Intelligent and Converged Networks ISSN 2708-6240
2021, 2(4): 306–319 DOI: 10.23919/ICN.2021.0022

Public-private-core maintenance in public-private-graphs

Dongxiao Yu, Xilian Zhang, Qi Luo*, Lifang Zhang, Zhenzhen Xie, and Zhipeng Cai

Abstract: A public-private-graph (pp-graph) is developed to model social networks with hidden relationships, and it consists of one public graph in which edges are visible to all users, and multiple private graphs in which edges are only visible to its endpoint users. In contrast with conventional graphs where the edges can be visible to all users, it lacks accurate indexes to evaluate the importance of a vertex in a pp-graph. In this paper, we first propose a novel concept, public-private-core (pp-core) number based on the k-core number, which integrally considers both the public graph and private graphs of vertices, to measure how critical a user is. We then give an efficient algorithm for the pp-core number computation, which takes only linear time and space. Considering that the graphs can be always evolving over time, we also present effective algorithms for pp-core maintenance after the graph changes, avoiding redundant re-computation of pp-core number. Extension experiments conducted on real-world social networks show that our algorithms achieve good efficiency and stability. Compared to recalculating the pp-core numbers of all vertices, our maintenance algorithms can reduce the computation time by about 6–8 orders of magnitude.

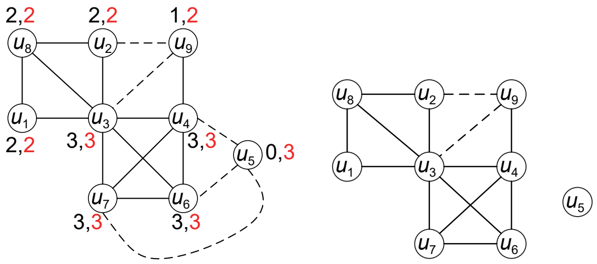
Key words: core decomposition maintenance; public-private-graph (pp-graph); critical user; social network

1 Introduction

In social networks, due to privacy concerns, users tend to hide their social connections, making the relations between two users not visible to other users in public but only to themselves. For example, Dey et al.[1] crawled a snapshot of 1.4 million New York City Facebook users and reported that 52.6% of them hid their friends list. A model of public-private-graphs (pp-graph) is developed to represent this kind of social network[2]. For a pp-graph $\mathcal{G}$, it contains a public graph G, which is visible to all users, and each user u has a private graph G_u, which is only visible to itself. Therefore, the pp-graph $\mathcal{G}$ can be regarded as the union of the public graph and the private graphs of all vertices. Recently, many graph analytic tasks have been investigated on pp-graphs, such as all-pairs shortest path distances[3], pairwise node similarities[4], and correlation clustering[5]. However, there are few researches on user engagement on public-private graphs.

User engagement on social networks has attracted significant interest over recent years[6]. It can be used as a measure of how critical a user is, such as k-core[6–8] and k-truss[9]. k-core is a simple and popular model based on degree constraint that the degree of each vertex in a k-core is no less than k, and the core number of a vertex can be used to measure its importance/influence. But the definition of core number cannot be used in pp graphs directly, since there are some edges private so that a vertex cannot know the number of neighbors the other vertices have. For example, though vertex u_5 in Fig. 1a does not have any neighbor in the public graph, its core number in the whole pp-graph is 3, indicating that the vertex is relatively important. Therefore, simply considering the public graph cannot do a good job of capturing the importance of vertices. To solve this problem, adapting the concept of core number in general graphs[10], we propose the concept

• Dongxiao Yu, Xilian Zhang, Qi Luo, Lifang Zhang, and Zhenzhen Xie are with the School of Computer Science and Technology, Shandong University, Qingdao 266200, China. E-mail: dxyu@sdu.edu.cn; xilianzhang@mail.sdu.edu.cn; luoqi2018@mail.sdu.edu.cn; zhanglf@mail.sdu.edu.cn; xiezz 21@sdu.edu.cn.
• Zhipeng Cai is with the Department of Computer Science, Georgia State University, Atlanta, GA 30080, USA. E-mail: zcai@gsu.edu.
* To whom correspondence should be addressed.
Manuscript received: 2021-12-03; accepted: 2022-01-18



(a) pp-graph $\mathcal{G}$ (b) Personalized pp-graph of vertex u_9

Fig. 1 Examples of pp-graph and personalized pp-graph. Public edges are represented by solid lines and privated edges are represented by dashed lines. The first number denotes the public-core of the corresponding vertex, while the second number denotes the pp-core number.

of public-private-core (pp-core) number in the pp graph, which considers both the public graph and the private graph for each vertex.

Our contributions are summarized as follows.

● We first propose the concept of pp-core number of a vertex u, which integrally considers both the public graph and the private graphs of vertices. We also give an algorithm that can compute the pp-core number of vertices in linear time and linear space.

● We further propose efficient algorithms for the core maintenance problem, i.e., updating the pp-core number of vertices after the graph is changed. We focus on the scenario of edge change, since the vertex change can also be seen as edge change[11]. Specifically, the core maintenance algorithm will be divided into four cases, i.e., public edge insertion/deletion and private edge insertion/deletion. We first show the pp-core number of each vertex in the pp-graph changes by at most 1 after the insertion or deletion of an edge (public or private), and then give sufficient conditions for identifying the vertices whose pp-core number will change for each case. In addition, we propose an optimization algorithm by giving a definition named Super Support vertex Number (SSN) in the case of a public edge insertion, where SSN indicates a more accurate condition for the change of pp-core number of a vertex.

● Finally, we perform extensive experiments to evaluate our algorithms over four real-world datasets, and the results demonstrate the good efficiency and scalability of the algorithms. The proposed dynamic maintenance algorithm can reduce the computation time by about 6–8 orders of magnitude compared to recomputing the pp-core number of all vertices. In addition, the algorithm based on traversal algorithm and SSN is five or six times better than the one based on traversal only.

The rest of this paper is organized as follows. We first present some basic concepts in Section 2, and we give the method and algorithm of pp-core computation in Section 3. Then, the theoretical findings that facilitate incremental pp-core maintenance are proposed in Section 4. In Section 5, the experimental results are reported, and Section 6 reviews the related work. At last, the paper is concluded in Section 7.

2 Problem definition

We consider a simple undirected and unweighted graph $G=(V,E)$, where $V(G)$ and $E(G)$ are the vertex set and the edge set, respectively. Let $n=|V(G)|$ and $m=|E(G)|$. We say a graph $H=(V(H),E(H))$ is a subgraph of G, denoted as $H\subseteq G$, if $V(H)\subseteq V(G)$ and $E(H)\subseteq E(G)$. Given a vertex set $V'\subseteq V$, the subgraph of G induced by V', is defined as $G(V')=(V',E')$ where $E'=\{(u,v)\in E \mid u,v\in V'\}$. We define $N_H(u)=\{u\in V \mid (u,v)\in E(H)\}$ and $d_H(u)=|N_H(u)|$ as the set of neighbors and the degree of a vertex u in H. The maximum and minimum degrees of vertices in H are denoted as $\Delta(H)$ and $\delta(H)$, respectively. We next give some useful formal definitions.

Definition 1 (*k*-core) Given a graph $G=(V,E)$, the k-core of G is a maximal connected subgraph H of G, such that each vertex in H has at least k neighbors, i.e., $\delta(H)\geqslant k$. The core number of a vertex u, denoted by $c(u)$, is defined as the largest k, such that u is contained in a k-core of G.

For an edge (u,v) in graph G, it is called a public edge if (u,v) is visible to each vertex $w\in G$, while it is called a private edge if (u,v) is only visible to vertex u and v. A graph is a public one where each edge is a public edge, and a private one of vertex u if each edge in the graph is only visible to u.

Definition 2 (pp-graph[2]) Given a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, it is called a pp-graph if it contains both public and private edges.

Given a pp-graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, it constains a public graph $G=(V,E)$ as a subgraph, where $V=\mathcal{V}$ and E

consists of all public edges in $\mathcal{E}$. Besides, for each vertex $u \in \mathcal{V}$, u has an associated private graph $G_u = (V_u, E_u)$, where $V_u \subseteq \mathcal{V}$ and $E_u \subseteq \mathcal{E} \setminus E$. The public graph G is visible to everyone, but the private graph G_u is only visible to vertex u. Hence, in the view of the vertex u, the entire graph it can see and access is the one composed by the public graph G and its own private graph G_u. For the sake of simplicity, we will use a pp-graph to represent a pp-graph directly in the rest of this paper.

Definition 3 (**personalized pp-graph**) Given a pp-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a vertex $u \in \mathcal{V}$, let $G = (V, E)$ be the public graph and $G_u = (V_u, E_u)$ the private graph of vertex u. The personalized pp-graph of u is denoted by $\mathcal{G}_u = (\mathcal{V}_u, \mathcal{E}_u)$, where $\mathcal{V}_u = \mathcal{V}$ and $\mathcal{E}_u = E \cup E_u$.

In a pp-graph $\mathcal{G}$, each vertex has a core number $c(u)$, which is public, to represent its cohesiveness in the public graph. However, it does not represent the cohesiveness of vertices in the pp-graph well, where each vertex not only has public neighbors, but also has private neighbors. So we propose a new concept of pp-core number to solve this problem.

Definition 4 (**pp-core number**) Given a pp-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the pp-core number of u in $\mathcal{G}_u$, denoted by $pc(u)$, is computed by the following formula.

$$\mathop{arg\,max}_{pc \geqslant 0} \{ |\{ v \in N(u) | c(v) \geqslant pc \}| \geqslant pc \} \tag{1}$$

Example 1 Consider the pp-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in Fig. 1a, the public edges and private edges are represented by solid lines and dashed lines, respectively. The public graph $G = (V, E)$ is the subgraph of $\mathcal{G}$ that is represented by all vertices and the solid lines. Given a private edge (u_2, u_9), it is only visible to vertices u_2 and u_9. The private graph $G_{u_9} = (V_{u_9}, E_{u_9})$ of u_9 is the subgraph of $\mathcal{G}$ induced by V_{u_9}, where $V_{u_9} = \{u_9, u_2, u_3\}$ and the personalized pp-graph of u_9 is $\mathcal{G}_{u_9} = (\mathcal{V}, E \cup E_{u_9})$, which is shown in Fig. 1b. The core number and the pp-core number of each vertex are illustrated using numbers of black and red, respectively. From Fig. 1b, it can be seen that the core number and the pp-core number can be significantly different. For example, the core number of u_5 is 0, while its pp-core number is 3. Clearly, the pp-core number can better illustrate the real cohesiveness of vertices when private edges exist.

In subsequent sections, we will present efficient algorithms for computing the pp-core number of vertices and maintaining the pp-core number when the graph changes (with public/private edge insertion/deletion).

3 pp-core number computation

We present an efficient algorithm in this section for computing the pp-core number of vertices in a pp-graph. Some useful theretical results that support the correctness of our algorithm will be first given.

3.1 Theoretical basis

Different from core decomposition in a general graph [12], where the core number can be computed globally, the pp-core number of a vertex in a pp-graph has to be computed locally, based on the core numbers of its neighbors, as shown in Definition 4, due to the existence of the private edges. Hence, given a public-private graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we first conduct core decomposition in the public graph of $\mathcal{G}$ to calculate the core numbers of all vertices, and then compute the pp-core of each vertex.

Based on Formula 1, the pp-core number of a vertex u is the largest integer pc, such that there are at least pc neighbors with core number at least pc. We here let $ulist$ be a list containing the core numbers of u's neighbors in a descending order. Then we can get that

$$k(u) = \max_{1 \leqslant i \leqslant len(ulist)} (min(i, ulist[i-1])) \tag{2}$$

According to the above equation, $k(u)$ is the largest i in $1 \leqslant i \leqslant len(list)$ to guarantee that u has at least i neighbors with core number not less than the $(i-1)$-th value in $ulist$. Because the values in $ulist$ are stored in a decreasing order, for each u, it has at least i neighbors whose core numbers are at least $k(u)$. By the definition of pp-core number of a vertex u, $pc(u)$ is not less than $k(u)$. However, if $k(u) > pc(u)$, then u has at least $pc(u)+1$ neighbors whose core numbers are not less than $pc(u)+1$ according to Eq. 2, which will conflict with Definition 4. Based on these analysis, we give the following lemma.

Lemma 1 Given a pp-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the pp-core number $pc(u)$ of vertex $u \in \mathcal{V}$ satisfies that $pc(u) = k(u)$.

In the following sections, we will use the above formula to calculate the pp-core number of a vertex u and directly use $pc(u)$ to represent $k(u)$. Note that $pc(u)$ is the maximum value of the smaller of $ulist(i-1)$ and i, then we can regard $ulist(i-1)$ and i as two functions for ease of understanding, where $ulist(i-1)$ is a monotonous non-increasing and i is monotonically increasing. From a geometric perspective, the maximum of their minimums occurs at the intersection of the curve $ulist(i-1)$ and i, which is shown in Fig. 2.

Lemma 2 Given a pp-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the pp-core number $pc(u)$ satisfies that $pc(u) \geqslant c(u)$ for each vertex u in $\mathcal{G}$.

Proof. We prove the lemma by contradiction. Suppose there exists a vertex $u \in \mathcal{V}$ and $pc(u) < c(u)$. This means u has at most $c(u)-1$ neighbors whose core numbers are not less than $c(u)-1$, according to the definition of pp-core number. Otherwise, $pc(u)$ is not the maximum value that satisfies Formula 1. However, the core number of u is $c(u)$. This means u has at least $c(u)$ neighbors whose core numbers are not less than $c(u)$ according to Definition 1. Therefore, our hypothesis is impossible, i.e., $pc(u) \geqslant c(u)$ for each vertex u in $\mathcal{G}$. ■

Example 2 In Fig. 1a, u_9 has three vertices, i.e., u_2, u_3, and u_4, and their core numbers are 2, 3, and 3, respectively. Let $ulist$ be the list containing the core numbers of neighbors of vertex u_9 in a descending order, then $ulist = \{3,3,2\}$. Because the maximum value of i that satisfied $ulist[i-1] \geqslant i$ is equals to 2, we can know that u_9 has at least 2 neighbors whose core number is not less than 2, i.e., $pc(u_9) = 2$, which is the same as the result calculated by Formula 1.

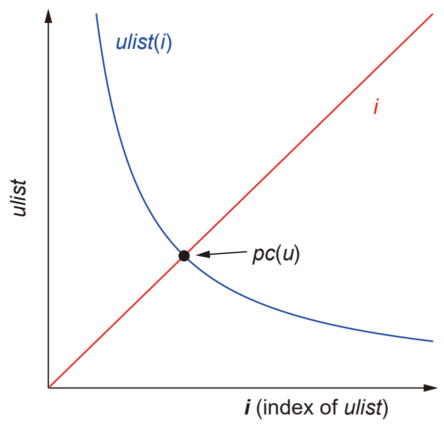


Fig. 2 $pc(u)$ **is the** y**-value at intersection of two functions.**

3.2 Algorithm

In this section, we will introduce the algorithm to compute the pp-core number of vertices in a pp-graph $\mathcal{G}$, as shown in Algorithm 1.

In Algorithm 1, according to the definition of pp-core number, we need first compute the core number of each vertex $u \in \mathcal{V}$, then compute the pp-core number of each vertex u in its personalized pp-graph $\mathcal{G}_u$ (Lines 1–3). In Procedure *ComputePC*, the pp-core number of vertex u is computed based on Eq. 2, and $ulist$ stores the core numbers of all neighbors of u in a decreasing order (Lines 4–8). The algorithm next computes the largest i in $1 \leqslant i \leqslant len(ulist)$ to guarantee that u has at least i neighbors with core number at least $ulist[i-1]$ (Lines 9–13).

Algorithm complexity. For each vertex in graph G, we compute their pp-core numbers by sorting their neighbors based on the core number, thus the worst case happens when the vertex's degree is equal to $\Delta(\mathcal{G})$, which is the max degree of the pp-graph. The time complexity of the sorting process is $O(\varDelta \log \varDelta)$. For the whole graph with $|V|$ vertices, the complexity is $O(|V| \varDelta \log \varDelta)$, which is simplified to $O(|E| \log \varDelta)$. In addition, we should first compute the core number of vertices before we compute their pp-core number. According to the linear time complexity of the traditional core decomposition algorithm[13], the whole time complexity is $O(|E| + |E| \log \varDelta)$.

Algorithm 1 pp-core number computation

```
   Input: G = (V, E)
1  Compute c(u) for all vertices u ∈ V ;
2  for each u ∈ V do
3      pc(u)= ComputePC(G_u, u, c(V));
4  Procedure ComputePC(G_u, u, c()):
5      ulist ← φ ;
6      for each (u, v) ∈ E do
7          ulist = ulist ∪ {c(v)};
8      Sort the values in ulist in decreasing order;
9      i ← 1 ;
10     while i ⩽ len(ulist) do
11         if ulist[i] ⩽ i then
12             break;
13     return i ;
```

4 pp-core number maintenance

In this section, we first give some lemmas to explain how to maintain pp-core numbers in Section 4.1, and we propose pp-core maintenance algorithms in the scenario of single edge insertion/deletion. When multiple edges are inserted/deleted, it can be handled by executing our maintenance algorithms for multiple times. The maintenance algorithms consider four cases: (i) private edge insertion; (ii) private edge deletion; (iii) public edge insertion; and (iv) public edge deletion.

4.1 Theoretical basis

In this section, we will give theoretical basis of our core maintenance algorithms when an edge is inserted or deleted. Previous work has proved that the core number of each vertex in a simple graph changes by at most 1 when an edge is inserted/deleted[14], and this also holds true for pp-core number in a pp-graph.

Lemma 3 If an edge $e=(u,v)$ is inserted to or removed from a pp-graph $\mathcal{G}$, then the pp-core number of each vertex in $\mathcal{G}'$ can change by at most 1.

Proof. We first analyze the case of insertion. According to the type of inserted edge, it can be divided into two cases, i.e., a public edge and a private edge.

If e is a private edge, the core numbers of all vertices will not increase because the public graph of $\mathcal{G}$ has not changed. The only change in the graph is that u and v add a new neighbor, so the pp-core number of u or v may increase by at most 1 and the pp-core numbers of others cannot change, which can be easily obtained by Formula 1.

On the other hand, if e is a public edge, assume there is a vertex w whose pp-core number changes from p to $p+x$, where $x>1$. That is to say, after inserting an edge, w has at least $p+x$ neighbors whose core values are not less than $p+x$. Because the core number of each vertex can change by at most 1 after inserting one edge, so there must be at least $p+x-1$ neighbors of w whose core numbers are not less than $p+x-1$ after deleting the edge (u,v). According to Definition 4, $pc(w)$ is at least $p+x-1$ in $\mathcal{G}$, which contradicts our assumption.

For the deletion case, assume the pp-core number $pc(w)$ of vertex w is decreased by x after deleting an edge (u,v), where $x>1$. Adding (u,v) back to the graph will increase $pc(w)$ by x, which contradicts with the result proved above. ■

Lemma 4 Given a pp-graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, if we insert/delete a private edge (u,v) into/from $\mathcal{G}$ and $\mathcal{G}$ becomes $\mathcal{G}'$, then the pp-core number $pc(u)$ of u cannot change if $c(v)\leqslant pc(u)$. Similarly, when $c(u)\leqslant pc(v)$, then the pp-core number $pc(v)$ of v cannot change.

Proof. For the case of insertion, we take the vertex u as an example for analysis. Assume that $c(v)\leqslant pc(u)=p$ and $pc(u)$ becomes $p+1$ after a private edge (u,v) is inserted into $\mathcal{G}$. Note that the core numbers of all vertices in the public graph of $\mathcal{G}$ are not changed, so the increase of $pc(u)$ can only be due to the support of the new neighbor v. The pp-core number of u changes from p to $p+1$ after insertion, which means u has at most p neighbors in $\mathcal{G}$ whose core number is at least $p+1$, but it has at least $p+1$ neighbors in $\mathcal{G}'$ whose core number is at least $p+1$. In other words, the only new neighbor v of u must satisfy that $c(v)\geqslant p+1$, which contradicts with our assumption.

For the deletion case, assume that the pp-core number $pc(u)$ of vertex u decreases when $c(v)\leqslant pc(u)$. Adding (u,v) back to the graph will increase $pc(u)$, which contradicts with the result above. ■

Lemma 5 Given a pp-graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ and a vertex $w\in\mathcal{V}$, the pp-core number $pc(w)$ of w changes only when there exists a neighbor of w whose core number changes or the number of w's neighbors changes.

Proof. According to Definition 4, because the pp-core number $pc(w)$ of w is determined by the core numbers of its neighbors, so $pc(w)$ cannot change if the core numbers of its all neighbors do not change. ■

Next, we will give a lemma to find the vertices whose core number might change after inserting or deleting an edge.

Lemma 6 Given a pp-graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, if a public edge (u,v) is inserted/deleted, where $k=c(u)\leqslant c(v)$, and $\mathcal{G}$ becomes $\mathcal{G}'$, then only the vertices $w\in\mathcal{V}\setminus\{u\}$ satisfying $pc(w)=k$ may have their pp-core numbers changed.

Proof. We first analyze the case of insertion, it can be known that only the vertices w that have $c(w)=k$

may have their core numbers changed. As for the pp-core numbers of the vertices in $\mathcal{G}$, we consider two cases: $pc(w) < k$ and $pc(w) > k$.

(1) When $pc(w) < k$, assume that $pc(w)$ changes from p to $p+1$, where $p+1 \leqslant k$ (Lemma 3 claims the pp-core number of each vertex changes by at most 1). Note that w cannot be v, since the pp-core number of v is not less than k according to Lemma 2, which means w has no new neighbors after insertion. If $pc(w)$ can become from p to $p+1$, then it means w has at most p neighbors whose core numbers are not less than $p+1$ in $\mathcal{G}$, and has at least $p+1$ neighbors whose core numbers are not less than $p+1$ in $\mathcal{G}'$. In other words, there exists the core number of a neighbor of w that changes from p to $p+1$, which contradicts the previous conclusion, since only the vertices w that have $c(w) = k$ may have their core numbers changes.

(2) When $pc(w) > k$, assume that $pc(w)$ changes from p to $p+1$, where $p > k$, then there must be the core number of a neighbor of w that changes from p to $p+1$ or w adds a new neighbor whose core number is not less than $p+1$ (i.e., $w = v$). However, neither case is true, because only the core number of the vertex with value k can change and $c(u) = k < p+1$.

In the case of deletion, we can use a method similar to that of insertion. Up to now, we have proved Lemma 6 is correct. ■

We next give a definition to explain which neighbors of a vertex u in a pp-graph can support the increase in the pp-core number of u.

Definition 5 (**Superr SSN**) Given a pp-graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ and $u \in \mathcal{V}$, if the neighbor w of u in $\mathcal{G}$ satisfies that $c(w) > pc(u)$, then w is called a super support vertex of u and the number of super support vertices of u is denoted by SSN(u).

According to Definition 4, only the super support vertex of u can support its pp-core number $pc(u)$ increase. Based on this observation, we will give a more specific lemma to illustrate the pp-core number maintenance after inserting an edge to the public-private graph.

Lemma 7 Given a pp-graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, if we insert a public edge (u,v) into $\mathcal{G}$ and $\mathcal{G}$ becomes $\mathcal{G}'$. Suppose that $k = c(u) \leqslant c(v)$, then only the vertex $w \in \mathcal{V} \setminus \{u\}$ whose SSN(w) value in $\mathcal{G}'$ is larger than k, may have $pc(w)$ increased.

Proof. According to Lemma 6, it can be known that only the vertices $w \in \mathcal{V}$ except u satisfying $pc(w) = k$ may have their pp-core numbers changed. For a vertex w, if its pp-core number can increase from k to $k+1$, then it has at least $k+1$ neighbors whose core numbers are not less than $k+1$ in $\mathcal{G}'$, i.e., the SSN(w) of w in $\mathcal{G}'$ is larger than k. ■

4.2 Private edge insertion/deletion

Here we give the pp-core number maintenance algorithm after inserting/deleting a private edge.

After inserting a private edge (u,v) into $\mathcal{G}$, this edge is only visible to its endpoint vertexes u and v, so the insertion of (u,v) can only impact $pc(u)$ and $pc(v)$. In addition, we know that the pp-core number $pc(u)$ of u increases only when $c(v) > pc(u)$ according to Lemma 4 and we then recalculate the $pc(u)$ in this case. A similar process is done for the vertex v. The detailed pp-core number maintenance algorithm after inserting a private edge is given in Algorithm 2. We first compare $c(v)$ and $pc(u)$ to make sure whether it is necessary to update $pc(u)$. If $c(v) > pc(u)$, the algorithm calls *ComputePC* to recompute $pc(u)$ (Lines 1 and 2). The similar operations are done for vertex v (Lines 3 and 4).

Algorithm complexity. When a private edge (u,v) is inserted or deleted, there are only two vertices whose pp-core number changes. Thus, the time complexity is $O(\varDelta \log \varDelta)$.

4.3 Public edge insertion

In this section, we discuss public edge insertion in a pp-graph $\mathcal{G}$. According to Lemma 5, the pp-core number $pc(w)$ of a vertex w changes only when there exists a neighbor of w whose core number changes.

Algorithm 2 $PrivateEdge$ $(\mathcal{G}(\mathcal{V},\mathcal{E}), c(), pc(), u, v)$

Input: pp-graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, private edge $e = (u,v)$ to be inserted/deleted, $c()$ is the set of core number, and $pc()$ is the pp-core number of each vertex

Output: The updated pp-core number for each vertex

1 **if** $c(v) > pc(u)$ **then**
2 $\quad ComputePC(\mathcal{G}_u, u)$;
3 **if** $c(u) > pc(v)$ **then**
4 $\quad ComputePC(\mathcal{G}_v, v)$;

Thus, if we have found a vertex set, denoted by C in which each vertex changes its core number, then we can reduce the range of vertices in $\mathcal{G}$ whose pp-core number changes. Let NC represent the vertex set in which each vertex has at least a neighbor in C. Then we only need to update the pp-core number of the vertices in NC.

To find the vertices whose core number changes, we adopt an algorithm called Traversal, which is proposed in Ref. [14]. In Traversal algorithm, there are two definitions, MCD and PCD. $MCD(u)$ represents the number of neighbors w of u, such that $c(w) \geqslant c(u)$; $PCD(u)$ is defined as the number of neighbor w of u, such that either $c(w)=c(u)$ or $c(w)>c(u)$ and $MCD(w)>c(u)$. The PCD value of a vertex u represents its potential number of neighbors that support the increase of $c(u)$. We give two useful lemmas as follows and more details can be found in Ref. [14].

Lemma 8 If a public edge (u,v) is inserted to or removed from a pp-graph $\mathcal{G}$, where $c(u)<c(v)$, then $c(v)$ cannot change.

Lemma 9 Given a pp-graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, if a public edge (u,v) is inserted to or removed from $\mathcal{G}$ and $c(u) \leqslant c(v)$, then only the vertices $w \in \mathcal{V}$ that have $c(w)=c(u)$ and $MCD(w)>c(u)$, and are reachable from u via a path that consists of vertices with core number equal to $c(u)$ and MCD values greater than $c(u)$, may have their core numbers incremented.

Given a pp-graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, the algorithm for updating the pp-core numbers of all vertices in $\mathcal{V}$ requires two steps. First, it computes the set C based on the Traversal algorithm, which is a set of vertices having their core numbers updated. The second step is to compute the pp-core number of all vertices that have neighbors in C according to Lemma 5.

The detailed pp-core number maintenance algorithm after inserting a public edge (u,v) is given in Algorithm 3. We first set the root vertex as the vertex with a smaller core number between u and v, then add (u,v) into $\mathcal{E}$ (Lines 1–4). Let S be the vertex set in which the core number is possible to increase and add r to S (Line 5). For each vertex in $\mathcal{V}$, we set the flags *visited* and *cd* to 0, and set *removed* to 1 (Lines 6 and 7). As mentioned

Algorithm 3 $InsertPublicEdge$ ($\mathcal{G}(\mathcal{V},\mathcal{E})$, $G(V,E)$, $c()$, u, v , $MCD(), PCD()$)

Input: $\mathcal{G}$: the pp-graph; G: public graph; $c()$: set of core number; (u,v): inserted edge; $MCD()$: set of MCD values; $PCD()$: the set of PCD values

Output: The updated pp-core number for each vertex

1 $r \leftarrow u$;
2 **if** $c(v) < c(u)$ **then**
3     $r \leftarrow v$;
4 $\mathcal{G} \leftarrow \mathcal{E} \cup \{(u,v)\}$;
5 $S \leftarrow \varnothing$; $S.push(r)$;
6 $visited\ [w] = 0$ and $cd(w) = 0, \forall w \in \mathcal{V}$;
7 $removed\ [w] = 1, \forall w \in \mathcal{V}$;
8 $k \leftarrow c(r)$;
9 $cd(r) \leftarrow PCD(r)$;
10 $visited\ [r] \leftarrow 1$;
11 **while** $S \neq \varnothing$ **do**
12     $w \leftarrow S.pop()$;
13   **if** $cd(w) > k$ **then**
14   **for each** $(w,t) \in E$ **do**
15   **if** $c(t) = k$ and $MCD(t) > k$ and $visited\ [t] = 0$ **then**
16                 $S.push(t)$;
17                 $visited\ [t] \leftarrow 1$;
18                 $cd\ [t] \leftarrow cd\ [t] + PCD\ [t]$;
19   **else**
20   **if** $removed\ [w] = 0$ **then**
21             $DfsRemove(G, c, cd, removed, k, w)$;
22   **for each** $visited\ [w]$=1 **do**
23   **if** $removed\ [w] = 0$ **then**
24             $c(w) \leftarrow c(w) + 1$;
25             $C \leftarrow C \cup \{w\}$;
26   **for each** $x \in C$ **do**
27   **for each** $(x,y) \in \mathcal{E}$ and $y \notin NC$ **do**
28             $NC \leftarrow NC \cup \{y\}$;
29             $ComputePC(\mathcal{G}, y)$;

in Lemma 6, only the vertices $w \in \mathcal{V} \setminus \{u\}$ satisfying $pc(w)=k$ may have their pp-core numbers changed, where $k=c(r)$ (Line 8). We use the $cd(r)$ to record the value $PCD(r)$ of r and set $visited\,[r]=1$ (Lines 9 and 10). For a vertex $w \in S$, we check if it is possible to increase its core numbers by compare $cd(w)$ and k (Lines 11–13). If yes, then we check its neighbors t according to Lemma 9 and add the vertices whose core numbers may increase into S. (Lines 11–18). Otherwise, we call Algorithm 4 to remove w and update the information of its neighbors (Lines 19–21). After processing all vertices whose core number may increase, then we add the core number of a vertex that is visited but not removed by 1 and we add it to C

Algorithm 4 $DfsRemove$ **(**$G(V,E)$**,** $c()$**,** $cd()$**,** $removed$ **[ ],** k**,** w**)**

$removed\ [w] \leftarrow 1$;
for each $(w,t) \in E$ **do**
  if $c(t) = k$ **then**
    $cd(t) \leftarrow cd(t) - 1$;
    if $cd(t) = k$ and $removed\ [t]$ =*0* **then**
      $DfsRemove(G, ccd, removed, k, w)$;

(Lines 22–25).

In the second step, we put all neighbors of C into the set NC and recompute their pp-core numbers by invoking *ComputePC* (Lines 26–30).

By using the SSN value and Lemma 7, we propose an optimized algorithm named $InsertWithSSN$ in Algorithm 5. For each vertex $w \in C$, we will compare the pp-core number $pc(v)$ of the neighbor v of w with k (Lines 1 and 2). If $pc(v) = k$, then SSN(v) will increase by 1, since v has the neighbor w to support its pp-core number increase (Lines 3 and 4). When SSN value is increased to be more than k, the vertex v can have its pp-core number increased by 1 (Lines 5 and 6).

Algorithm complexity. Traversal algorithm in Algorithm 3 basically does a depth-first traversal on vertices whose cd values are greater than the k value of root vertex. In the worst case, the whole graph will be traversed, i.e., each edge in the graph will be visited at least once. Thus, the time complexity for Traversal algorithm is $O(|E|)$ at the worst. After we find vertices set C whose core number was changed by Traversal algorithm, we need to find the neighbor set NC of vertices in C and recompute their pp-core number, which will take $O(|C|)$ in the worst case. In total, the time complexity for public edge insertion algorithm is

Algorithm 5 $InsertWithSSN$ **(**$\mathcal{G}(\mathcal{V},\mathcal{E})$**,** C**,** SSN()**,** K**)**

Input: $\mathcal{G}$: pp-graph; G: public graph; C: set of vertices having their core numbers changed; SSN : set of SSN value; k: core number of root vertex
Output: The updated pp-core number for each vertex

1 **for each** $w \in C$ **do**
2   **for each** $v \in n(w)$ **do**
3   **if** $pc(v) = k$ **then**
4   SSN$(v)++$;
5   **if** SSN $(v) > k$ **then**
6         $pc(v) = pc(v) + 1$;

$O(|C| + |E| + |NC|\varDelta \log \varDelta)$. Algorithm 5 introduces a method to facilitate incremental pp-core maintenance by filtrating vertices from NC which can surely increase their pp-core number.

4.4 Public edge deletion

The detailed algorithm to maintain the pp-core number of each vertex when a public edge is deleted is given in Algorithm 6. We first set the root as the vertex between u and v as the one with the smaller core number, and remove (u,v) from $\mathcal{G}$ (Lines 1–5). By comparing $c(u)$ and $c(v)$, the algorithm will be analyzed in two cases. If $c(u) = c(v)$, then we traverse u and v respectively by invoking Procedure $SearchVertex$ (Lines 6–8). $SearchVertex$ helps us find the vertex set V' in which each vertex has its core number changed and computes

Algorithm 6 $PublicEdgeDelete$ **(**$\mathcal{G}(\mathcal{V},\mathcal{E})$**,** $c()$**,** $pp\text{-}core()$**,** u**,** v**)**

Input: $\mathcal{G} = (\mathcal{V},\mathcal{E})$: pp-graph; (u,v): deleted edge; $c()$: core number of each vertex in public graph; $pc()$: pp-core number of each vertex
Output: The updated pp-core number for each vertex

1 $r \leftarrow u$;
2 $S = \varnothing$;
3 **if** $c(v) < c(u)$ **then**
4   $r \leftarrow v$;
5 $\mathcal{G} \leftarrow \mathcal{G} \setminus (u,v)$;
6 **if** $c(u) = c(v)$ **then**
7   $V_1, cd_1 \leftarrow SeachVertex(\mathcal{G}, c, u)$;
8   $V_2, cd_2 \leftarrow SeachVertex(\mathcal{G}, c, v)$;
9   $V' \leftarrow V_1 \cup V_2$;
10   $cd \leftarrow cd_1 \cup cd_2$;
11 **else**
12   $V', cd \leftarrow SeachVertex(\mathcal{G}, c, r)$;
13 $k \leftarrow c(r)$;
14 Sort cd values of the vertices in V' in increasing order;
15 **for each** $w \in V'$ in order **do**
16   **if** $cd\ [w] < k$ **then**
17     $c(w) \leftarrow k - 1$;
18   **for each** $(w,t) \in \mathcal{V}$ **do**
19   **if** $t \notin S$ **then**
20         $S \leftarrow S \cup \{t\}$;
21   **if** $cd\ [t] > cd\ [w]$ **then**
22         $cd\ [t] \leftarrow cd\ [t] - 1$;
23   Reorder cd values accordingly;
24   **else**
25   **break;**
26 **for each** $w \in S$ **do;**
27   $ComputePC(\mathcal{G}_w, w)$;

the cd value for vertices in V' (Lines 9 and 10). In the case of $c(u) \neq c(v)$, we only invoke Procedure $SearchVertex$ to traverse from root vertex (Lines 11 and 12). Let k be the core number of root vertex $c(r)$ and sort cd values in an increasing order (Lines 13 and 14). In the next step, we select a vertex w with the minimum cd value in the set V', then judge whether $cd(w)$ value is smaller than k (Lines 15 and 16). If $cd(w)$ is less than k, then it means w has no more than k neighbors such that $c(w) \geqslant k$, i.e., u can no longer keep $c(w) = k$. In consequence, its core number decreases to $k-1$ (Line 17). The decrease in $c(w)$ may lead to the decrease of the pp-core numbers of its neighbors, then we put all its neighbors into the set S (Lines 18–20). Meanwhile, the cd value of the neighbor t of w which is larger than $cd(w)$ will also decrease, as it decreases a neighbor w whose core number is larger than k. Then we reorder the cd values accordingly (Line 21–23). If $cd(w) \geqslant k$, the algorithm will exit the loop because all remaining vertices in V' are still in a k-core (Lines 24 and 25). For each vertex w in S, we call $ComputePC$ to recalculate their pp-core numbers (Lines 26 and 27).

In Algorithm 7, it first initializes some parameters like Algorithm 3 (Lines 1–5). While Q is not empty, the algorithm will pop the vertex v one by one and put it into V' (Lines 6–8). We next check each neighbor w of v, if the vertex w satisfies the condition that $c(w) \geqslant k$, then we increase the cd value of v by 1 (Lines 9–11). Finally, we put all vertices whose core numbers are equal to k and are not visited into Q, and set their *visited* value to be 1 (Lines 12–14).

Algorithm complexity. Procedure $SearchVertex$ does a depth-first traversal from the root vertex, the worst case happens when all vertices are be traversed, thus the time complexity of procedure $SearchVertex$ is $O(|E|)$. After the vertex set V' is found, it will take $O(|V'|)$ time to search S, which is the vertex set of the

Algorithm 7 $\boldsymbol{SearchVertex(\mathcal{G}(\mathcal{V},\mathcal{E}), c(), pc(), w)}$

Input: $\mathcal{G}$: pp-graph; $c()$: core number of each vertex in public graph; r: root vertex

Output: The set of vertex V' that has its core number updated and the cd values for all vertices

1 $V' \leftarrow \varnothing$;
2 $Q \leftarrow \varnothing$; $Q.push(r)$;
3 $cd\ [w] = 0$, $visited\ [w] = 0, \forall w \in \mathcal{V}$;
4 $k \leftarrow c(r)$;
5 $visited\ [r] = 1$;
6 **while** $Q \neq \varnothing$ **do**
7     $v \leftarrow Q.pop()$;
8     $V'.push(v)$;
9   **for each** $(v, w) \in \mathcal{E}$ **do**
10   **if** $c(w) \geqslant k$ **then**
11             $cd\ [v] \leftarrow cd\ [v] + 1$;
12   **if** $c(w) = k$ and not $visited\ [w]$ **then**
13             $Q.push(w)$;
14             $visited\ [w] \leftarrow 1$;

neighbors of vertices in V'. According to the time complexity of $ComputePC$, the computation of the pp-core number in S will take $O(|S|\Delta\log\Delta)$. In total, the time complexity for public edge deletion algorithm is $O(|V'| + |E| + |S|\varDelta\log\varDelta)$.

5 Experiment

In this section, we conduct empirical studies to evaluate the performance of our proposed algorithms. We evaluate the algorithms on 4 real-world graphs, as shown in Table 1. All programs were implemented in Java language and compiled with IntelliJ IDEA, and all experiments were performed on a machine with Intel Core i5-7 500 3.41 GHz and 8 GB DDR3-RAM in Windows 10.

Datasets. All the real graphs we used are downloaded from KONECT[§]. These datasets are all undirected and unweighted graphs representing different social network structures, where vertices represent users and edges represent relationships

Table 1 Real-world graph datasets.

Dataset	$\lvert V\rvert$	$\lvert E_{public}\rvert$	$\lvert E_{private}\rvert$	deg_{max}	c_{max}	pc_{max}
HF (hamster friend)	1858	12 534	1525	89	17	17
AC (as-caida)	26 475	50 842	2539	2400	14	14
HE (HR-edges)	54 572	451 382	46 820	372	18	18
DB (douban)	154 908	308 182	18 980	287	13	13

[§]http://konect.cc/networks/

between users. We did a little bit of processing to turn the downloaded datasets into pp-graphs by setting some edges as private. The specific approach is: for each vertex, we randomly select one of its adjacent edges, if the other end vertex of the edge has no less than 8 neighbors and the number of private edges is less than one eighth of all adjacent edges associated with this end vertex, then we make the edge private. By traversing all vertices of the graph, we get a pp-graph $\mathcal{G}$. Table 1 provides the details about each used pp-graph, including the size of their vertex $|V|$ and edge set $|E_{public}|$, maximum degree of all vertices deg_{max}, the maximum public-core value c_{max}, and the maximum pp-core number pc_{max}. In addition, we used four pp-graphs of PP-DBLP-2013† by randomly selecting different numbers of vertices, respectively. All graphs are undirected.

Figure 3 shows the cumulative distribution of public-core and pp-core number for all graphs in Table 1. Here HE-k is the cumulative public-core number distribution in pp-graph HE, while HE-pp is the cumulative pp-core number distribution in pp-graph. Figure 3 shows that the curve representing pp-core number is roughly below the curve representing public-core number, and the line representing pp-core number first drops and then rises compared with the line representing the public-core number. This means that for the smaller value, the number of vertices whose pp-core number is equal to this value is less than the number of vertices whose public-core number is equal to this value. Instead, for the larger value, the number of vertices whose pp-core number is equal to this value is more than the number of vertices whose public-core number is equal to this value. It demonstrates that the number of vertices that have their pp-core number larger than their core number in a middle value is more than those in a smaller value and a bigger value. In addition, Fig. 3 also shows that the max pp-core number is equal to the max public-core number for all pp-graphs. The higher the ratio of private edges is, The larger the differences between the two lines (pp and k) in the same graph are. For example, the two lines in AC almost coincide, while there are some significant differences between the two lines in HE. We can find in Table 1 that the ratio of private edges to private edges in AC is about 0.05 and 0.10 in HE.

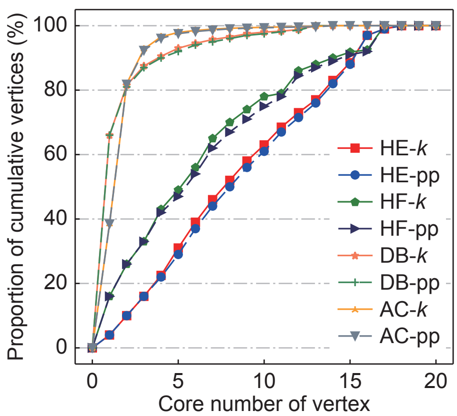


Fig. 3 Cumulative public core number and pp-core number distribution of real-world graphs.

†https://github.com/samjjx/pp-data

5.1 Performance evaluation

We first evaluate the experimental results of the core number and pp-core number computation algorithms, then we evaluate the impact of the size of inserted/deleted edges on core maintenance algorithms.

The time of core decomposition on real graphs is given in Fig. 4. In general, the time of computing public-core and pp-core number increases as the size of the graph grows. Figure 4 also shows that the vast majority of time is spent in calculating the public-core values when calculating pp-core number values of a pp-graph.

Next we evaluate the impact of the size of inserted/deleted edges on core maintenance algorithms after changing an edge, i.e., private edge insertion/deletion (Algorithm 2), public edge insertion

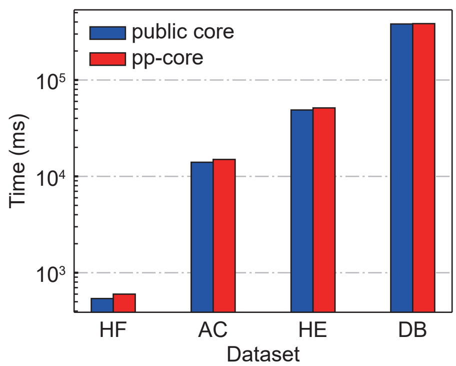


Fig. 4 Time for public core number and pp-core number computation in different real-world datasets.

(Algorithm 3), and public edge deletion (Algorithm 6). The evaluation is conducted on the four pp-graphs. In each pp graph, $E_i = 10^i$ public edges are selected randomly, where $i = 0, 1, 2, 3, 4$. These edges are first deleted from the original public graph to evaluate the deletion algorithm, and then are inserted back to evaluate the insertion algorithm. The processing time per edge for the edge insertion and deletion cases are shown in Figs. 5a and 5b, from which, we can see that there is a downward trend in the processing time per edge as the number of edges inserted/deleted increases. At the same time, there are some special cases where the time increases for individual graphs. In theory, the execution time of each edge should be about the same no matter how many public edges are added or removed. In fact, because of caching and other factors, the execution time per edge will have a downward trend when we continuously invoke pp-core number computation algorithms. For example, the time to compute the *MCD* and *PCD* values while executing the Traversal algorithm can decrease when the second edge is inserted/deleted. Comparing to recomputing pp-core numbers, it can be seen that core maintenance algorithms are more efficient when inserting/deleting a public edge. The reason is that only a few vertices whose pp-core numbers are changed when the graph changes only one edge, which is shown in Lemma 5.

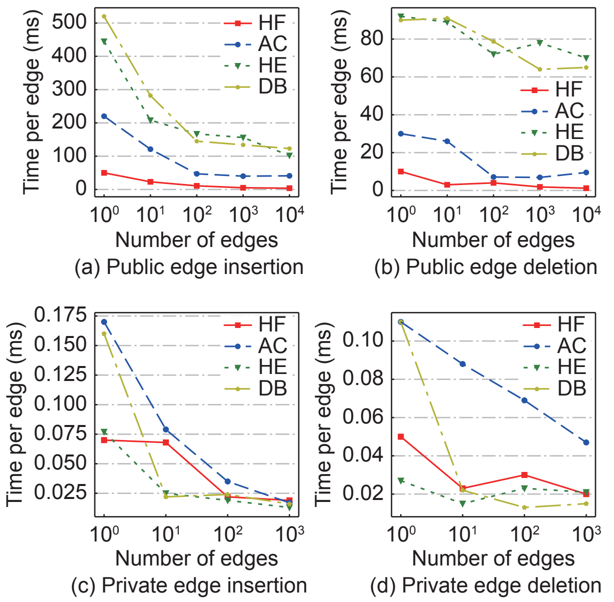


Fig. 5 Impact of the size of inserted/deleted edges on the private edge and public edge insertion/deletion algorithms.

The case of private edge insertion/deletion is easier than the public edge, since only two vertexes may change their pp-core number values when a private edge is inserted. Similar to public edge insertion/deletion, $P_i = 10^i$ private edges are selected randomly in each pp graph, where $i = 0, 1, 2, 3$. These private edges are first deleted from the original graphs to evaluate the deletion algorithm, and then are inserted back to evaluate the insertion algorithm. The processing time per edge for the deletion and insertion cases are shown in Figs. 5c and 5d, similarly, the trend is also generally downward as the number of edges inserted/deleted increases. The difference is that it takes less time than changing a public edge.

Next, we compare the time after public edges insertion of updating pp-core number only using Traversal algorithm and using SSN number additionally. In order to complete the comparison more accurately, we compare the time taken after traversal algorithm, one of which directly calculates the pp-core number, and the other optimizes the algorithm to further filter to calculate the pp-core number by using SSN. In Fig. 6, HF_original represents the original algorithm calculating the pp-core number directly and HF_optimization represents the optimized algorithm with SSN. From Fig. 6, it is obviously that pp-core number maintenance using SSN number can reduce

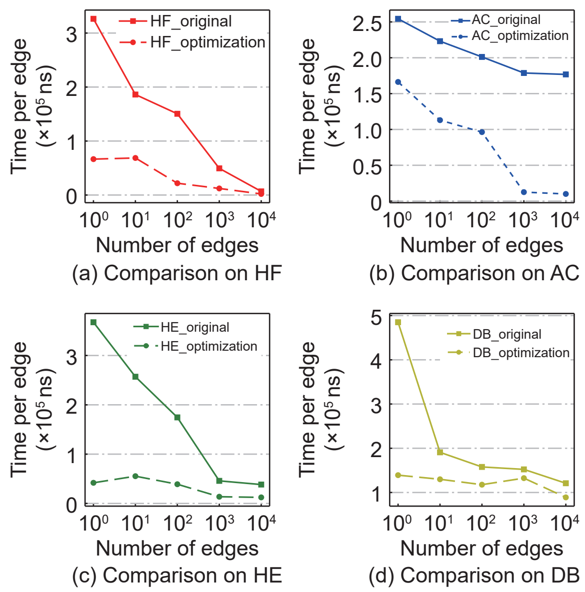


Fig. 6 Comparison of processing time per public edge with and without SSN.

time, and as the insertion edges increased, the processing time per edge decreases.

5.2 Scalability evaluation

We finally evaluate the scalability of our algorithms in four subgraphs of the pp-graph named PP-DBLP, by letting the number of vertices scale from 2^{13} to 2^{16}. The results are shown in Fig. 7. For each graph, we randomly select 100 edges to delete, and then insert them back. Figure 7 shows that as the size of the pp-graph increases exponentially, the average processing time to maintain the pp-core number increases when adding 100 public or private edges, and the increase tends to level off. At the same time, the time required for pp-core number is much larger when adding 100 public edges than adding 100 private edges. Whether adding public/private edges or removing public/private edges, the average processing time per edge tends to be stable when the graph size is large enough, which also demonstrates that our algorithm works well even when the pp-graphs have an extremely large size.

6 Related work

In previous studies of public graphs, there are various metrics to measure the importance of a node, such as degree, centrality, proximity, pageRank, katz, permeability, cross-centrality[15−18], k-core[7, 8], and k-truss[9, 19].

Some studies on pp-graphs have been investigated recently. In Ref. [20], the authors proposed a new model of attributed publicprivate networks and generated the corresponding PP-DBLP datasets with attributes from the rich keywords of paper titles on DBLP records. The public-private model of data summarization has been investigated and solved by a

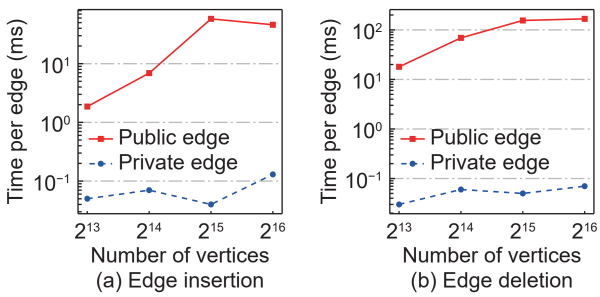


Fig. 7 Impact of the graph size on the private edge and public edge insertion/deletion algorithms.

fast distributed algorithm[21]. The problem of reachability indexing for pp-graphs was considered in Ref. [22], which aims to identify a set of additional visible seed nodes for each user. Moreover, the authors in Ref. [23] proposed a new keyword search framework *PPKWS* on public-private networks.

In addition, there are many relevant state-of-the-art developments in the field of core decomposition. The standard algorithm for computing k-core decomposition is the one originally proposed by Batagelj and Zaveršnik[13], which is based on the recursive deletion of vertices (and edges incident to them) of degree less than k. Cheng et al.[24] proposed a disk oriented algorithm in massive graphs, which can achieve comparable performances as the in-memory algorithm when the memory is large enough to hold the graph. Besides, core decomposition in the distributed setting was studied in Ref. [25], and the core number of the vertices is updated based on the core number of their neighbors. In Ref. [26], a parallel algorithm called Park was proposed to compute core numbers of vertices on multicore processors. Moreover, in Ref. [27], the authors propose a distributed algorithm for core decomposition on probabilistic graphs.

The problem of core maintenance has been studied extensively in recent years. Li et al.[25] published a report on incremental algorithms for core decomposition, and Saríyüce et al.[14] proposed a traversal algorithm with linear complexity, by which the speedup results achieved performed better. Moreover, Bai et al.[28] proposed a novel solution to tackle k-core maintenance of dynamic graphs, which provides an effective solution to maintain the core number of vertices affected by multiple inserted (removed) edges simultaneously. Yu et al.[29] presented fast algorithms for core maintenance in dynamic algorithms by processing multiple edges concurrently to improve core maintenance efficiency.

7 Conclusion

In this paper, we first propose a new definition of pp-core number of each vertex in a pp-graph, and give the algorithm of core computation with linear time and space complexity. In order to get the pp-core number of each vertex quickly when the pp-graph changes, we give core maintenance algorithms for public edge insertion/deletion and private edge insertion/deletion, respectively. Experiments on real-world graphs illustrate the efficiency and scalability of our algorithms. Future work is expected to further propose the core maintenance algorithms for multiple edges insertion/deletion.

References

[1] R. Dey, Z. Jelveh, and K. Ross, Facebook users have become much more private: A large-scale study, in *Proc. 2012 IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, Lugano, Switzerland, 2012, pp. 346−352.

[2] F. Chierichetti, A. Epasto, R. Kumar, S. Lattanzi, and V. Mirrokni, Efficient algorithms for public-private social networks, in *Proc. 21th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, Sydney, Australia, 2015, pp. 139−148.

[3] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy, A sketch-based distance oracle for web-scale graphs, in *Proc. 3rd ACM Int. Conf. on Web Search and Data Mining*, New York, NY, USA, 2010, pp. 401−410.

[4] T. H. Haveliwala, Topic-sensitive PageRank, in *Proc. 11th Int. Conf. on World Wide Web*, Honolulu, HI, USA, 2002, pp. 517−526.

[5] N. Bansal, A. Blum, and S. Chawla, Correlation clustering, *Mach. Learn.*, vol. 56, nos. 1–3, pp. 89–113, 2004.

[6] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, Preventing unraveling in social networks: The anchored *k*-core problem, *SIAM J. Discrete Math.*, vol. 29, no. 3, pp. 1452–1475, 2015.

[7] F. Zhang, W. J. Zhang, Y. Zhang, L. Qin, and X. M. Lin, OLAK: An efficient algorithm to prevent unraveling in social networks, *Proc. VLDB Endow.*, vol. 10, no. 6, pp. 649–660, 2017.

[8] F. Zhang, Y. Zhang, L. Qin, W. J. Zhang, and X. M. Lin, Finding critical users for social network engagement: The collapsed k-core problem, in *Proc. 31st AAAI Conf. on Artificial Intelligence*, San Francisco, CA, USA, 2017, pp. 245−251.

[9] F. Zhang, C. G. Li, Y. Zhang, L. Qin, and W. J. Zhang, Finding critical users in social communities: The collapsed core and truss problems, *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 1, pp. 78–91, 2020.

[10] S. B. Seidman, Network structure and minimum degree, *Soc. Netw.* , vol. 5, no. 3, pp. 269−287, 1983.

[11] Q. S. Hua, Y. L. Shi, D. X. Yu, H. Jin, J. G. Yu, Z. P. Cai, X. Z. Cheng, and H. H. Chen, Faster parallel core maintenance algorithms in dynamic graphs, *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1287–1300, 2020.

[12] V. Batagelj and M. Zaversnik, An O(m) algorithm for cores decomposition of networks, arXiv preprint arXiv: cs/0310049, 2003.

[13] V. Batagelj and M. Zaveršnik, Fast algorithms for determining (generalized) core groups in social networks, *Adv. Data Anal. Classi.* , vol. 5, no. 2, pp. 129−145, 2011.

[14] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K. L. Wu, and Ü. V. Çatalyürek, Streaming algorithms for k-core decomposition, *Proc. VLDB Endow.*, vol. 6, no. 6, pp. 433–444, 2013.

[15] M. A. Beauchamp, An improved index of centrality, *Behav. Sci.*, vol. 10, no. 2, pp. 161–163, 1965.

[16] P. Bonacich, Power and centrality: A family of measures, *Am. J. Sociol.*, vol. 92, no. 5, pp. 1170–1182, 1987.

[17] U. Brandes, A faster algorithm for betweenness centrality, *J. Math. Sociol.* , vol. 25, no. 2, pp. 163−177, 2001.

[18] L. C. Freeman, A set of measures of centrality based on betweenness, *Sociometry*, vol. 40, no. 1, pp. 35−41, 1977.

[19] Q. Luo, D. X. Yu, H. Sheng, J. G. Yu, and X. Z. Cheng, Distributed algorithm for truss maintenance in dynamic graphs, in *Proc. 21st Int. Conf. on Parallel and Distributed Computing, Applications and Technologies*, Shenzhen, China, 2021, pp. 104−115.

[20] X. Huang, J. Jiang, B. Choi, J. Xu, Z. Zhang, and Y. Song, PP-DBLP: Modeling and generating attributed public-private networks with DBLP, in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, doi: 10.1109/ICDMW.2018.00142.

[21] B. Mirzasoleiman, M. Zadimoghaddam, and A. Karbasi, Fast distributed submodular cover: Public-private data summarization, in *Proc. 30th Int. Conf. on Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 3601−3609.

[22] A. Archer, S. Lattanzi, P. Likarish, and S. Vassilvitskii, Indexing public-private graphs, in *Proc. 26th Int. Conf. on World Wide Web*, Perth, Australia, 2017, pp. 1461−1470.

[23] J. X. Jiang, X. Huang, B. Choi, J. L. Xu, S. S. Bhowmick, and L. Xu, PPKWS: An efficient framework for keyword search on public-private networks, in *Proc. 36th Int. Conf. on Data Engineering (ICDE)*, Dallas, TX, USA, 2020, pp. 457−468.

[24] J. Cheng, Y. P. Ke, S. M. Chu, and M. T. Özsu, Efficient core decomposition in massive networks, in *Proc. 27th Int. Conf. on Data Engineering*, Hannover, Germany, 2011, pp. 51−62.

[25] R. H. Li, J. X. Yu, and R. Mao, Efficient core maintenance in large dynamic graphs, *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2453–2465, 2014.

[26] N. Dasari, D. Ranjan, and M. Zubair, ParK: An efficient algorithm for k-core decomposition on multicore processors, in *Proc. 2014 IEEE Int. Conf. on Big Data*

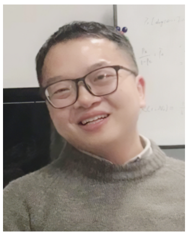
(*Big Data*), Washington, DC, USA, 2014, pp. 9−16.

[27] Q. Luo, D. X. Yu, F. Li, Z. H. Dou, Z. P. Cai, J. G. Yu, and X. Z. Cheng, Distributed core decomposition in probabilistic graphs, in *Proc. 8th Int. Conf. on Computational Data and Social Networks*, Ho Chi Minh City, Vietnam, 2019, pp. 16−32.

[28] W. Bai, Y. X. Zhang, X. Z. Liu, M. Chen, and D. Wu, Efficient core maintenance of dynamic graphs, in *Proc. 25th Int. Conf. on Database Systems for Advanced Applications*, Jeju, Republic of Korea, 2020, pp. 658−665.

[29] D. X. Yu, N. Wang, Q. Luo, F. Li, J. G. Yu, X. Z. Cheng, and Z. P. Cai, Fast core maintenance in dynamic graphs, *IEEE Trans. Comput. Soc. Syst.*, doi: 10.1109/TCSS. 2021.3064836.

Dongxiao Yu received the BS degree from Shandong University in 2006 and the PhD degree from the University of Hong Kong, China in 2014. He became an associate professor at the School of Computer Science and Technology, Huazhong University of Science and Technology in 2016. He is currently a professor at the School of Computer Science and Technology, Shandong University. His research interests include wireless networks, distributed computing, and graph algorithms.



Qi Luo received the BEng degree in computer science from Northeastern University at Qinhuangdao, China in 2015, and the MEng degree from Shandong University, China in 2018. He is currently a PhD candidate at the School of Computer Science and Technology, Shandong University. His research interests include graph mining and analysis.

Lifang Zhang received the BEng degree from Shandong University in 2019. She is currently a master student at Shandong University. Her research interests include graph analysis and data mining.

Zhenzhen Xie received the MEng degree in computer science from Jilin University, China in 2014, she is currently a PhD candidate at the School of Computer Science and Technology, Shandong University. Her research areas are reinforcement learning, IoTs, and representation learning.

Xilian Zhang received the BEng degree from Shandong University at Weihai, China in 2019. She is currently a master student at Shandong University. Her research interests include graph analysis and data mining.

Zhipeng Cai is currently an associate professor at the Department of Computer Science, Georgia State University, USA. He received the PhD and MEng degrees from University of Alberta, USA in 2008 and 2004, respectively, and the BEng degree from Beijing Institute of Technology, China in 2001. His research areas focus on wireless networking, IoTs, machine learning, cyber-security, and big data. He is the recipient of an NSF CAREER Award. He served as a steering committee co-chair and a steering committee member for WASA and IPCCC. He also served as a technical program committee member for more than 20 conferences, including INFOCOM, MOBIHOC, ICDE, and ICDCS. He has been serving as an associate editor-in-chief for Elsevier *High-Confidence Computing Journal* (*HCC*), and an associate editor for several international journals, such as *IEEE Internet of Things Journal* (*IoT-J*), *IEEE Transactions on Knowledge and Data Engineering* (*TKDE*), and *IEEE Transactions on Vehicular Technology* (*TVT*). He has published more than 70 papers in prestigious journals with more than 40 papers published in IEEE/ACM Transactions.