i.e., $S = \emptyset$. Thus, we recover S and try another unvisited vertex in S until no more vertices can be deleted. Finally, the vertices in $\cup_{1 \le i \le 5} u_i$ induce a minimal p-cohesion containing q.

Local search algorithm. Due to the giant component phenomenon [12], the connected component containing the query may occupy a large part of the graph. In such cases, Algorithm 4 is inefficient on large graphs. To improve algorithm efficiency, we can compute the minimal p-cohesion on a reduced S. In a bottom-up manner, we repeatedly expand the vertices starting from the query vertex to form a p-cohesion subgraph which may be much smaller than the initial connected component S. Algorithm 5 shows the local search procedure from q on a graph G. The set D records the to-expand vertices, and set T ensures that each chosen vertex is expanded only once (Lines 1-4). When we expand a vertex v, b is the number of v's neighbors in D (Line 4). We add $max(\lceil p \times deg(v, G) \rceil - b, 0)$ neighbors of v to D, such that v can stay in the resulting p-cohesion (Line 5).

In Algorithm 5, at Line 3, the chosen vertex v is the vertex with the largest degree in D at that time. At Line 2, we compute a score $f(u)$ for every vertex. At Line 5, we choose the vertices in $N(v, G)$ with the largest score $f(u)$ in G-$G(D)$ since the existence of such vertices can get a good trade-off between the gain and penalty effect of adding one vertex to D. All these chosen vertices are added to D for further expansion (Line 6). After the expansion of every vertex in D, the vertices in D can induce a p-cohesion subgraph. By invoking Algorithm 4 (Line 7), we can get the minimal p-cohesion containing the query vertex q.

Score function. A straightforward score definition for a vertex $u \notin D$ is the gain effect of adding u to D, denoted by $f^+(u)$, specifically, $f^+(u)$ records the number of u's neighbor v in D with $deg(v, G(D)) < \lceil p \times deg(v, G) \rceil$:

$$f^+(u) = |\{v | deg(v, G(D)) < \lceil p \times deg(v, G) \rceil, v \in N(u, G(D))\}| \tag{5}$$

Intuitively, $f^+(u)$ denotes the inclusion of u into D could contribute to increasing the degrees of some vertices in D s.t. they are closer to satisfy the p-cohesion constraint.

Besides, there is also a penalty effect of adding u to D, since u may need extra neighbors outside of D to make it have at least $\lceil p \times deg(u, G) \rceil$ neighbors in D. We denote the penalty by $f^-(u)$:

$$f^-(u) = max\{0, \lceil p \times deg(u, G) \rceil - |N(u, G(D))|\} \tag{6}$$

By considering both the gain and penalty effect, we define a score for a vertex u to determine which neighbor should be selected in Line 5 with the trade-off between its gain and penalty. The ultimate score of a vertex u is defined as:

$$f(u) = f^+(u) - f^-(u) \tag{7}$$

As for Algorithm 5, in Line 5, for an expanding vertex v, we choose its neighbors $u \notin D$ with the largest scores $f(u)$.

Example 4 Figure 1 shows a graph with the label of the minimum number of neighbors required for every vertex in a p-cohesion, when $p = 0.6$. If $u = u_1$, Algorithm 5 may firstly add u_2, u_3 and u_4 to D, s.t., u_1 satisfies the threshold for existing in a p-cohesion. Then, u_2 and u_3 are expanded with no vertex pushed into D. Then, u_4 is expanded, which adds u_5 to D, because $f(u_5) = f^+(u_5) - f^-(u_5) = 1 - 0 = 1$ is larger than $f(u_6) = 1 - 2 = -1$ and $f(u_7) = 1 - 2 = -1$. When all the vertices in D have been expanded, the algorithm returns the induced subgraph by $\cup_{1 \le i \le 5} u_i$.

Time complexity. Let $\hat{n}$ and $\hat{m}$ denote the number of vertices and edges of $G(D)$, respectively. In Line 3 of Algorithm 5, the total number of visited vertices in D is $\hat{n}$. For each vertex in D, the value b can be retrieved by visiting its neighbors, which takes $\mathcal{O}(m+n)$ for one iteration. The update of $f(\cdot)$ for the neighbors of a vertex v takes $\mathcal{O}(deg(v, G) * log(deg(v, G)))$. The retrieval of P takes $\mathcal{O}(m+n)$ for one iteration. The score computation at each iteration takes $\mathcal{O}(m+n)$ because $deg(\cdot, G(D))$ and $N(\cdot, G(D))$ can be maintained when each vertex is added to D by visiting the neighbors of the vertex, and each vertex is added to D at most once. At Line 7, Algorithm 4 takes $\mathcal{O}(\hat{n}(\hat{m}+\hat{n}))$. As $\hat{n} \le n$ and $\hat{m} \le m$, the time complexity of Algorithm 5 is $\mathcal{O}(\hat{n}(m+n))$.

Space complexity. The sets D, T, P, $f(\cdot)$ and $deg(\cdot)$ take $\mathcal{O}(n)$ space, respectively. G and $N(\cdot)$ take $\mathcal{O}(m+n)$ space, respectively. The space complexity of Algorithm 5 is $\mathcal{O}(m+n)$.

Algorithm correctness. In Algorithm 5, every vertex v in D is expanded once at Line 3, which ensures v has sufficient neighbors in the partial set D by adding enough neighbors of

v into D (Lines 5-6). When every vertex v in D has been expanded, every v in D satisfies $deg(v, G(D)) >= \lceil p \times deg(v, G) \rceil$, i.e., the returned $G(D)$ is a p-cohesion containing u. Then, we invoke Algorithm 4 in Line 7. As the correctness of Algorithm 4 is immediate (it follows the definition of p-cohesion), Algorithm 5 is correct.

4.4 Progressive search algorithm

In this section, we devise a progressive search algorithm to get a p-cohesion containing a query vertex, namely *PSA-PC*. Given a vertex set V_t as a partial solution, we can compute the upper/lower bounds of the minimum size of a p-cohesion containing V_t. When the size upper/lower bounds of the partial solution is converged, we would get a size guaranteed p-cohesion regarding the size of optimally minimum p-cohesion.

Motivated by [24], we would conduct a Best-First Search (BFS). A BFS tree would be constructed, where the root is the query vertex and every tree node contains one vertex. For each tree node t, its partial solution V_t for this node contains the vertex in the node and all vertices in its ancestor nodes. In *PSA-PC*, when a tree node t is visited and t contains the vertex u, we add the child nodes of t to the search tree where each child node contains a unique neighbor of a vertex in V_t with vertex id larger than u, in case of duplicate computation. Then, for each partial solution V_t, we compute the upper/lower bounds based on the following algorithms.

(i) **Upper bound** We conduct the LocalSearch algorithm (Algorithm 5) equipped with Eq. (7) to get a minimal p-cohesion containing V_t. The returned p-cohesion will be used to update the global size upper bound s^+ of the optimally minimum p-cohesion. Also, the returned p-cohesion will be the current best solution containing V_t;

(ii) **Lower bound** For each vertex in $u \in V_t$, we compute the number of neighbors that u need to stay in a p-cohesion: $D_u = max\{0, \lceil p \times deg(u, G) \rceil - |N(u, G(V_t))|\}$. We use $ND_{max} = max_{u \in V_t}\{D_u\}$ to denote the largest number of new vertices required to stay in the p-cohesion containing V_t. So, $|V_t| + ND_{max}$ is the size lower bound $s^-(t)$ of the optimally minimum p-cohesion containing V_t.

The algorithm *PSA-PC* will return once $\frac{s^+}{s^-} \leq c$ is satisfied, where c is a user-specified approximation ratio on the size of p-cohesion.

The pseudo-code is given in Algorithm 6. We use t to denote a tree node in the BFS tree $\mathcal{T}$. The vertex set V_t of a node t consists of the vertex $t.v$ in t and all vertices in the ancestor nodes of t. We use s^+ to denote the upper bound of optimal minimum p-cohesion and use $s^-(t)$ to denote the size lower bound of the minimum p-cohesion containing V_t. A priority queue $\mathcal{Q}$ is used to denote the leaf nodes in $\mathcal{T}$ to be visited, where the key of a node t is $s^-(t)$ in ascending order.

In each iteration (Lines 5-17), the node t with the smallest lower bound value $s^-(t)$ is popped at Line 5. For current processing $t.v$, we expand it by each neighbor u of a vertex in V_t with $id(u) > id(t.v)$ (Line 6), where $id(u)$ is the identifier of u. At Lines 7-8, for each child node t' of t, we compute the size lower bound of the minimum p-cohesion containing V_t', where V_t' contains $t'.v$ and all vertices in V_t. For the size upper bound $s^+(t')$, at Line 10, we conduct Algorithm 5 equipped with Eq. (7) to get the p-cohesion containing V_t', denoted by R', with $s^+(t') = |R'|$. The global upper bound s^+ and current best solution R will be updated by $s^+(t')$ and R' if $s^+(t') < s^+$ (Lines 11-12). At Line 15, a search branch following t' can be stopped if $s^-(t') \geq s^+$ because the size of the p-cohesion containing current V_t' cannot be smaller than current solution $|R|$.

Algorithm 6: PSA-PC(G, p, q, c)

```
Input  : $G$ : a graph, $p$ : a real number in (0, 1),
         $c$ : approximation ratio, $q$ : query vertex
Output : $R$ : the approximate minimum $p$-cohesion
1  $t \leftarrow$ the (root) node of search tree $\mathcal{T}$, where $t.v = q$;
2  $\mathcal{Q}.push(t)$; $R :=$ **LocalSearch**($G$,$p$,$V_t$);
3  $s^+ := |R|$; $s^-(t) := 1$;
4  while $\mathcal{Q} \neq \emptyset$ do
5      $t \leftarrow \mathcal{Q}$.pop(); //$\mathcal{Q}$ is a priority queue with key on $s^-(t)$;
6      for every $u \in N(t.v)$ with $id(u) > id(t.v)$ do
7          $t' \leftarrow$ the child node of $t$, where $t'.v := u$;
8          $s^-(t') := max_{u \in V_{t'}} \{max\{0, \lceil p \times deg(u, G) \rceil - |N(u, G(V_{t'}))|\}$;
9          if $s^-(t') < s^+$ then
10             $R' :=$ **LocalSearch**($G$,$p$,$V_{t'}$); $s^+(t') := |R'|$;
11             if $s^+(t') < s^+$ then
12                 $R := R'$; $s^+ := s^+(t')$;
13             $\mathcal{Q}$.push($t'$); attach child node $t'$ to $t$ in $\mathcal{T}$;
14         else
15             $s^-(t') \leftarrow +\infty$ ;
16     $s^- \leftarrow$ smallest key value among nodes in $\mathcal{Q}$ ;
17     if $\frac{s^+}{s^-} \leq c$ then return $R$;
18 return $R$
```

At Line 16, the global lower bound s^- is updated as the smallest $s^-(\cdot)$ among all nodes in $\mathcal{Q}$. The algorithm will return when $\frac{s^+}{s^-} \leq c$ is satisfied at Line 17 or the queue is empty.

Algorithm Correctness. Every subgraph R' retrieved at Line 1 is a p-cohesion containing q, and hence, the upper bound s^+ is correctly maintained in Algorithm 6. Given the correctness of the lower bound s^-, we have $s^- \leq |R^*| \leq s^+$, where R^* is the optimal solution. When Algorithm 6 terminates, we will return current best solution R with $\frac{s^+}{s^-} \leq c$.

5 Diversified enumeration

In this section, we study the diversified p-cohesion enumeration problem (DPCE): Enumerate a set of disjoint minimal p-cohesions.

5.1 Problem analysis

Firstly, we show the number of minimal p-cohesion for a graph can be exponential by the following theorem.

Theorem 4 *There exists a graph G that contains an exponential number of minimal p-cohesions, for every fixed $p \in (0, 1)$.*

Proof We prove the theorem based on the exponential number of maximal cliques in a graph G. Suppose G is empty initially, we add a vertex set $O = \cup_{1 \leq i \leq n} v_i$ to G where $n = 2x$ and $x \in N^+$. For every vertex $v_i \in O$, we connect v_i to every other vertex in O except the opposite vertex v_j where $|j - i| = \frac{n}{2}$, i.e., the degree of every vertex $v \in O$ is $deg(v, G) = n - 2$.

For every vertex $v \in O$, we add y extra vertices and connect each of them to v:

$$y = \begin{cases} \lceil (n-2) \cdot \left(\frac{1}{2p} - 1\right) \rceil, & 0 < p < \frac{1}{2} \\ \lceil (n-2) \cdot \left(\frac{1}{2(1-p)} - 1\right) \rceil, & \frac{1}{2} \leq p < 1 \end{cases} \tag{8}$$

The construction is completed. We have $|V(G)| = n + n \cdot y$. For every vertex $v \in O$, we have $deg(v, G) = (n-2) + y$:

$$deg(v, G) = \begin{cases} \lceil \frac{(n-2)}{2p} \rceil, & 0 < p < \frac{1}{2} \\ \lceil \frac{(n-2)}{2(1-p)} \rceil, & \frac{1}{2} \leq p < 1 \end{cases} \tag{9}$$

For both cases of p, we have $p \cdot deg(v, G) \geq \frac{n-2}{2}$. So for each vertex in O to be in a minimal p-cohesion S, at least $\frac{n-2}{2}$ of its neighbors are also in S. Note that, for a maximal clique of $G(O)$, the degree of every vertex in the clique is $\frac{n-2}{2}$. If we count each minimal p-cohesion in which each vertex contained in O has exactly $\frac{n-2}{2}$ neighbors in O, the number of the minimal p-cohesions is at least the number of maximal cliques in $G(O)$ which is $2^{n/2}$. Thus, the number of minimal p-cohesions in G is exponential. □

According to Theorem 4, the number of minimal p-cohesions may be overwhelming to users. Moreover, the p-cohesions discovered may heavily overlap with each other. Thus, we are interested in tackling the diversified minimal p-cohesion enumeration problem (DPCE) for graph G: We prefer to find a set of disjoint minimal p-cohesions for G.

5.2 Pivot-based local search (PLS)

In this section, we efficiently find a set of disjoint minimal p-cohesions based on the algorithms for MPCS.

Baseline algorithm. A straightforward method to find a set of disjoint p-cohesions for a graph is to repeatedly find a minimal p-cohesion and remove it. Thus, we propose an algorithm in a top-down manner: For a graph G, starting with a connected component S, we can find a minimal p-cohesion by Algorithm 4: "$C_p = GlobalSearch(S, p, \emptyset)$." We remove C_p and the vertices violating the p-cohesion constraint after the removal of C_p. Repeatedly when all vertices are removed from graph G, we can get a set of disjoint minimal p-cohesions.
A pivot-based local search algorithm Finding one minimal p-cohesion in a top-down manner without a pivot is time-costly. Motivated by the MPCS, we can improve the efficiency of the `BaseLine Algorithm` by adding a pivot u and finding a minimal p-cohesion containing u by Algorithm 5. The details are shown in Algorithm 7.

At Line 1, we record the degree of every vertex in G. The set T is used to ensure every vertex is checked exactly once. Then, we compute a minimal p-cohesion on each connected component S of current graph G from Line 2. A minimal p-cohesion C_p can be computed by Algorithm 5 with subgraph S, threshold p and a vertex $u \in S$. We delete C_p from S and record C_p in C (Line 6). We delete the vertices violating the fraction threshold of p-cohesion in S to further reduce S (Lines 7-8). Algorithm 7 returns the set of minimal p-cohesions in C.

At Line 3, we select the vertex with the smallest degree in S because a small degree pivot can fast expand to a p-cohesion and keep the advantage of the pivot that prunes more vertices at the early stage. Besides, the chosen pivot allows us to compute the minimal p-cohesion on a reduced initial S at Line 5 of Algorithm 7, which can improve the efficiency significantly.

Algorithm 7: **PLS**(G, p)

```
  Input   : $G$ : a graph, $p$ : a real number in (0,1)
  Output  : $C$ : a set of disjoint minimal $p$-cohesions
1 $deg(v) \leftarrow deg(v, G)$ for every $v \in V(G)$; $T \leftarrow \emptyset$;
2 while $\exists$ a non-empty connected component $S \in G$ do
3     while $\exists u \in S \setminus T$ do
4         $T \leftarrow T \cup \{u\}$;
5         $C_p \leftarrow$ LocalSearch($S, p, u$);
6         $S \leftarrow S$-$C_p$; $C \leftarrow C \cup \{C_p\}$;
7         while $\exists v \in V(S)$ with $deg(v, G) < \lceil p \times deg(v) \rceil$ do
8             $S \leftarrow S \setminus \{v \cup E(v)\}$;
9 return $C$
```

Time complexity. The visit of the connected components in the graph takes $\mathcal{O}(n)$ at most (Line 2). Algorithm 5 takes $\mathcal{O}(n(m + n))$ for one iteration (Line 5). The update of S and C takes $\mathcal{O}(m+n)$ at most (Lines 6-8). Thus, the time complexity of Algorithm 7 is $\mathcal{O}(n^2(m+n))$ in the worst case.

Space complexity. Algorithm 5 takes $\mathcal{O}(m + n)$ space. The G, C_p and C take $\mathcal{O}(m + n)$ space. The $deg(\cdot)$ and T take $\mathcal{O}(n)$ space. The space complexity of Algorithm 7 is $\mathcal{O}(m+n)$.

5.3 An application on MinSeed

In this section, we study an application of the minimal p-cohesion subgraphs on the contagion model introduced by [12,29] along with the p-cohesion. To promote the sale of a product B, the company may give incentives to some seed users, such as a discount or free product trial. These seed users are regarded as activated (i.e., influenced) for using B, which may influence (i.e., activate) their friends to use B. The influence will further cascade to the friends of the activated users. In the following, we formally introduce the cascading rule.

Definition 4 **Cascading Rule**. Given a graph G, the set of activated vertices $A \in V(G)$ and a cascading threshold $r \in (0, 1)$, we have: (1) a vertex $u \in (V(G) \setminus A)$ is immediately activated iff there are at least $\lceil r \times deg(u, G) \rceil$ activated neighbors of u in G, i.e., $deg(u, G(A)) \geq \lceil r \times deg(u, G) \rceil$, and (2) a seed vertex u is always activated.

Given a target user group (which induces a graph) to cascade, a company may wish to find the fewest seed users (initial adopters) such that all the target users are activated while the promotion expense is minimized. Thus, the MinSeed problem is defined.
MinSeed problem Given a target graph G with no activated vertices, and a cascading threshold $r \in (0, 1)$, the MinSeed problem is to find a set of seed vertices D in $V(G)$, such that (1) all the vertices in $V(G)$ are activated by applying the *Cascading Rule* repeatedly, and (2) $|D|$ is minimized.

We prove that MinSeed is NP-hard by a reduction from Vertex Cover problem. Given a graph G, when r is large enough, e.g., $r = |V(G) - 1|/|V(G)|$, the activation of a vertex needs all its neighbors to be activated first. To activate all the vertices in G, each edge in G should be incident to at least one seed vertex. Thus, MinSeed with such a r is exactly Vertex Cover which is NP-hard.

Algorithm 8: **SeedSelection**(G, r)

```
Input    : $G$ : a target graph, $r$ : a real number in (0,1)
Output   : $D$ : the seed set
1  $c(v) \leftarrow 0$ for every $v \in V(G)$;
2  $D \leftarrow \emptyset$; $A \leftarrow \emptyset$;
3  while $\exists u \in V(G) \setminus \{D \cup A\}$ do
4      $D \leftarrow D \cup \{u\}$; $L \leftarrow \{u\}$;
5      for each $u \in L$ do
6          for each $v \in N(u, G) \setminus \{D \cup A\}$ do
7              $c(v) \leftarrow c(v) + 1$;
8              if $c(v) \geq \lceil r \times deg(v, G) \rceil$ then
9                  $L \leftarrow L \cup \{v\}$; $A \leftarrow A \cup \{v\}$;
10 return $D$
```

Heuristic seed selection. Algorithm 8 shows the basic framework to find an approximate solution for MinSeed. For every vertex v in G, we use $c(v)$ to record the number of activated neighbors (Line 1). Set D records the selected seeds, and set A records the activated vertices except the seeds (Line 2). We select an inactivated vertex u as a seed and use L to record the activated vertices by seed u (Lines 3-4). We update the $c(\cdot)$ value for the inactivated neighbors of each vertex in L (Lines 6-7). An inactivated neighbor v is activated if $c(v) \geq \lceil r \times deg(v, G) \rceil$ (Lines 8-9). The algorithm returns D as the seed set.

Time complexity. Algorithm 8 activates each vertex in G by exactly one time and updates the $c(\cdot)$ value of its neighbors. So the time complexity of Algorithm 8 is $\mathcal{O}(m + n)$ if the seed selection at Line 3 takes up to $\mathcal{O}(m + n)$.

Space complexity. Sets $c(\cdot)$, D, A, L and $deg(\cdot)$ take $\mathcal{O}(n)$ space, respectively. G and $N(\cdot)$ take $\mathcal{O}(m + n)$ space, respectively. The space complexity of Algorithm 8 is $\mathcal{O}(m + n)$.
Algorithm correctness. Every vertex in G is either pushed into D as a seed, or pushed into A as an activated vertex by the seeds. Since the seeds are regarded as activated, the D returned by Algorithm 8 is a feasible solution of MinSeed.

Selection order. The vertex selection order in Line 3 decides the number of resulting seeds. Because the minimal p-cohesions prevent outside influence spread according to the Fortress property, the vertices in the p-cohesions are relatively isolated from the non-p-cohesion vertices. By giving certain priorities to fortress vertices, we may break through the barrier of influence spread from the non-p-cohesion vertices to the p-cohesion vertices. Thus, the number of seeds required may be reduced. For MinSeed, we explore the following seed selection orders.

IC-Deg selection. We select a vertex u with the largest degree in G at Line 3, because such a u can increase the $c(\cdot)$ values by a great extent and such a u can relax the fortress property when it is in a minimal p-cohesion.

IC-Core selection. The coreness of a vertex u is the largest value of $k_{\max}$ such that u is in the $k_{\max}$-core, i.e., $u \in k_m$-core and $u \notin (k_{\max} + 1)$-core. We select a vertex u with the largest coreness in G at Line 3, because the coreness of a vertex reflects its importance/influence [24].

IC-Truss selection. The trussness of a vertex u is defined as the largest value of trussness of an edge incident to u. The trussness of an edge e is the largest value of k_m such that e is in the k_m-truss, i.e., $e \in k_m$-truss and $e \notin (k_m + 1)$-truss. Similar to the IC-Core Selection, we select a vertex u with the largest trussness in G at Line 3, because the trussness of a vertex also reflects the importance/influence of the vertex [38].

IC-BF selection. We select a vertex u at Line 3 such that the size increase of A is the largest with the selection of u. Such a u is "best" in a greedy view. When there are ties, we choose the one with the largest degree in G, as in the degree-based selection. We hope these "best" vertices can effectively break the boundaries of the fortresses.

IC-PC selection. We find that the vertices with extremely large degrees have great influence power. So firstly we select $\alpha \times |V(G)|$ vertices as seeds which have the largest degrees in G. Then, we retrieve a set C of minimal p-cohesions on G-$G(D \cup A)$ by the `PLS` algorithm where $p = 1 - r + \varepsilon$ and ε is an infinitesimal positive number. We compute a weight $\beta \times deg(u, G)$ for each vertex u in C. For a vertex v not in C, its weight is just $deg(v, G)$. Then, we continue to select a vertex u as a seed which has the largest weight in G at Line 3. In this way, the fortress property may be relaxed by giving priorities to the vertices in the minimal p-cohesions.

Example 5 Figure 1 shows a graph where $r = 0.5$ and all the users are inactivated. By IC-Deg, Algorithm 8 may select u_{10} and u_4 sequentially. By IC-BF, Algorithm 8 may select u_{10}, u_7 and u_5 sequentially. By IC-PC, when $\alpha = 0.01$ and $\beta = 2$, Algorithm 8 may select u_4 and u_5.

6 Experimental evaluation

This section evaluates the effectiveness and efficiency of all techniques through comprehensive experiments.
Algorithms. To the best of our knowledge, there is no existing work investigating the p-cohesion computation. We implement and evaluate 6 algorithms for p-cohesion computation and 3 algorithms for MinSeed as shown in Table 2.

Datasets. 8 real-life graphs are deployed in our experiments. The original data of *Yelp* are from [40], *DBLP* is from [23], and the others are from [22]. In *DBLP*, each vertex represents an author and each edge between two authors represents the two authors have at least 5 co-authored papers. The other datasets have existing vertices and edges. For *Gowalla* and *Brightkite*, we remove the vertices without check-ins and their incident edges. We transfer directed edges to undirected edges. Table 3 shows the statistics of the datasets.

Settings. All programs are implemented in standard C++ and are compiled with G++ in Linux. All experiments are performed on a machine with Intel Xeon 2.3GHz CPU and Redhat Linux system. The runtime of an algorithm is set to INF if it cannot finish in 1 hour.

6.1 Statistical results

Exact number of p-cohesions. We can enumerate the minimal p-cohesions in a graph by `ExactPC` (Algorithm 1). Due to the huge time and space cost, we only compute the minimal/minimum p-cohesions on a small graph G with $p = 0.6$. We extract the small graph G with 70 vertices from the *Yelp* dataset. Note that the minimal p-cohesion computation on a graph with 80 vertices cannot finish in one week. For the search of the minimum p-cohesion containing a query vertex, we report the results from `SubPCExact` (Algorithm 2) over 100 independent tests. In each test, we randomly select a vertex from G as the query vertex q and

Table 2 Summary of Algorithms

Algorithm	Description
ExactPC	Finding the minimum p-cohesion containing a query vertex q by Algorithm 2
SubPCExact	Enumerating all minimal p-cohesions by Algorithm 1
GlobalS	Finding a minimal p-cohesion containing a query vertex q by Algorithm 4
LocalS	Local search algorithm (Algorithm 5) equipped with Equation (5)
LocalS* *	Local search algorithm (Algorithm 5) equipped with Equation (7)
PSA-PC	Progressive search algorithm (Algorithm 6)
BaseTD	The top-down heuristic algorithm to find disjoint minimal p-cohesions (Sect. 5.2)
BaseTD+	BaseTD equipped with the pivot vertices
PLS	The pivot-based local search algorithm (Algorithm 7)
IC-Deg	Algorithm 8 equipped with the degree-based vertex selection
IC-Core	Algorithm 8 equipped with the coreness-based vertex selection
IC-Truss	Algorithm 8 equipped with the trussness-based vertex selection
IC-BF	Algorithm 8 equipped with the best first vertex selection
IC-PC	Algorithm 8 equipped with the minimal p-cohesion-based vertex selection

Table 3 Statistics of Datasets

Dataset	Vertices	Edges	d_{avg}	$d_{\max}$
Email	36,692	183,831	10	1383
Brightkite	50,111	194,090	7.7	1098
DBLP	210,840	363,299	3.4	159
Epinion	75,879	405,740	10.7	3044
Gowalla	99,563	456,830	9.2	9967
Deezer	54,573	498,202	18.3	420
Amazon	334,863	925,872	5.5	549
Yelp	249,440	1,781,908	14.3	3812

compute all the minimal p-cohesions containing q by Algorithm 2. The results show that the average number of minimal p-cohesions containing a query q is 16, 549.27, and the size of a minimal p-cohesion is in average 1.194 times the size of (exact) minimum p-cohesion. We also compute the number of all the minimal p-cohesions in G with 70 vertices by Algorithm 1, i.e., "*SubPCExact*$(\emptyset, V(G), \emptyset, p, |V(G)|)$." The number of minimal p-cohesions of G is 87, 429, which is over-whelming.

Score function evaluation for $MPCS$. Figure 4 reports the average minimal p-cohesion subgraph size returned by GlobalS, LocalS and LocalS* over 100 runs. One query vertex is selected randomly from all the vertices. Figure 4a shows the result on 8 datasets with $p = 0.6$. The score function-based algorithm LocalS* can significantly outperform GlobalS regarding average size, because the search space of GlobalS is larger. Some large degree vertices may be chosen in the execution of GlobalS, and they usually need more neighbors to stay in a p-cohesion. On most of the datasets, LocalS* can significantly outperform LocalS. For example, on *DBLP*, the number of vertices returned by LocalS is almost 5 times greater than the number of vertices returned by LocalS*. This implies that considering only the gain effect (Eq. (5)) during the expansion procedure cannot guarantee

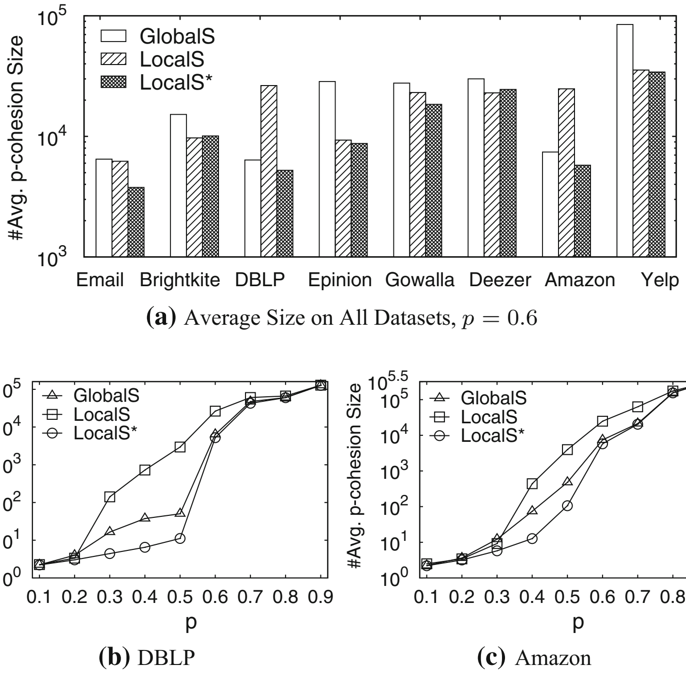


Fig. 4 Average Size of Minimal p-Cohesions

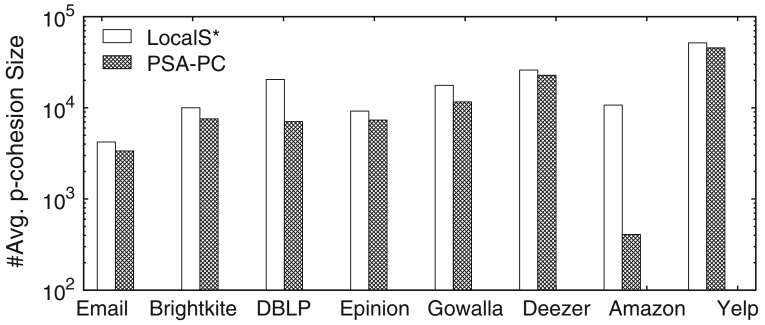


Fig. 5 Average Size of Minimal p-Cohesions, $p = 0.6$, $c = 1.8$

a good performance. Figure 4b, c reports the results of three algorithms by varying the constraint p from 0.1 to 0.9 on *DBLP* (4b) and *Amazon* (4c), respectively. In both figures, the sizes of the minimal p-cohesion subgraphs returned by `GlobalS`, `LocalS` and `LocalS`* increase as p grows, since a large p inherently requires more vertices in a p-cohesion subgraph. When p is very large, the results of the three algorithms are similar, because nearly the whole graph is returned.

Size guaranteed p-cohesion. We can get a minimal p-cohesions with certain size guarantee by *PSA-PC* (Algorithm 6). Figure 5 shows the average size of p-cohesion subgraphs on 8 datasets, where $p = 0.6$ and approximation ratio $c = 1.8$. As depicted in Fig. 4, the `LocalS`* outperforms the other two algorithms. Thus, in Fig. 5, we compare the results

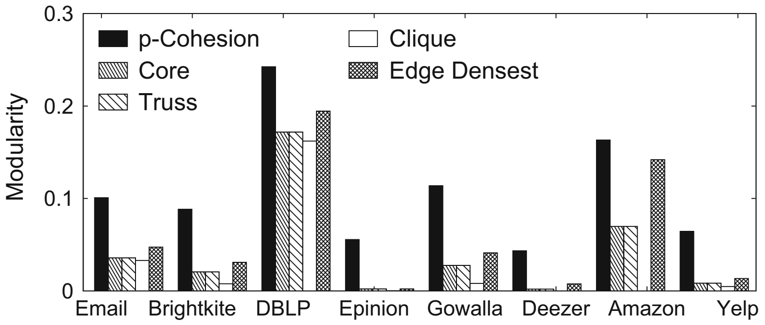


Fig. 6 Comparing Modularity of Different Cohesive Subgraph Models, $p = 0.6$

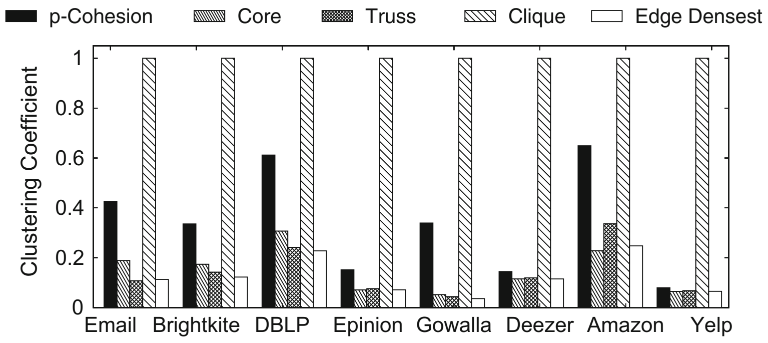


Fig. 7 Comparing Clustering Coefficient of Different Cohesive Subgraph Models, $p = 0.6$

returned by *PSA-PC* and LocalS* over 100 independent tests. In each test, we randomly select a vertex from a graph G as the query vertex q and compute the minimal p-cohesions containing q by *PSA-PC* and LocalS*, respectively.

As *PSA-PC* is costly, it cannot finish in one hour for about 87% queries in the experiments. Thus, we terminate *PSA-PC* when the runtime reaches one hour and get the latest p-cohesion from it. Given such time limit, the returned results from PSA-PC can outperform LocalS* regarding the average size from all the queries, due to the theoretical guarantee of PSA-PC.

6.2 Effectiveness on cohesive subgraph modeling

Comparing different cohesive subgraphs. For each query vertex v, we compute the minimal p-cohesion by LocalS*, the $\lceil p \times deg(v, G) \rceil$-core by [6], the $(\lceil p \times deg(v, G) \rceil + 1)$-truss by [9], the maximal clique by [36] and the edge densest subgraph by [14] where every computed subgraph contains v. The query vertex is randomly selected from all the vertices in the graph. In Fig. 6, we report the modularity scores [30] on the evaluated subgraph and the subgraph of the outside, for each of the above models, when $p = 0.6$. The scores are the average values from 100 independent tests. Consistent with the definition, the minimal p-cohesion shows better modularity scores because it holds both inner-cohesiveness and outer-sparseness. In Fig. 7, we report the clustering coefficient of the induced subgraphs computed by each model. Figure 7 shows the p-cohesions have higher clustering coefficients on all the datasets than other models except the clique.

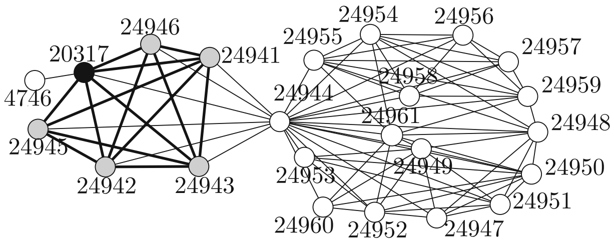


Fig. 8 A Minimal p-cohesion on *Email*

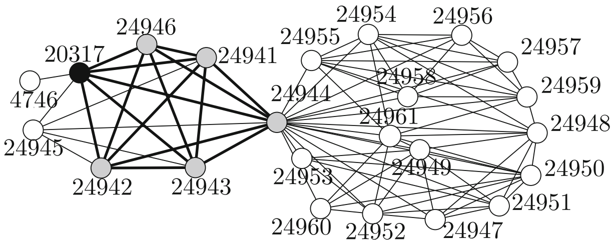
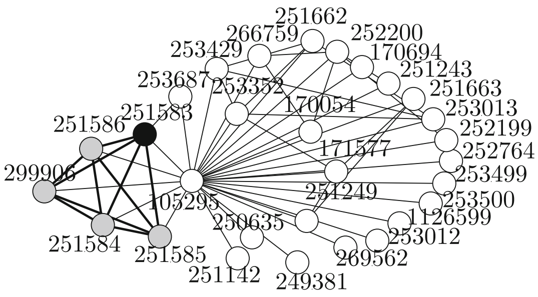


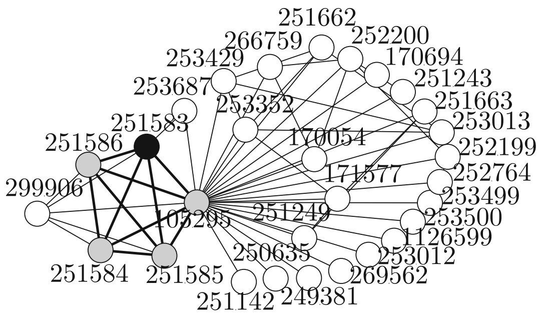
Fig. 9 An s-Clique on *Email*

Fig. 10 A Minimal p-cohesion on *DBLP*



Cases of $\underline{MPCS}$. In Fig. 8, we depict a minimal p-cohesion S containing the query vertex "20317" found by `LocalS`* with $p = 0.6$ on *Email*. The p-cohesion contains all the grey vertices. In Fig. 9, we find a s-clique containing the query vertex "20317" where $s = |V(S)|$. The s-clique contains all the gray vertices. We also depict all the 1-hop neighbors of one vertex in the p-cohesion or the s-clique, to show the outer connections to the subgraph. In Fig. 8, we can see the vertices in S are sparsely connected to their outside neighbors. However, in Fig. 9, vertex "24944" has a dense connection with its neighbors outside of the s-clique. The outside influence cascades may enter the s-clique through "24944." We also compute the $\lceil p \times deg(v, G)\rceil$-core and the $(\lceil p \times deg(v, G)\rceil{+}1)$-truss containing vertex "20317." The numbers of vertices in two subgraphs are 11,538 and 10,097, respectively. The sizes are too large for community-oriented applications. For each subgraph, there are some vertices with many neighbors outside the subgraph. In Figs. 10 and 11, we depict the minimal p-cohesion S found by `LocalS`* with $p = 0.6$ and a $|V(S)|$-clique containing the query vertex "251583" on *DBLP*. We also compute the $\lceil p \times deg(v, G)\rceil$-core and the $(\lceil p \times deg(v, G)\rceil{+}1)$-truss containing vertex "251583," and the sizes of these two subgraphs are 35,281 and 22,499, respectively. The results on *DBLP* are similar to that on *Email*.

Fig. 11 An s-Clique on *DBLP*



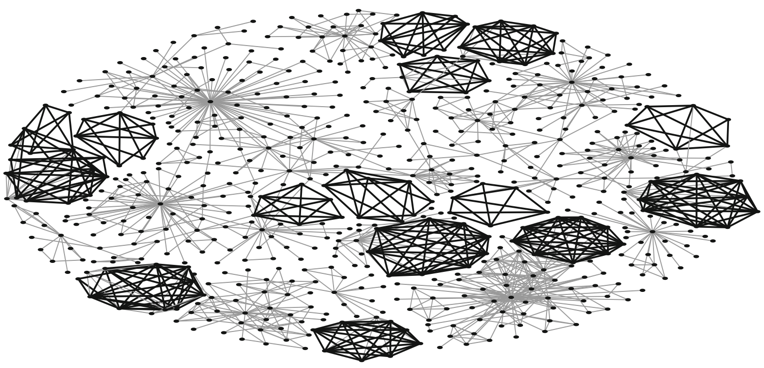


Fig. 12 PLS on DBLP, $p = 0.8$

Cases of $DPCE$. Figure 12 depicts a part of DBLP with the minimal p-cohesions discovered by PLS when $p = 0.8$. The p-cohesions are marked by the black edges and their incident vertices. We also show all the vertices which are within the 5-hop neighborhood of one vertex in the p-cohesions. We can see the minimal p-cohesions have a loose connection to their outside neighbors.

6.3 Effectiveness on fortress property

Fortress property on different influence models. We examine the fortress property of p-cohesions under different cascade models: the contagion model, the independent cascade (IC) model and the linear threshold (LT) model [18]. The target minimal p-cohesions are computed by PLS. For each influence test, we randomly choose $b = 0.1 \times |V(G)|$ seeds. We follow the settings in [18] to compute the influence spread. During the cascade procedures of the LT and IC models, each edge (u, v) is split to two directed edges (u, v) and (v, u). The existing probability of each directed edge, i.e., (u, v), is set as $1/deg(v)$. For the LT model, we randomly distribute a threshold from 0 to 1 for each vertex v as the influence threshold. We generate 10 possible worlds on both the IC and LT models, to compute the influenced vertices and generate the average influence ratios.

In Fig. 13, Contagion, IC and LT represent the average influence ratio of "the number of influenced p-cohesion vertices" divided by "the number of all the influenced vertices" for contagion, IC model and LT model, respectively. We use BaseValue, i.e., $|V(C_p(G))|/|V(G)|$, to represent "the number of p-cohesion vertices" divided by "the number of all the vertices,"

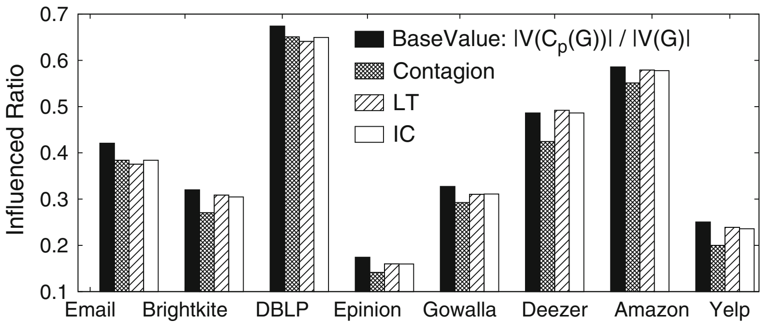


Fig. 13 Fortress Property of Different Cascade Models

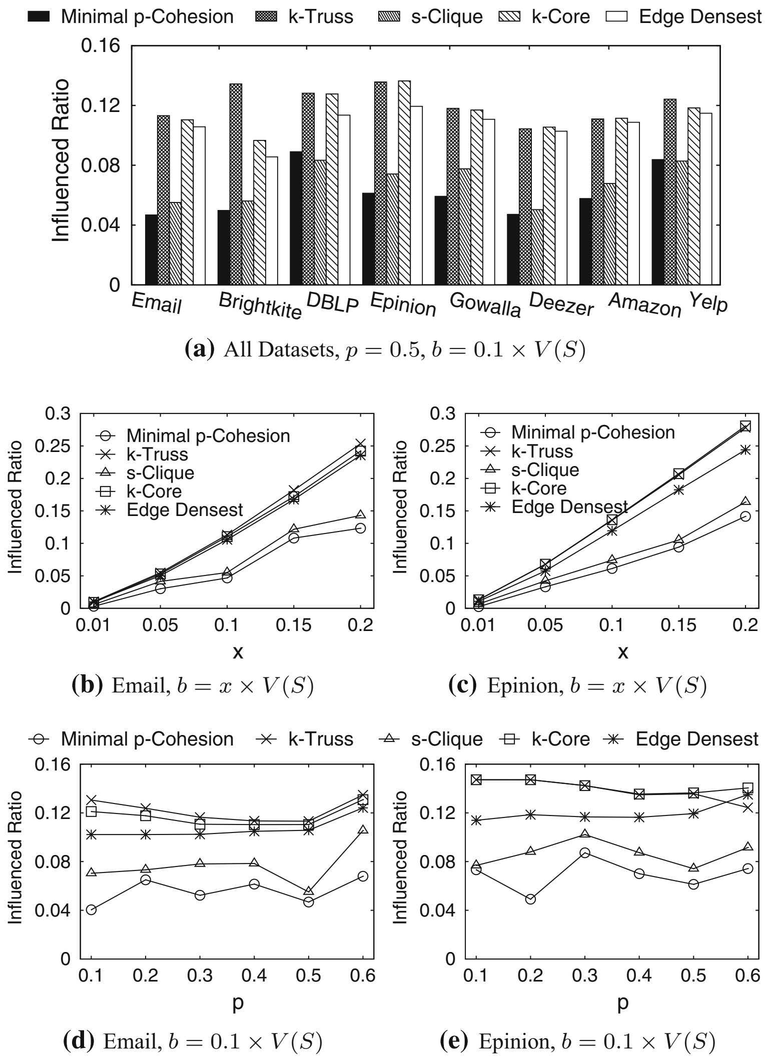
as a base value for influence comparison. The setting is $p = 0.6$ and $r = 1 - p + 0.001$. `BaseValue` is basically larger than `Contagion`, `IC` and `LT`, among different settings, which implies that the vertices in a p-cohesion have a relatively smaller possibility to be influenced than the other vertices, under every evaluated influence model. The p-cohesion holds the fortress property on all the evaluated influence models, because a p-cohesion maintains a sparse connection to the outside s.t. the influence spread from outside is hard to enter the p-cohesion on these influence spread models.

Figure 13 also shows that the percentages of p-cohesion vertices over different graphs are different, which would affect the influence power of contagion model. For example, about 17.4% of the vertices from *Epinion* are in p-cohesions, and only about 14.1% of them can be influenced under the contagion model, while about 67.4% of the vertices of *DBLP* are in p-cohesions, and almost 65.1% of them can be influenced under the contagion model. It indicates that the higher occupation of the p-cohesion vertices would result in the higher influence power of the contagion model.

Fortress property on different cohesive subgraphs. In Fig. 14, we evaluate the fortress property of the minimal p-cohesion, which is compared with different cohesive subgraph models: k-core [33] and s-clique [20], k-truss [9] and the edge densest subgraph [14]. The query vertex is randomly selected on the graph. The minimal p-cohesion is computed by `LocalS`*. For the s-clique containing the query vertex, we choose the size of the minimal p-cohesion as the parameter s. For k-core (resp. k-truss), the input of k is the largest value of k such that the k-core (resp. k-truss) containing the query vertex is not empty.

We report the average ratio of influenced (i.e., activated) vertices over all the vertices under the contagion model [12,29] over 100 independent tests with $p = 0.5$ and influence ratio $r = 1.0 - p + 0.001$. For each influence test, we randomly choose $b = x \times |V(S)|$ seeds where S is the connected component containing the query vertex in the graph. In Fig. 14, `Minimal p-Cohesion` represents the average ratio of "the number of influenced vertices in the p-cohesion (denoted by S)" divided by "the number of all the vertices in S." It is similar for `s-Clique`, `k-Truss`, `k-Core` and `Edge Densest`. The figure shows that all other cohesive subgraphs have more difficulty in hindering the influence spread from outside, compared with p-cohesion, because they do not guarantee the outer-sparseness. The minimal p-cohesion shows a stronger fortress property than the others on all the settings.

Evaluation of minimal and non-minimal p-cohesions. Figure 15 evaluates the effectiveness of "minimal" constraint on p-cohesion, by comparing the fortress property of minimal p-cohesions and non-minimal p-cohesions. Each time for a random query vertex in G, we compute a minimal p-cohesion by `LocalS`* and a corresponding non-minimal p-cohesion



(a) All Datasets, $p = 0.5, b = 0.1 \times V(S)$

(b) Email, $b = x \times V(S)$

(c) Epinion, $b = x \times V(S)$

(d) Email, $b = 0.1 \times V(S)$

(e) Epinion, $b = 0.1 \times V(S)$

Fig. 14 Fortress Property of Different Cohesive Subgraphs

by the expansion procedure of `LocalS*`. We conduct 100 influence tests under the contagion model. In each test, we randomly choose $b = 0.1 \times |V(S)|$ seeds to compute the influenced vertices, for 10 times, where S is the connected component containing the query vertex.

In Fig. 15, `Minimal p-Cohesion` represents the average ratio of “the number of influenced vertices in the minimal p-cohesion (denoted by S)” divided by “the number of all the vertices in S.” `Non-Minimal p-Cohesion` represents the average ratio of “the number of influenced vertices in the non-minimal p-cohesion (denoted by S')” divided by “the number of all the vertices in S'.” The minimal p-cohesion shows stronger fortress property than the non-minimal p-cohesion, because the smaller diameter and size may benefit the defend of information cascades coming from the outside.

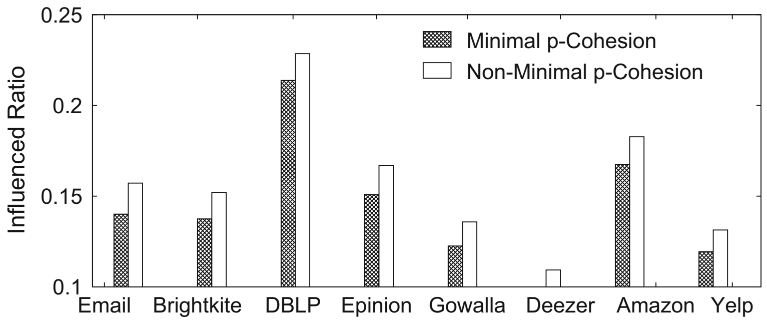


Fig. 15 Minimal vs Non-Minimal p-Cohesions

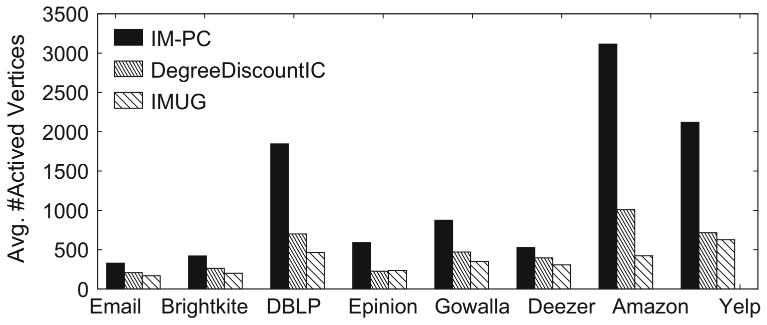


Fig. 16 Influence Maximization of Different Methods

6.4 Effectiveness on influence maximization and MinSeed problems

Evaluation on influence maximization. To further verify the effectiveness of the fortress property, we utilize the p-cohesions in the problem of influence maximization, compared with existing works [8,28]. The minimal p-cohesions used in the experiment are computed by PLS with $p = 0.6$. We follow the settings in [28] to compute the influence spread. The IC model is used as the influence cascade model, and the influence spread probability $r = 1 - p = 0.4$. We report the result on $R = 100$ rounds where we select $b = 50$ inactive vertices as the seeds in each round.

Specifically, during the seed selection of our IM-PC on each round, we compute the influence spread with the following steps: (1) Choose an inactive vertex from all minimal p-cohesions with the largest degree as the seed; (2) compute the spread of the influence of this seed; (3) repeat step (1) and step (2) for b times; and (4) report the average number of active vertices for all rounds. For the method IMUG [28], we probe $m = \lceil 0.001 * |V(G)| \rceil$ vertices and select b seeds for each round s.t the expected number of active vertices in the R-th round is maximized [28]. Since the IMUG is for unknown graph, we probe the vertex with the largest degree as a seed, because such a vertex has a larger influence spread than other heuristics [18]. The DegreeDiscountIC [8] is based on degree discount method, which select the vertices with the largest number of inactivated neighbors as the seeds.

In Fig. 16, we report the average number of active vertices for different seed selection methods. The results show that the number of activated vertices by IM-PC is larger than DegreeDiscountIC and IMUG. It is because that the vertices in a minimal p-cohesion

Table 4 Seed Numbers of *Email* and *Brightkite* with Different Methods

r	Email					Brightkite				
	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC
0.2	1129	1130	1130	1140	1129	457	459	461	459	457
0.4	2410	2397	2377	2343	2083	1262	1291	1338	1310	1253
0.6	4925	5107	5153	4909	2156	7699	8280	8442	7382	1312
0.8	11,145	11,426	11,498	10,535	2292	17,501	18,032	18,513	15,960	1264

Table 5 Seed Numbers of *Gowalla* and *Amazon* with Different Methods

r	Gowalla					Amazon				
	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC	IC-Deg	IC-Core	IC-Truss	IC-BF	IC-PC
0.2	1088	1088	1091	1102	**1088**	1126	1119	1112	1125	1126
0.4	2722	2714	2700	2902	2692	21,122	21,445	21,487	–	4549
0.6	16,131	17,123	17,082	15,487	3023	76,215	77,633	78,845	–	4013
0.8	39,264	40,663	41,358	–	3043	149,177	150,616	155,377	–	4115

are hard to be influenced than other vertices, and the seed from p-cohesions (IM-PC) can break the entry barriers of the p-cohesions, while DegreeDiscountIC and IMUG do not consider this fortress property.

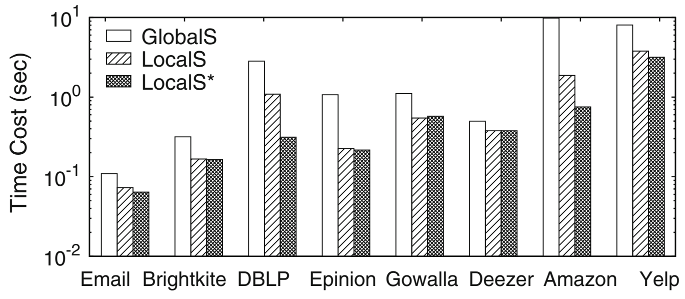
Evaluation of*MinSeed* algorithms. In Table 4 and Table 5, we report the seed numbers returned by IC-Deg, IC-BF, IC-Core, IC-Truss, and IC-PC when r varies from 0.2 to 0.8. We report the seed numbers on four datasets, i.e., *Email*, *Brightkite*, *Gowalla* and *Amazon*. According to Property 1, it is difficult for the p-cohesions to be influenced from the outside vertices. Our IC-PC gives selection priority to the vertices in minimal p-cohesions such that the resulting seed numbers can be reduced. We set ε as 0.001 and $p = 1 - r + \varepsilon$. According to statistical observation, we set $\alpha = 0.01$ and $\beta = 2$.

Tables 4 and 5 show that the IC-PC significantly selects less seeds than the other algorithms, which helps to reduce the cost of cascading the network. Some results of IC-BF are not reported because the computation cannot finish within 1 week. All methods of IC-Deg, IC-Core, IC-Truss and IC-PC are efficient, because IC-PC just additionally conducts PLS on a reduced graph and the PLS is efficient. We observe that the seed number is much smaller than the size of a dataset, e.g., the seed number of IC-PC on *Brightkite* (resp. *Gowalla*) is only 2.5% (resp. 2.7%) of its vertex number when $r = 0.4$.

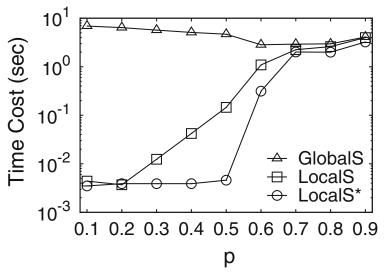
6.5 Efficiency report

In this section, we report the runtime of finding a minimal p-cohesion and finding the disjoint minimal p-cohesions.

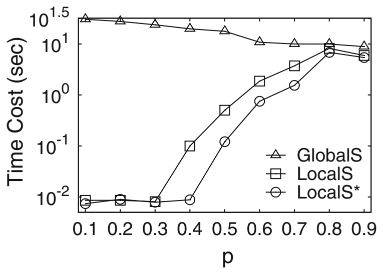
Score function evaluation on *MPCS*. Here, we report the average runtime of GlobalS, LocalS and LocalS* to compute a minimal p-cohesion containing a query vertex q over 100 runs. One query vertex is randomly selected among all the vertices. Figure 17a reports the runtime on all the datasets when $p = 0.6$. We observe that the runtime on a dataset is strongly affected by its vertex number, since every vertex belongs to at least 1 minimal p-cohesion in the result. Figure 17b, c reports the runtime on *DBLP* and *Amazon* with p



(a) All Datasets, $p = 0.6$



(b) DBLP

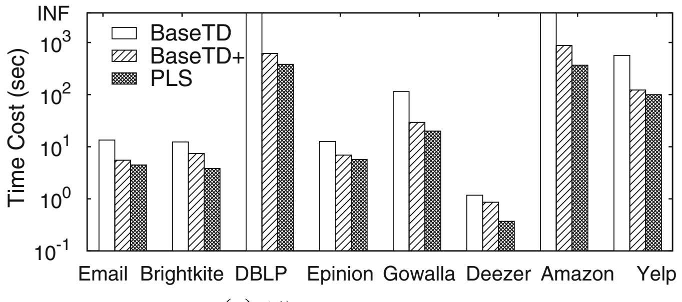


(c) Amazon

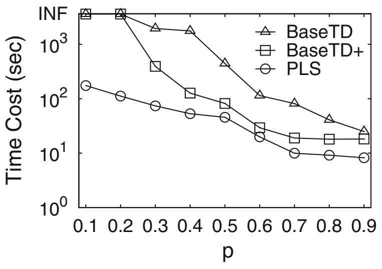
Fig. 17 Finding a Minimal p-Cohesion

varying from 0.1 to 0.9. The runtime of `GlobalS` drops slightly with the increase of p, because a larger p makes vertex deletion more efficient. `LocalS` and `LocalS`* perform faster when p is small, because the computation space is reduced for a small p. Note that `LocalS` and `LocalS`* expand the query vertex to a p-cohesion and then delete the vertices from the p-cohesion to produce a minimal p-cohesion containing q. `LocalS` runs slower than `LocalS`*, because the former does not consider the penalty effect during the expansion procedure and usually finds a larger p-cohesion than `LocalS`*. When $p = 0.9$, the runtime of `LocalS`* is not larger than that of $p = 0.8$ on *Amazon*, because the p-cohesions have a similar size for both p values, while the deletion procedure is faster for $p = 0.9$. In general, `LocalS`* significantly outperforms `GlobalS` on runtime.

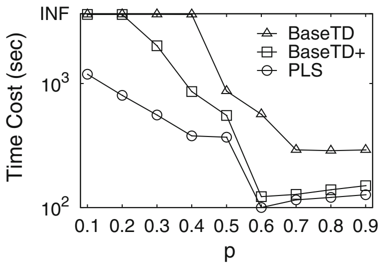
Evaluating algorithms for $DPCE$. Here, we evaluate the performance of `BaseTD`, `BaseTD+` and `PLS` to find disjoint minimal p-cohesions. Figure 18a reports the performance on all the datasets when $p = 0.6$. Figure. 18b and 18c shows that `BaseTD` and `BaseTD+` cannot finish the computation in 1 hour when p is small. Although the computation of one minimal p-cohesion is fast when p is small, the size of produced p-cohesion is quite small such that we have to compute much more minimal p-cohesions than that of large p values. This issue is relaxed when p is large enough such as $p = 0.6$. There are 4 factors which influence the trends of runtime: (a) the size of every expanded p-cohesion, (b) the size of every resulting minimal p-cohesion, (c) the number of resulting minimal p-cohesions and (d) the deletion procedure from an expanded p-cohesion to a minimal p-cohesion. When the p value increases, usually (a) increases, (b) increases, (c) decreases and (d) speeds up, which, in total, constitutes the trends of runtime for different p. In general, `PLS` is significantly faster than the other algorithms.



(a) All Datasets, $p = 0.6$



(b) Gowalla



(c) Yelp

Fig. 18 Finding Disjoint Minimal p-Cohesions

7 Conclusion

In this paper, we study two representative problems on the fortress-like p-cohesion subgraphs: minimum p-cohesion search and diversified p-cohesion enumeration. We analyze the complexity of the problems and prove that finding a minimum p-cohesion is NP-hard to approximate for any constant factor and the minimal p-cohesion enumeration is intractable. From theory to practice, we propose efficient algorithms to find a minimal p-cohesion for a query vertex and a set of disjoint minimal p-cohesions. For a feasible solution of MinSeed, we employ the discovered minimal p-cohesions to reduce the seed number required for cascading the whole network. Comprehensive experiments show that our algorithms are efficient, the minimal p-cohesions hold the fortress property, and the algorithms help solve the MinSeed problem. In the future, it is interesting to study whether there are more applications of p-cohesion, e.g., community deception, graph decomposition and network stability.

Acknowledgements Fan Zhang is supported by National Natural Science Foundation of China under Grant 62002073 and Guangzhou Basic and Applied Basic Research Foundation under Grant 202102020675. Ying Zhang is supported by ARC DP180103096 and FT170100128. Lu Qin is supported by ARC DP160101513. Wenjie Zhang is supported by ARC DP180103096. Xuemin Lin is supported by NSFC61232006, ARC DP180103096, DP170101628 and the National Key R&D Program of China under grant 2018YFB1003504.

References

1. Alimonti P, Kann V (1997) Hardness of approximating problems on cubic graphs. In CIAC, pages 288–298

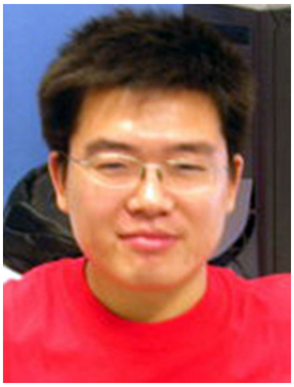
2. O. Amini, D. Peleg, S. Pérennes, I. Sau, S. Saurabh. On the approximability of some degree-constrained subgraph problems. Discrete Applied Mathematics, 160(12), 1661–1679, 2012
3. Arora A, Galhotra S, Ranu S (2017) Debunking the myths of influence maximization: an in-depth benchmarking study. In SIGMOD, pages 651–666
4. Bakshy E, Karrer B, Adamic LA (2009) Social influence and the diffusion of user-created content. In ACM Conference on Electronic Commerce, pages 325–334
5. N. Barbieri, F. Bonchi, E. Galimberti, F. Gullo. Efficient and effective community search. Data Min. Knowl. Discov., 29(5):1406–1433, 2015
6. Batagelj V, Zaversnik M (2003) An o(m) algorithm for cores decomposition of networks. CoRR, cs.DS/0310049
7. C. Bron, J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). Commun. ACM, 16(9), 575–576, 1973
8. Chen W, Wang Y, Yang S (2009) Efficient influence maximization in social networks. In IV, Fogelman-Soulié F, Flach PA, and Zaki MJ editors, SIGKDD, pages 199–208. ACM
9. Cohen J (2008) Trusses: Cohesive subgraphs for social network analysis. National Security Agency Technical Report, 16
10. Cui W, Xiao Y, Wang H, Wang W. (2014) Local search of communities in large graphs. In SIGMOD, pages 991–1002
11. Danisch M, Balalau OD, Sozio M (2008) Listing k-cliques in sparse real-world graphs. In WWW, pages 589–598
12. Easley DA, Kleinberg JM (2010) Networks, Crowds, and Markets - Reasoning About a Highly Connected World. Cambridge University Press
13. Epasto A, Lattanzi S, Sozio M (2010) Efficient densest subgraph computation in evolving graphs. In WWW, pages 300–310
14. Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, X. Lin. Efficient algorithms for densest subgraph discovery. PVLDB, 12(11), 1719–1732, 2019
15. H. Fernau, J. A. Rodríguez-Velázquez. A survey on alliances and related parameters in graphs. EJGTA, 2(1), 70–86, 2014
16. Fish B, Bashardoust A, Boyd D, Friedler SA, Scheidegger C, Venkatasubramanian S (2019) Gaps in information access in social networks? In WWW, pages 480–490
17. Fricke G, Hedetniemi ST, Jacobs DP (1998) Independence and irredundance in k-regular graphs. Ars Comb., 49
18. Kempe D, Kleinberg JM, (2003) É. Tardos. Maximizing the spread of influence through a social network. In SIGKDD, pages 137–146
19. D. Kempe, J. M. Kleinberg, É. Tardos. Maximizing the spread of influence through a social network. Theory of Computing, 11:105–147, 2015
20. J. M. Kumpula, M. Kivelä, K. Kaski, J. Saramäki. Sequential algorithm for fast clique percolation. Physical Review E, 78(2):026109, 2008
21. Laishram R, Sariyüce AE, Eliassi-Rad T, Pinar A, Soundarajan S (2018) Measuring and improving the core resilience of networks. In WWW, pages 609–618
22. Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data
23. Ley M (2002) The dblp computer science bibliography: evolution, research issues, perspectives. String processing and information retrieval. http://dblp.uni-trier.de
24. C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, X. Lin. Efficient progressive minimum k-core search. PVLDB, 13(3), 362–375, 2019
25. Liu B, Yuan L, Lin X, Qin L, Zhang W, Zhou J (2019) Efficient (α, β)-core computation: an index-based approach. In WWW, pages 1130–1141
26. R. D. Luce, A. D. Perry. A method of matrix analysis of group structure. Psychometrika, 14(2), 95–116, 1949
27. Maehara T, Suzuki H, Ishihata M (2017) Exact computation of influence spread by binary decision diagrams. In WWW, pages 947–956
28. Mihara S, Tsugawa S, Ohsaki H (2015) Influence maximization problem for unknown social networks. In Pei J, Silvestri F, and Tang J, editors, ASONAM, pages 1539–1546. ACM, 2015
29. S. Morris. Contagion. The Review of Economic Studies, 67(1), 57–78, 2000
30. M. E. Newman. Modularity and community structure in networks. Proceedings of the national academy of sciences, 103(23):8577–8582, 2006
31. Sariyüce AE, Seshadhri C, Pinar A, Ü. V. Çatalyürek (2015) Finding the hierarchy of dense subgraphs using nucleus decompositions. In WWW, pages 927–937

32. H. Seba, S. Lagraa, H. Kheddouci. Alliance-based clustering scheme for group key management in mobile ad hoc networks. The Journal of Supercomputing, 61(3), 481–501, 2012
33. S. B. Seidman. Network structure and minimum degree. Social networks, 5(3):269–287, 1983
34. S. B. Seidman, B. L. Foster. A graph-theoretic generalization of the clique concept*. Journal of Mathematical sociology, 6(1):139–154, 1978
35. Tang J, Tang X, Xiao X, Yuan J (2018) Online processing algorithms for influence maximization. In SIGMOD, pages 991–1005
36. E. Tomita, A. Tanaka, H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. Theor. Comput. Sci., 363(1):28–42, 2006
37. J. Ugander, L. Backstrom, C. Marlow, J. Kleinberg. Structural diversity in social contagion. PNAS, 109(16), 5962–5966, 2012
38. J. Wang, J. Cheng. Truss decomposition in massive networks. PVLDB, 5(9), 812–823, 2012
39. C. I. Wood, I. V. Hicks. The minimal k-core problem for modeling k-assemblies. The Journal of Mathematical Neuroscience, 5(1):14, 2015
40. Yelp (2015) Yelp Dataset Challenge: Discover what insights lie hidden in our data. https://www.yelp.com/dataset/challenge
41. Yero IG, Rodríguez-Velázquez JA (2013) Defensive alliances in graphs: a survey. http://arxiv.org/abs/1308.2096
42. Zarezade A, Khodadadi A, Farajtabar M, Rabiee HR, Zha H (2017) Correlated cascades: Compete or cooperate. In AAAI, pages 238–244
43. Zhang F, Yuan L, Zhang Y, Qin L, Lin X, Zhou A (2018) Discovering strong communities with user engagement and tie strength. In DASFAA, pages 425–441
44. Zhang F, Zhang Y, Qin L, Zhang W, Lin X (2018) Efficiently reinforcing social networks over user engagement and tie strength. In ICDE, pages 557–568
45. Zhang P, Chen W, Sun X, Wang Y, Zhang J (2014) Minimizing seed set selection with probabilistic coverage guarantee in a social network. In SIGKDD, pages 1306–1315

Conggai Li is currently a research associate in the Centre for Audio, Acoustics and Vibration (CAAV) within the Faculty of Engineering and Information Technology (FEIT), the University of Technology Sydney (UTS), Australia. In March 2021, Dr. Li received her Doctoral degree from the Australian Artificial Intelligence Institute (AAII), UTS. She received her MS degree and BS degree both from Taiyuan University of Technology, China. Her research interests include graph databases and efficient query algorithms. She has published papers in top venues including VLDB and TKDE.

Fan Zhang is currently an associate professor in Guangzhou University. He was a research associate in the School of Computer Science and Engineering, University of New South Wales. He received the BEng degree from Zhejiang University in 2014 and the PhD from University of Technology Sydney in 2017. His research interests include graph algorithms and social networks. Since 2017, he has published more than 12 papers in top venues including SIGMOD, PVLDB, ICDE, IJCAI, AAAI, TKDE and VLDB Journal.

Ying Zhang is a Professor and ARC Future Fellow (2017–2021) at Australian Artificial Intelligence Institute (AAII), the University of Technology Sydney (UTS). He received his BSc and MSc degrees in Computer Science from Peking University and PhD in Computer Science from the University of New South Wales. His research interests include query processing on data stream, uncertain data and graphs. He was an ARC APD (2011-2013) and ARC DECRA (2014-2016) holder. He is currently an associated Editor of TKDE (2017-now).



Lu Qin received the BEng degree from the Department of Computer Science and Technology, Renmin University of China, in 2006, and the PhD degree from the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, in 2010. He is now an Associate Professor in the Australian Artificial Intelligence Institute (AAII), University of Technology Sydney. His research interests include big graph processing and I/O efficient algorithms.

Wenjie Zhang received the PhD degree in computer science and engineering from the University of New South Wales, in 2010. She is currently a full professor and ARC DECRA fellow in the School of Computer Science and Engineering, the University of New South Wales, Australia. Her research interests include spatial–temporal data analysis, uncertain data analysis and graph data processing. She has published more than 90 papers in top venues including SIGMOD, VLDB, ICDE, WWW, TODS, TKDE and VLDBJ.

Xuemin Lin received the BSc degree in applied math from Fudan University, in 1984, and the PhD degree in computer science from the University of Queensland, in 1992. He is a UNSW scientia professor with the School of Computer Science and Engineering, University of New South Wales. He has been the head of the Database Research Group, University of New South Wales, since 2002. He is currently the editor-in-chief of the IEEE Transactions on Knowledge and Data Engineering. He is an IEEE Fellow. His current research interests lie in graph mining, spatial data analysis, text similarity and uncertain data mining.

Authors and Affiliations

Conggai Li[1] · **Fan Zhang**[2] · **Ying Zhang**[3] · **Lu Qin**[3] · **Wenjie Zhang**[4] · **Xuemin Lin**[4]

✉ Fan Zhang
zhangf@gzhu.edu.cn

Conggai Li
conggai.li.cs@gmail.com

Ying Zhang
ying.zhang@uts.edu.au

Lu Qin
lu.qin@uts.edu.au

Wenjie Zhang
zhangw@cse.unsw.edu.au

Xuemin Lin
lxue@cse.unsw.edu.au

1 University of Technology Sydney, Sydney, Australia

2 Guangzhou University, Guangzhou, China

3 AAII, University of Technology Sydney, Sydney, Australia

4 University of New South Wales, Sydney, Australia