Index-Based Intimate-Core Community Search in Large Weighted Graphs

Longxu Sun, Xin Huang, Rong-Hua Li, Byron Choi, and Jianliang Xu

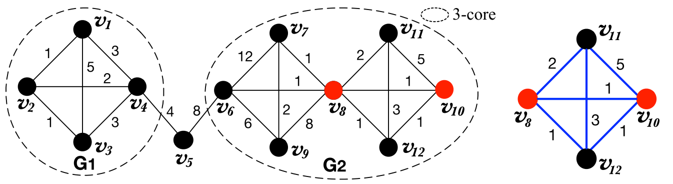
Abstract—Community search that finds query-dependent communities has been studied on various kinds of graphs. As one instance of community search, intimate-core group (community) search over a weighted graph is to find a connected k-core containing all query nodes with the smallest group weight. However, existing state-of-the-art methods start from the maximal k-core to refine an answer, which is practically inefficient for large networks. In this paper, we develop an efficient framework, called local exploration k-core search (LEKS), to find intimate-core groups in graphs. We propose a small-weighted spanning tree to connect query nodes, and then expand the tree level by level to a connected k-core, which is finally refined as an intimate-core group. In addition, to support the intimate group search over large weighted graphs, we develop a weighted-core index (WC-index) and two new WC-index-based algorithms for expansion and refinement phases in LEKS. Specifically, we propose a WC-index-based expansion to efficiently find a candidate graph of intimate-core group, leveraging on a two-level expansion of k-breadth and 1-depth. We propose two graph removal strategies: coarse-grained refinement is designed for large graphs to delete a batch of nodes in a few iterations; fine-grained refinement is designed for small graphs to remove nodes carefully and achieve high-quality answers. Extensive experiments on real-life networks with ground-truth communities validate the effectiveness and efficiency of our proposed methods.

Index Terms—Graph mining, weighted graphs, k-core, community search

1 Introduction

Graphs widely exist in social networks, biomolecular structures, traffic networks, world wide web, and so on. Weighted graphs have not only the simple topological structure but also edge weights. The edge weight is often used to indicate the strength of the relationship, such as interval in social communications, traffic flow in the transportation network, carbon flow in the food chain, and so on [1], [2], [3]. Weighted graphs provide information that better describes the organization and hierarchy of the network, which is helpful for community detection [3] and community search [4], [5], [6], [7]. Community detection aims at finding all communities on the entire network, which has been studied a lot in the literature. Different from community detection, the task of community search finds only query-dependent communities, which has a wide application of disease infection control, tag recommendation, and social event organization [8], [9]. Recently, several community search models have been proposed in different dense subgraphs of k-core [10], [11] and k-truss [7], [12].

As a notation of dense subgraph, k-core requires that every vertex has k neighbors in the k-core. For example, Fig. 1a shows a graph G. Subgraphs G_1 and G_2 are both connected 3-cores, in which each vertex has at least three neighbors. K-core has been popularly used in many community search models [8], [13], [14], [15], [16], [17]. Recently, Zheng *et al.* [9] proposed one problem of intimate-core group search in weighted graphs as follows.



(a) Graph G (b) Intimate-core group

Fig. 1. An example of intimate-core group search in graph G for $Q = \{v_8, v_{10}\}$ and $k = 3$.

Motivating Example. Consider a social network G in Fig. 1a. Two individuals have a closer friendship if they have a shorter interval for communication, indicating a smaller weight of the relationship edge. The problem of intimate-core group search aims at finding a densely-connected k-core containing query nodes Q with the smallest group weight as an answer. For $Q = \{v_8, v_{10}\}$ and $k = 3$, the intimate-core group is shown in Fig. 1b with a minimum group weight of 13.

This paper studies the problem of intimate-core group search in weighted graphs. Given an input of query nodes in a graph and a number k, the problem is to find a connected k-core containing query nodes with the smallest weight. With the consideration of edge weight, this problem can discover a community personalized related to query nodes, which has intimate internal connections and high cohesiveness. In real life, the intimate-core group search has a wide

- *Longxu Sun, Xin Huang, Byron Choi, and Jianliang Xu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, China. E-mail: {cslxsun, xinhuang, bchoi, xujl}@comp.hkbu.edu.hk.*
- *Rong-Hua Li is with the Beijing Institute of Technology, Beijing, China. E-mail: lironghuabit@126.com.*

Manuscript received 25 Dec. 2019; revised 11 Oct. 2020; accepted 12 Nov. 2020. Date of publication 26 Nov. 2020; date of current version 5 Aug. 2022. (Corresponding author: Xin Huang.) Recommended for acceptance by A. Khan. Digital Object Identifier no. 10.1109/TKDE.2020.3040762

range of applications, such as collaboration group search [9], tag recommendation [8] and infectious disease control [9].

In the literature, existing solutions proposed in [9] find the maximal connected k-core and iteratively remove a node from this subgraph for intimate-core group refinement. However, this approach may take a large number of iterations, which is inefficient for big graphs with a large component of k-core. Therefore, we propose a solution of local exploration to find a small candidate k-core, which takes a few iterations to find answers. To further speed up the efficiency, we build a k-core index, which keeps the structural information of k-core for fast identification. Based on the k-core index, we develop a local exploration algorithm LEKS for intimate-core group search. Our algorithm LEKS first generates a tree to connect all query nodes, and then expands it to a connected subgraph of k-core. Finally, LEKS keeps refining candidate graphs into an intimate-core group with small weights. We propose several well-designed strategies for LEKS to ensure the fast-efficiency and high-quality of answer generations.

Over the conference version [18] of this manuscript, we further investigate the problem of intimate-core group search and propose efficient index-based algorithms in large weighted graphs in Section 5. The motivation is that the existing index proposed k-core index in Section 4.1 keeps no records of the important information of edge weights. To this end, we propose a new data structure of Weighted-Core index (WC-index), which keeps the node corenesses and edge weights. For each vertex, WC-index sorts its neighbors in the increasing order of their edge weights from low to high. This index is simple but particularly useful to speedup the tree-to-graph expansion and intimate-core refinement phases. The basic tree-to-graph strategy expands from nodes in the tree to all their neighbors, which may construct a large-size graph. It requires a large number of deletions and can influence the efficiency of the refinement. Also, the ignoring of edge weight leads to a large group weight of the candidate graph. To overcome the drawbacks, we design a two-level WC-index-based expansion strategy consists of k-breadth expansion and 1-depth expansion. Integrating both methods, it constructs a candidate graph with both small size and small group weight. For the refinement phase, we propose a minimal-weight-based removal order to identify nodes with weak relationships to the graph. Moreover, we design a coarse-grained binary deletion strategy for large graphs to improve efficiency. We also propose a fine-grained deletion strategy for small graphs that finally constructs a high-quality community with a small weight.

We conduct extensive experiments of effectiveness and efficiency evaluations of our algorithms on large real-life datasets of weighted graphs with ground-truth communities. First, we compare an existing method ICG-M, LEKS methods, and our WC-index-based method on weighted networks. We find that our methods always have higher efficiency and better effectiveness than ICG-M. Second, our proposed index is compact and useful. The WC-index-based algorithm is highly efficient especially on large graphs. Last but not least, the quality evaluations on ground-truth communities confirm that our proposed WC-index-based method achieves higher-quality communities than state-of-the-art methods of LEKS [18] and ICG-M [9].

Contributions. Our main contributions of this paper are summarized as follows.

- We investigate and tackle the problem of intimate-core group search in weighted graphs, which has wide applications on real-world networks. The problem is NP-hard, which brings challenges to develop efficient algorithms.
- We develop an efficient local exploration framework of LEKS based on the k-core index for intimate-core group search. LEKS consists of three phases: tree generation, tree-to-graph expansion, and intimate-core refinement.
- In the phase of tree generation, we propose to find a seed tree to connect all query nodes, based on two generated strategies of *spanning tree* and *weighted path* respectively. Next, we develop the tree-to-graph expansion, which constructs a hierarchical structure by expanding a tree to a connected k-core subgraph level by level. Finally, we refine a candidate k-core to an intimate-core group with a small weight. During the phases of expansion and refinement, we design a protection mechanism for query nodes, which protects critical nodes to collapse the k-core.
- We design a useful index, named WC-index, which keeps the node corenesses and edge weights in graphs. Based on WC-index, we propose several improved strategies to speedup the expansion and refinement phases. We develop a WC-index-based expansion algorithm using a two-level expansion of k-breadth and 1-depth, which can find a small candidate graph efficiently. Moreover, we develop a WC-index-based refinement algorithm using a hybrid removal strategy. It performs the coarse-grained refinement over large graphs, which deletes a batch of nodes in a few iterations. It performs the fine-grained refinement over small graphs, which removes nodes carefully and achieve high-quality answers.
- Our experimental evaluation demonstrates the effectiveness and efficiency of our LEKS algorithms on large weighted graphs with ground-truth communities. We show the superiority of our methods in finding intimate groups with smaller weights, against the state-of-the-art methods [9], [18].

Roadmap. The rest of the paper is organized as follows. Section 2 reviews the previous work related to ours. Section 3 presents the basic concepts and formally defines our problem. Section 4 introduces our index-based local exploration approach LEKS. Section 5 proposes a WC-index and two new algorithms of WC-index-based expansion and WC-index-based refinement. Section 6 presents the experimental evaluation. Finally, Section 7 concludes the paper.

2 Related Work

Our work is related to the topics of community search, community detection, and minimum subgraph mining.

Community Search. In the literature, numerous studies have been investigated community search based on various kinds of dense subgraphs, such as k-core [10], [11], k-truss [7], [12] and clique [5], [19]. Community search has been

TABLE 1
A Comparison of Existing Community Search Studies and Ours

Method	Dense Subgraph Model	Node Type	Edge Type	Local Search	Index-based	Multiple Query Nodes	NP-hard
[19]	clique	×	×	✓	✓	×	✓
[5]	clique	×	×	×	✓	✓	✓
[20]	k-truss	×	×	✓	✓	✓	✓
[16], [17]	k-core	×	×	×	×	×	✓
[21]	k-core	×	×	✓	×	×	✓
[8]	k-core	×	×	×	×	✓	✓
[15]	k-core	×	×	✓	✓	✓	✓
[22]	k-truss	keyword	×	✓	✓	✓	✓
[13]	k-core	keyword	×	✓	✓	×	✓
[14]	k-core	influential	×	×	✓	×	×
[23]	k-core	influential	×	✓	×	×	×
[24]	k-truss	×	weighted	✓	✓	×	×
[9]	k-core	×	weighted	×	×	✓	✓
Ours	k-core	×	weighted	✓	✓	✓	✓

also studied on many labeled graphs, including weighted graphs [9], [24], [25], influential graphs [14], [23], and keyword-based graphs [13], [22], [26]. The additional information sometimes provides to nodes or to edges. Besides, the containing constraint is given one or more query nodes and requires them included in the community [13], [22]. For example, [15] studies the community search problem with multiple query nodes using the k-core model. However, this work is to query communities on simple unweighted graphs, which may ignore some useful information. [30] is to find k-truss communities on attributed graphs, the node in the graph has one or more keywords. Table 1 compares different characteristics of existing community search studies and ours in terms of dense subgraph models, graph types, search algorithms, query nodes, and problem hardness.

Community Detection. Community detection aims to determine all densely-connected communities in the entire network [27]. There exist various studies of community detection in the literature. Zhang *et al.* study the community detection on graphs with node features [28]. Li *et al.* use an embedding approach to solve community detection problems in attributed graphs [27]. [29] aims to find hierarchy community structures in attributed graphs. There are many applications about community detection, such as disease module identification in protein–protein interaction networks [30], discover multistage video clusters with related topics on Youtube [31] and so on.

Minimum Subgraph Mining. Minimum subgraph mining investigates various problems of finding the minimum subgraph satisfying the given objectives/constraints of dense subgraphs and communication costs. The problem of k-core minimization [15], [16], [17], [21] aims to find a minimal connected k-core subgraph containing query nodes. Barbieri *et al.* proved that it is NP-Hard [15]. [15] proposed a connected k-core index and local search methods. [16] used KC-Edge method by greedy deleting edges from the graph. [17] defined Shapley value to measure the joint effect of edges to choose priority deleted edges. The minimum wiener connector problem is finding a small connected subgraph to minimize the sum of all pairwise shortest-path distances between the discovered vertices [32].

Different from all the above studies, our work aims at finding an intimate-core group containing multiple query nodes in weighted graphs. We propose fast algorithms for intimate-core group search, which outperform the state-of-the-art method [9] in terms of quality and efficiency.

3 Preliminaries

In this section, we formally define the problem of intimate-core group search and revisit the existing intimate-core group search approaches. Table 2 lists the notations that we will frequently use in this paper.

3.1 Problem Definition

Let $G(V, E, w)$ be a weighted and undirected graph where V is the set of nodes, E is the set of edge, and w is an edge weight function. Let $w(e)$ to indicate the weight of an edge $e \in E$. The number of nodes in G is defined as $n = |V|$. The number of edges in G is defined as $m = |E|$. We denote the set of neighbors of a node v by $N_G(v) = \{u \in V : (u, v) \in E\}$, and the degree of v by $deg_G(v) = |N_G(v)|$. When the context is obvious, we drop the subscript such as using $deg(v)$ instead of $deg_G(v)$. For example, Fig. 1a shows a weighted graph G. Node v_5 has two neighbors as $N_G(v_5) = \{v_4, v_6\}$, thus the degree of v_5 is $deg_G(v_5) = 2$ in graph G. Edge (v_2, v_3) has a weight of $w(v_2, v_3) = 1$. Based on the definition of degree, we can define the k-core as follows.

Definition 1 (K-Core [10]). *Given a graph G, the k-core is the largest subgraph H of G such that every node v has degree at least k in H, i.e., $deg_H(v) \geq k$.*

For a given integer k, the k-core of graph G is denoted by $C_k(G)$, which is determinative and unique by the definition

TABLE 2
Frequently Used Notations

Notation	Description
$G = (V, E, w)$	A weighted, undirected simple graph G
$w(e); w(u, v)$	Weight of edge e; weight of edge (u, v)
$n; m$	Number of nodes/edges in graph G
$N_G(v)$	The neighbors for node $v \in V_G$
$deg_G(v)$	The degree of node $v \in V_G$
Q	The set of query vertices
$\delta(v)$	The coreness of node v in graph G
$C_k(G)$	The k-core subgraph in graph G
$C^*_k(G)$	The connected k-core subgraph
$w(G)$	Group weight of graph G
H	The intimate-core group

of largest subgraph constraint. For example, the 3-core of G in Fig. 1a has two components G_1 and G_2. Every node has at least 3 neighbors in G_1 and G_2 respectively. However, the nodes are disconnected between G_1 and G_2 in the 3-core $C_3(G)$. To incorporate connectivity into k-core, we define a connected k-core.

Definition 2 (Connected K-Core). *Given graph G and number k, a connected k-core H is a connected component of G such that every node v has degree at least k in H, i.e., $deg_H(v) \geq k$.*

Intuitively, all nodes are reachable in a connected k-core, i.e., there exist paths between any pair of nodes. G_1 and G_2 are two connected 3-cores in Fig. 1a.

Definition 3 (Group Weight). *Given a subgraph $H \subseteq G$, the group weight of H, denoted by $w(H)$, is defined as the sum of all edge weights in H, i.e., $w(H) = \sum_{e \in E(H)} w(e)$.*

Example 1. For the subgraph $G_1 \subseteq G$ in Fig. 1a, the group weight of G_1 is $w(G_1) = \sum_{e \in E(G_1)} w(e) = 1 + 3 + 5 + 2 + 1 + 3 = 15$.

On the basis of the definitions of connected k-core and group weight, we define the *intimate-core group* in a graph G as follows.

Definition 4 (Intimate-Core Group [9]). *Given a weighted graph $G = (V, E, w)$, a set of query nodes Q and a number k, the intimate-core group is a subgraph H of G if H satisfies following conditions:*

- Participation. *H contains all the query nodes Q, i.e., $Q \subseteq V_H$;*
- Connected K-Core. *H is a connected k-core with $deg_H(v) \geq k$;*
- Smallest Group Weight. *The group weight $w(H)$ is the smallest, that is, there exists no $H' \subseteq G$ achieving a group weight of $w(H') < w(H)$ such that H' also satisfies the above two conditions.*

Condition (1) of participation makes sure that the intimate-core group contains all query nodes. Moreover, Condition (2) of connected k-core requires that all group members are densely connected with at least k intimate neighbors. In addition, Condition (3) of minimized group weight ensures that the group has the smallest group weight, indicating the most intimate in any kinds of edge semantics. A small edge weight means a high intimacy among the group. Overall, intimate core groups have several significant advantages of small-sized group, offering personalized search for different queries, and close relationships with strong connections.

The problem of *intimate-core group search* studied in this paper is formulated in the following.

Problem Formulation. Given an undirected weighted graph $G(V, E, w)$, a number k, and a set of query nodes Q, the problem is to find the intimate-core group of Q.

Example 2. In Fig. 1a, G is a weighted graph with 12 nodes and 20 edges. Each edge has a positive weight. Given two query nodes $Q = \{v_8, v_{10}\}$ and $k = 3$, the answer of intimate-core group for Q is the subgraph shown in Fig. 1b. This is a connected 3-core, and also containing two query nodes $\{v_8, v_{10}\}$. Moreover, it has the minimum group weight among all connected 3-core subgraphs containing Q.

3.2 Existing Intimate-Core Group Search Algorithms

The problem of intimate-core group search has been studied in the literature [9]. Two heuristic algorithms, namely, ICG-S and ICG-M, are proposed to deal with this problem in an online manner. No optimal algorithms have been proposed yet because this problem has been proven to be NP-hard [9]. The NP-hardness is shown by reducing the NP-complete clique decision problem to the intimate-core group search problem.

Existing solutions ICG-S and ICG-M both first identify a maximal connected k-core as a candidate, and then remove the node with the largest weight of its incident edges at each iteration [9]. The difference between ICG-S and ICG-M lies on the node removal. ICG-S removes one node at each iteration, while ICG-M removes a batch of nodes at each iteration. Although ICG-M can significantly reduce the total number of removal iterations required by ICG-S, it still takes a large number of iterations for large networks. The reason is that the initial candidate subgraph connecting all query nodes is the maximal connected k-core, which may be too large to shrink. This, however, is not always necessary. In particular, if there exists a small connected k-core surrounding query nodes, then a few numbers of iterations may be enough token for finding answers. This paper proposes a local exploration algorithm to find a smaller candidate subgraph. On the other hand, both ICG-S and ICG-M apply the core decomposition to identify the k-core from scratch, which is also costly expensive. To improve efficiency, we propose to construct an index offline and retrieve k-core for queries online.

4 Index-Based Local Exploration Algorithms

In this section, we first introduce a useful core index and the index construction algorithm. Then, we present the index-based intimate-core group search algorithms using local exploration.

4.1 K-Core Index

We start with a useful definition of coreness as follows.

Definition 5 (Coreness). *The coreness of a node $v \in V$, denoted by $\delta(v)$, is the largest number k such that there exists a connected k-core containing v.*

Obviously, for a node q with the coreness $\delta(q) = l$, there exists a connected k-core containing q where $1 \leq k \leq l$; meanwhile, there is no connected k-core containing q where $k > l$. The k-core index keeps the coreness of all nodes in G.

K-core Index Construction. We apply the existing core decomposition [10] on graph G to construct the k-core index. The algorithm is outlined in Algorithm 1. The core decomposition is to compute the coreness of each node in graph G. Note that for the self-completeness of our techniques and

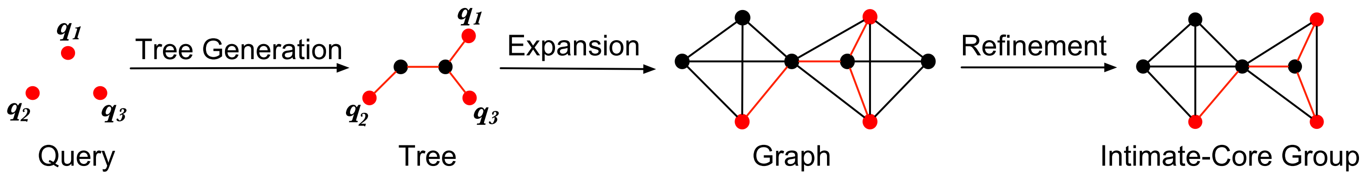


Fig. 2. LEKS framework for intimate-core group search.

reproducibility, the detailed algorithm of core decomposition is also presented (lines 1-7). First, the algorithm sorts all nodes in G based on their degree in ascending order. Second, it finds the minimum degree in G as d. Based on the definition of k-core, it next computes the coreness of nodes with $deg_G(v) = d$ as d and removing these nodes and their incident edges from G. With the deletion of these nodes, the degree of neighbors of these nodes will decrease. For those nodes which have a new degree at most d, they will not be in (d+1)-core while they will get $\delta(v) = d$. It continues the removal of nodes until there is no node has $deg_G(v) \leq d$. Then, the algorithm back to line 2 and starts a new iteration to compute the coreness of remaining nodes. Finally, it stores the coreness of each vertex v in G as the k-core index.

Algorithm 1. Core Index Construction

Input: A weighted graph $G = (V, E, w)$
Output: Coreness $\delta(v)$ for each $v \in V_G$
1: Sort all nodes in G in ascending order of their degree;
2: **while** $G \neq \emptyset$
3: Let d be the minimum degree in G;
4: **while** there exists $deg_G(v) \leq d$
5: $\delta(v) \leftarrow d$;
6: Remove v and its incident edges from G;
7: Re-order the remaining nodes in G in ascending order of their degree;
8: Store $\delta(v)$ in index for each $v \in V_G$;

4.2 Solution Overview

At a high level, our algorithm of local exploration based on k-core index for intimate-core group search (LEKS) consists of three phases:

1) *Tree Generation Phase*: This phase invokes the shortest path algorithm to find the distance between any pair of nodes, and then constructs a small-weighted tree by connecting all query nodes.
2) *Expansion Phase*: This phase expands a tree into a graph. It applies the idea of local exploration to add nodes and edges. Finally, it obtains a connected k-core containing all query nodes.
3) *Intimate-Core Refinement Phase*: This phase removes nodes with large weights, and maintains the candidate answer as a connected k-core. This refinement process stops until an intimate-core group is obtained.

Fig. 2 shows the whole framework of our index-based local exploration algorithm. Note that we compute the k-core index offline and apply the above solution of online query processing for intimate-core group search. In addition, we consider $|Q| \geq 2$ for tree generation phase, and skip this phase if $|Q| = 1$. Algorithm 2 also depicts our algorithmic framework of LEKS.

Algorithm 2. LEKS Framework

Input: $G = (V, E, w)$, an integer k, a set of query vertices Q
Output: Intimate-core group H
1: Find a tree T_Q for query nodes Q using Algorithms 3 or 4;
2: Expand the tree T_Q to a candidate graph G_Q in Algorithm 5;
3: Apply ICG-M [9] on graph G_Q;
4: Return a refined intimate-core group as answers;

4.3 Tree Generation

In this section, we present the phase of tree generation. Due to the large-scale size of k-core in practice, we propose local exploration methods to identify small-scale substructures as candidates from the k-core. The approaches produce a tree structure with small weights to connect all query nodes. We develop two algorithms, respectively based on the minimum spanning tree (MST) and minimum weighted path (MWP).

Tree-based Construction. The tree-based construction has three major steps. Specifically, the algorithm first generates all-pairs shortest paths for query nodes Q in the k-core C_k (lines 1-7). Given a path between nodes u and v, the path weight is the total weight of all edges along this path between u and v. It uses $\text{spath}_{C_k}(u, v)$ to represent the shortest path between nodes u and v in the k-core C_k. For any pair of query nodes q_i, $q_j \in Q$, our algorithm invokes the well-known Dijkstra's algorithm [33] to find the shortest path $\text{spath}_{C_k}(q_i, q_j)$ in the k-core C_k.

Second, the algorithm constructs a weighted graph G_{pw} for connecting all query nodes (lines 3-8). Based on the obtained all-pairs shortest paths, it collects and merges all these paths together to construct a weighted graph G_{pw} correspondingly.

Third, the algorithm generates a small spanning tree for Q in the weighted graph G_{pw} (lines 9-22), since not all nodes or edges are needed to keep the query nodes connected in G_{pw}. This step finds a compact spanning tree to connect all query nodes Q, which removes useless components to reduce weights. Specifically, the algorithm starts from one of the query nodes and does expand based on Prim's minimum spanning tree algorithm [33]. The algorithm stops when all query nodes are connected into a component in G_{pw}. Against the maximal connected k-core, our compact spanning tree has three significant features: (1) Query-centric. The tree involves all query nodes of Q; (2) Compactly connected. The tree is a connected and compact structure; (3) Small-weighted. The generation of minimum spanning tree ensures a small weight of the discovered tree.

Path-Based Construction. Algorithm 3 may take expensive computation for finding the shortest path between every pair of nodes that are far away from each other. To improve efficiency, we develop a path-based approach to connect all query nodes directly. The path-based construction is outlined in Algorithm 4.

Algorithm 3. Tree Construction

Input: $G=(V,E,w)$, an integer k, a set of query vertices Q, the k-core index
Output: Tree T_Q

1: Identify the maximal connected k-core of C_k containing query nodes Q;
2: Let G_{pw} be an empty graph;
3: **for** $q_1, q_2 \in Q$
4:   **if** there is no path between q_1 and q_2 in C_k **then**
5:     **return** $\emptyset$;
6:   **else**
7:     Compute the shortest path between q_1 and q_2 in C_k;
8:     Add the $\mathsf{spath}_{C_k}(q_1,q_2)$ between q_1 and q_2 into G_{pw};
9: Tree: $T_Q \leftarrow \emptyset$;
10: Priority queue: $L \leftarrow \emptyset$;
11: **for** each node v in G_{pw}
12:   $\mathsf{dist}(v) \leftarrow \infty$;
13: $Q \leftarrow Q - \{q_0\}$; $\mathsf{dist}\,(q_0) \leftarrow 0$; $L.push(q_0, \mathsf{dist}(q_0))$;
14: **while** $Q \neq \emptyset$ **do**
15:   Extract a node v and its edges with the smallest $\mathsf{dist}(v)$ from L;
16:   Insert node v and its edges into T_Q;
17:   **if** $v \in Q$ **then**
18:     $Q \leftarrow Q - \{v\}$;
19:   **for** $u \in N_{G_{pw}}(v)$ **do**
20:     **if** $\mathsf{dist}(u) > w(u,v)$ **then**
21:       $\mathsf{dist}(u) \leftarrow w(u,v)$;
22:       Update $(u, \mathsf{dist}(u))$ in L;
23: **return** T_Q;

Algorithm 4. Path-Based Construction

Input: $G=(V,E,w)$, an integer k, a set of query vertices Q, the k-core index
Output: Tree T_Q

1: Identify the maximal connected k-core of C_k containing query nodes Q;
2: Extract a query node $q_0 \in Q$ randomly;
3: $Q \leftarrow Q - \{q_0\}$;
4: **while** $Q \neq \emptyset$ **do**
5:   **if** there is no path between any query node $q \in Q$ and q_0 in C_k **then**
6:     **return** $\emptyset$;
7:   **else**
8:     Apply Dijkstra's algorithm [33] to compute the shortest path from q_0 to the nearest $q^* \in Q$ in C_k;
9:     Add the $\mathsf{spath}_{C_k}(q_0,q^*)$ between q_0 and q^* into T_Q;
10:     $q_0 \leftarrow q^*$, $Q \leftarrow Q - \{q^*\}$;
11: **return** T_Q;

First, the algorithm identifies the k-core C_k (line 1). Then, it randomly selects one query node $q_0 \in Q$ and removes q_0 from Q (lines 2-3). Starting from q_0, it applies the Dijkstra's algorithm [33] to find the shortest path from q_0 to the nearest query node $q^* \in Q$ (lines 5-8). Note that the query node q^* is determined by the vertex q_0. After that, it collects and merges the weighted path $\mathsf{spath}_{C_k}(q_0,q^*)$ into T_Q to construct the tree (line 9). Recursively, it treats q^* as the new vertex q_0 and starts the shortest path search algorithm from q_0, until all query nodes in Q are traversed once in this way (line 10). Finally, the algorithm returns the tree T_Q connecting all query nodes Q (line 11).

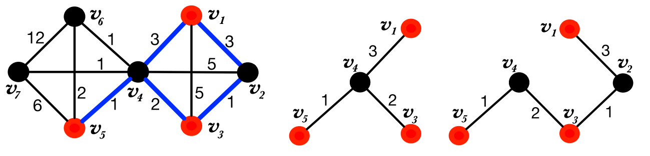


Fig. 3. An example of tree generation for query nodes v_1, v_3 and v_5 in graph G. Tree T_A is the spanning tree generated by Algorithm 3; tree T_B is the path-based tree generated by Algorithm 4.

A Comparison of Tree-Based and Path-Based Generation Methods. The tree-based generation in Algorithm 3 and path-based generation in Algorithm 4 may generate different results, i.e., different tree structures to connect all query vertices Q. Let us consider an example graph G in Fig. 3a. The weighted graph G has 7 vertices and 12 weighted edges. Assume that the query vertices are $Q=\{v_1,v_3,v_5\}$ in red in Fig. 3a and $k=3$. The whole graph G is a 3-core. We apply Algorithms 3 and 4 on G respectively. Algorithm 3 first finds the shortest path between every pair of query vertices in Q. We depict the edges along all such shortest paths in blue in Fig. 3a. For example, the shortest path between v_1 and v_3 is $\mathsf{spath}(v_1,v_3)=\{(v_1,v_2),(v_2,v_3)\}$. Similarly, $\mathsf{spath}(v_1,v_5)=\{(v_1,v_4),(v_4,v_5)\}$, $\mathsf{spath}(v_3,v_5)=\{(v_3,v_4),(v_4,v_5)\}$. Next, the three paths in blue are merged together to produce a weighted graph G_{pw} in Fig. 3a. Finally, the tree-based generation in Algorithm 3 constructs a spanning tree of T_A shown in Fig. 3b, which connects all query vertices $\{v_1,v_3,v_4,v_5\}$ with a group weight of 6. On the other hand, we apply the path-based generation in Algorithm 4 on G for the same query Q. First, Algorithm 4 randomly selects one vertex v_5 for the shortest path search. The nearest query node to v_5 is v_3. It finds the shortest path $\mathsf{spath}(v_5,v_3)=\{(v_5,v_4),(v_4,v_3)\}$. Next, it starts from v_3 in turns and finds the shortest path $\mathsf{spath}(v_3,v_1)=\{(v_3,v_2),(v_2,v_1)\}$. Finally, we merge the two paths to construct the tree T_B shown in Fig. 3c, which has a group weight of 7 different from the tree T_A in Fig. 3b. In summary, our tree-based and path-based algorithms may generate two different trees to connect Q using different strategies.

Complexity Analysis. We analyze the complexity of Algorithms 3 and 4. Assume that the k-core C_k has n_k nodes and m_k edges where $n_k \leq n$ and $m_k \leq m$.

For Algorithm 3, an intuitive implementation of all-pairs-shortest-paths needs to compute the shortest path for every pair nodes in Q, which takes $O(|Q|^2 m_k \log n_k)$ time. However, a fast implementation of single-source-shortest-path algorithm can compute the shortest path from one query node $q \in Q$ to all other nodes in Q, which takes $O(m_k \log n_k)$ time. Overall, the computation of all-pairs-shortest-paths can be done in $O(|Q| m_k \log n_k)$ time. In addition, the weighted graph G_{pw} is a subgraph of C_k, thus the size of G_{pw} is $O(n_k+m_k) \subseteq O(m_k)$. Identifying the spanning tree of G_{pw} takes $O(m_k \log n_k)$ time. Overall, Algorithm 3 takes $O(|Q| m_k \log n_k)$ time and $O(m_k)$ space.

For Algorithm 4, it applies $|Q|$ times of single-source-shortest-path to identify the nearest query node. Thus, Algorithm 4 also takes $O(|Q|m_k \log n_k)$ time and $O(m_k)$ space. In practice, Algorithm 4 runs faster than Algorithm 3 on large real-world graphs, which avoids the weighted tree construction and all-pairs-shortest-paths detection.

4.4 Tree-to-Graph Expansion

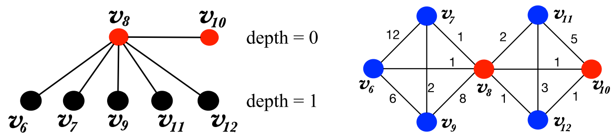
In this section, we introduce the phase of tree-to-graph expansion. This method expands the obtained tree from Algorithms 3 or 4 into a connected k-core candidate subgraph G_Q. It consists of two main steps. First, it adds nodes/edges to expand the tree into a graph layer by layer. Then, it prunes disqualified nodes/edges to maintain the remaining graph as a connected k-core. The whole procedure is shown in Algorithm 5.

Algorithm 5. Tree-to-Graph Expansion

Input: $G = (V, E, w)$, a set of query vertices Q, k-core index, T_Q
Output: Candidate subgraph G_Q
1: Identify the maximal connected k-core of C_k containing query nodes Q;
2: $L_0 \leftarrow \{v | v \in V_{T_Q}\}$; $L' \leftarrow L_0$;
3: $i \leftarrow 0$; $G_Q \leftarrow \emptyset$;
4: **while** $G_Q = \emptyset$ **do**
5: **for** each $v \in L_i$ **do**
6: **for** each $u \in N_{C_k}(v)$ and $u \notin L' \cup L_{i+1}$ **do**
7: $L_{i+1} \leftarrow L_{i+1} \cup \{u\}$;
8: $L' \leftarrow L' \cup L_{i+1}$; $i \leftarrow i + 1$;
9: Let G_L be the induced subgraph of G by the node set L';
10: Generate a connected k-core of G_L containing query nodes Q as G_Q;
11: **return** G_Q;

Algorithm 5 first gets all nodes in T_Q and puts them into L_0 (line 2). Let L_i be the vertex set at the ith depth of expansion tree, and L_0 be the initial set of vertices. It uses L' to represent the set of candidate vertices, which is the union of all L_i set. The iterative procedure can be divided into three steps (lines 4-10). First, for each vertex v in L_i, it adds their neighbors into L_{i+1} (lines 5-7). Next, it collects and merges $\{L_0, \ldots, L_{i+1}\}$ into L' and constructs a candidate graph G_L as the induced subgraph of G by the node set L' (lines 8-9). Finally, we apply the core decomposition algorithm on G_L to find the connected k-core subgraph containing all query nodes, denoted as G_Q. If there exists no such G_Q, Algorithm 5 explores the $(i+1)$th depth of expansion tree and repeats the above procedure (lines 4-10). In the worst case, G_Q is exactly the maximum connected k-core subgraph containing Q. However, G_Q in practice is always much smaller than it. The time complexity for expansion is $O(\sum_{i=0}^{l_{max}} \sum_{v \in V(G_i)} deg(v))$, where l_{max} is the iteration number of expansion in Algorithm 5.

Example 3. Fig. 1a shows a weighted graph G with query $Q = \{v_8, v_{10}\}$ and $k = 3$. We first identify the maximal connected 3-core containing query nodes Q. Since there is only 2 query nodes, the spanning tree is same as the shortest path between them, such that $T_Q = \mathsf{spath}_{C_3}(v_8, v_{10})$. Next, we initialize L_0 as $L_0 = \{v_8, v_{10}\}$ and expand nodes in L_0 to their neighbors. The expansion procedure is shown in Fig. 4a. We put all nodes in Fig. 4a into L' and construct a candidate subgraph G_L shown in Fig. 4b. Since G_L is a 3-core connected subgraph containing query nodes, the expansion graph G_Q is G_L itself.



(a) Expansion (b) Candidate subgraph construction

Fig. 4. Tree-to-graph expansion.

4.5 Intimate-Core Refinement

This phase refines the candidate connected k-core into an answer of the intimate-core group. We apply the existing approach ICG-M [9] by removing nodes to shrink the candidate graph obtained from Algorithm 5. This step takes $O(m' \log_{\varepsilon} n')$ time, where $\varepsilon > 0$ is a parameter of shrinking graph [9]. To avoid query nodes deleted by the removal processes of ICG-M, we develop a mechanism to protect important query nodes.

Protection Mechanism for Query Nodes. As pointed by [34], [35], [36], the k-core structure may collapse when critical nodes are removed. Thus, we precompute such critical nodes for query nodes in k-core and ensure that they are not deleted in any situations. We use an example to illustrate our ideas. For a query node q with an exact degree of k, it means that if any neighbor is deleted, there exists no feasible k-core containing q any more. Thus, q and all q's neighbors are needed to protect. For example, in Fig. 4b, assume that $k = 3$, there exists $deg_G(v_{10}) = k$. The removal of each node in $N_G(v_{10})$ will cause core decomposition and the deletion of v_{10}. This protection mechanism for query nodes can also be used for k-core maintenance in the phrase of tree-to-graph expansion.

5 WC-Index-Based Querying Algorithms

In this section, we propose to keep the node corenesses and edge weights into an index, called the Weighted-Core index (WC-index). We present the data structure of WC-index and an index construction method in Section 5.1. Leveraging on WC-index and our framework of LEKS, we develop two new algorithms of tree-to-graph expansion and intimate-core refinement respectively in Sections 5.2 and 5.3.

Overview. In weighted graphs, various weights of edges reflect different strengths of an intimate relationship between two nodes. However, the previous k-core index in Section 4.1 keeps no records of this importantly useful information of edge weights. To make use of edge weights, we propose a new WC-index to keep the node corenesses and edge weights in an integrated way. With the help of WC-index, we propose new search strategies to improve the LEKS framework in two ways: (1) we get a high-quality candidate graph with a few vertices and a smaller group weight; (2) we develop an intimate-core refinement strategy to achieve a good trade-off between efficiency and effectiveness. Equipped with these two new algorithms, our framework LEKS is able to find intimate-core groups with much smaller weights using less computational cost, which scales well with large graphs in practice.

TABLE 3
WC-index of Graph G in Fig. 1a

Node	$\delta(v)$	Neighbor-Weight Set
v_1	3	$(v_2,1);(v_4,3);(v_3,5)$
v_2	3	$(v_1,1);(v_3,1);(v_4,2)$
v_3	3	$(v_2,1);(v_4,3);(v_1,5)$
v_4	3	$(v_2,2);(v_1,3);(v_3,3);(v_5,4)$
v_5	2	$(v_4,4);(v_6,8)$
v_6	3	$(v_8,1);(v_9,6);(v_5,8);(v_7,12)$
v_7	3	$(v_8,1);(v_9,2);(v_6,12)$
v_8	3	$(v_6,1);(v_7,1);(v_{10},1);(v_{12},1);(v_{11},2);(v_9,8)$
v_9	3	$(v_7,2);(v_6,6);(v_8,8)$
v_{10}	3	$(v_8,1);(v_{12},1);(v_{11},5)$
v_{11}	3	$(v_8,2);(v_{12},3);(v_{10},5)$
v_{12}	3	$(v_8,1);(v_{10},1);(v_{11},3)$

5.1 Weighted-Core Index

A simple k-core index that keeps the coreness of all nodes, is proposed in Section 4.1. However, this index does not make use of importantly useful information of edge weights. Given the problem objective of finding an intimate-core group with a small weight, it motivates to design an effective index integrating the node corenesses and edge weights together as follows.

WC-Index Data Structure. We construct the WC-index for each vertex in graph G. Given a vertex $v \in V$, the WC-index of v consists of two components: the coreness $\delta(v)$ and a neighbor-weight set denoted by $NW(v) = \{(u, w(u,v)) : u \in N(v)\}$. The coreness $\delta(v)$ indicates the maximum value of k such that there exists a k-core containing v in G. The neighbor-weight set of $NW(v)$ is a list of sorted neighbors $u \in N(v)$ and their corresponding edge weights of $w(v,u)$. The neighbors in $NW(v)$ are sorted in the increasing order of their edge weights. For example, Table 3 shows the data structure of WC-index in graph G in Fig. 1a.

WC-Index Construction. We first apply Algorithm 1 on graph G to compute the coreness of all nodes in V. For each vertex $v \in V$, we simply sort its vertex neighbors in the ascending order of their corresponding edge weights. Finally, we construct the WC-index structure, consists of triple elements of $[v, \delta(v), NW(v)]$ for all vertices $v \in V$. The rational of sorting neighbors in terms of edge weights aims at the efficient retrieval of good candidate vertices with small weights. This idea is intuitive but shown to be very useful in developing two newly effective expansion and refinement algorithms for finding high-quality answers in Sections 5.2 and 5.3.

5.2 WC-Index-Based Expansion

Based on WC-index, we propose a tree-to-graph expansion called WC-index-based expansion, which uses two expansion strategies of *k-breadth expansion* and *1-depth expansion*.

Motivation. Algorithm 5 implements a tree-to-graph expansion in LEKS, which expands a tree T_Q to a candidate graph G_Q. However, the expansion procedure has two major drawbacks. First, Algorithm 5 expands from nodes in T_Q by adding all their neighbors. It may add too many new vertices into G_Q, especially when the expanding nodes have high degrees. Moreover, the newly added edges may have large edge weights, leading to a bad candidate graph G_Q with a large number of nodes and a large group weight. To refine this candidate graph G_Q, it requires a large number of removal iterations in the refinement phase, which incurs inefficiency. Second, Algorithm 5 iteratively expands level by level until it generates a connected k-core in a BFS manner. To overcome these drawbacks, we use k-breadth expansion to add k neighbors with the smallest edge weights, but not all neighbors. Beside this BFS expansion, we design 1-depth expansion to fast get a connected k-core in a DFS manner. Overall, the purpose of WC-index-based expansion is to find a candidate subgraph with a small weight quickly.

A Two-Level Expansion of k-Breadth and 1-Depth. WC-index-based expansion consists of two expansion strategies: *k-breadth expansion* and *1-depth expansion*. First, the k-breadth expansion adds into G_Q with k neighbors with the smallest edge weights, but not all neighbors. Such k neighbors can be obtained by an efficient retrieval of WC-index. In practice, the expanded G_Q is difficult to form a connected k-core directly. For leveraging on the k-breadth expansion only is not enough, we design another strategy of 1-depth expansion. The 1-depth expansion starts from one neighbor node X with the smallest weight and further expands one more depth to another neighbor of X with the smallest edge weight. Integrating both strategies, our method of WC-index-based expansion is able to find a candidate subgraph with a small weight quickly.

Algorithm 6. WC-index-Based Expansion

Input: WC-index, an integer k, a set of query vertices Q, the spanning tree T_Q
Output: Candidate subgraph G_Q

1: **for** vertex $v \in V_{T_Q}$ **do**
2:   Extract a set of sorted neighbors $N_{C_k}(v) = \{u \in N(v) : \delta(u) \geq k\}$ from WC-index;
3: $L_0 \leftarrow \{v | v \in V_{T_Q}\}$; $L' \leftarrow L_0$;
4: $i \leftarrow 0$; $D_i \leftarrow \emptyset$; $G_i \leftarrow \emptyset$; $G_Q \leftarrow \emptyset$;
5: **while** $G_Q = \emptyset$ **do**
6:   **for** each $v \in L_i$ **do**
7:     $count \leftarrow 0$; $L' \leftarrow L' \setminus D_i$;
8:     **for** each $u \in N_{C_k}(v)$ and $u \notin L'$ **do**
9:       $count \leftarrow count + 1$;
10:       **if** $count > k$ **then**
11:         **goto** Step 6;
12:       **if** $u \notin L_{i+1}$ **then**
13:         $L_{i+1} \leftarrow L_{i+1} \cup \{u\}$;
14:         **if** $count = 1$ **then**
15:           Let u' be the first item in $N_{C_k}(u) \setminus (L' \cup L_{i+1})$;
16:           $D_{i+1} \leftarrow D_{i+1} \cup \{u'\}$;
17:   $L' \leftarrow L' \cup L_{i+1} \cup D_{i+1}$; $i \leftarrow i + 1$;
18:   Let G_i be the induced subgraph of G by the node set L';
19:   Generate a connected k-core of G_i containing query nodes Q as G_Q;
20: **return** G_Q;

Algorithm. The whole procedure of WC-index-based expansion is shown in Algorithm 6. Based on the WC-index, the algorithm first extracts a set of sorted neighbors $N_{C_k}(v)$ with the coreness of k for all vertices in T_Q (lines 1-2). It iterative expands T_Q to a connected k-core G_Q using three steps: *k-breadth expansion*, *1-depth expansion*, and *candidate validation* (lines 3-20). At the ith iteration, let L_i be the node set for

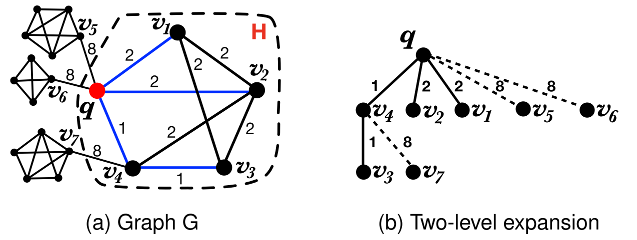


Fig. 5. An example of WC-index-based expansion.

k-breadth expansion, and D_i be the node set for 1-depth expansion. First, the k-breadth expansion adds the k neighbors with smallest weights into L_{i+1} (lines 5-13). Second, the 1-depth expansion adds only one neighbor with the smallest weights into D_{i+1} (lines 14-16). Third, it validates the qualification of the expanded graph. The algorithm collects all candidate nodes of L', which is formed by all nodes of L_i and D_i (line 17). Let G_i represents a candidate induced subgraph formed by L'. The algorithm stops when it finds a qualified connected k-core G_Q of G_i (lines 18-19).

Example 4. We apply WC-index-based expansion in Algorithm 6 on graph G in Fig. 5a for $Q = \{q\}$ and $k = 3$. Algorithm 6 expands from query vertex q using two strategies of *3-breadth expansion* and *1-depth expansion*. It obtains the sorted neighbors of q in connected 3-core from the WC-index, i.e., $NW(q) = \{(v_4, 1); (v_2, 2); (v_1, 2); (v_5, 8); (v_6, 8)\}$. First, the 3-breadth expansion starts from q to add nodes v_4, v_2, and v_1 into the candidate graph, as shown in Fig. 5b. Second, the 1-depth strategy adds vertex v_3, which is the closest neighbor to v_4. Finally, the inducted subgraph H of G formed by node $L' = \{q, v_1, v_2, v_3, v_4\}$ is a connected 3-core containing q shown in Fig. 5a. It only takes two simple steps to generate a small connected 3-core H efficiently. However, if we do not use this two-level expansion but adopt the full BFS expansion, a bad result will be generated. First, it collects all q's neighbors $N(q) = \{v_1, v_2, v_4, v_5, v_6\}$. The induced subgraph of G by $N(q) \cup \{q\}$ is not a 3-core. Next, it expands one more hop and generates the candidate graph, which is larger than H generated by our WC-index-based expansion.

5.3 WC-Index-Based Refinement

After the WC-index-based expansion from a tree to a graph, we need to shrink the graph into an intimate-core group with a small weight. Beside the protection mechanism shown in the intimate-core refinement in Section 4.5, we propose a new method of WC-index-based refinement. The WC-index-based refinement consists of three parts: 1) a new removal order for node deletion based on the minimal weight; 2) coarse-grained binary deletion for large graphs; and 3) fine-grained careful refinement for small graphs.

Minimal Weight Removal. The ICG-M algorithm [9] performs the intimate-core refinement by deleting nodes with the largest node weights, where the aggregated node weight of v is defined as the sum of all incident edge weights. However, the aggregated node weight cannot reflect the importance of nodes, especially lots of edges incident to a node is not belong an intimate-core group. To address this limitation, we define a new definition of minimal weight as follows.

Definition 6 (Minimal Weight). *Given a graph H and a node v, the minimal weight of v is defined as the minimum weight of all edges incident to v in H, denoted by $\chi_H(v)$, i.e., $\chi_H(v) = \min_{u \in N_H(v)} w(u, v)$.*

The minimal weight of a node represents the strongest strength of connection between this node to the graph. If a node has a large minimal weight, all edges have large weights, indicating weakly intimate relationships between this node to the graph.

Coarse-Grained/Fine-Grained Deletion Strategies. Different from deleting a constant proportion of nodes by ICG-M [9] at each iteration, we propose two deletion strategies: coarse-grained deletion for large graphs and fine-grained deletion for small graphs. The new strategies have the significant advantages of efficiency (fast identifying small-sized intimate groups by coarse-grained deletion) and effectiveness (finding high-quality intimate groups by fine-grained deletion). For large graphs, we propose to use a binary deletion by removing a half of nodes from graph, which can quickly reduce the graph size and group weight. For small graphs, we propose to use a careful refinement by removing one node from graph each time. This fine-grained deletion avoids deleting important nodes that may lead the graph to be a disqualified answer, which ensures a good answer of intimate-core group with a small weight.

Based on the minimal weight removal and coarse-grained/fine-grained deletion strategies, we propose the WC-index-based refinement in Algorithm 7. It illustrates the procedure of refining the candidate graph G_Q to the answer H. It computes the protected vertices V_p in advance as shown in Section 4.5. The algorithm has two key steps: sorting nodes by their minimal weights and deletion of nodes with large weights. It first gets the sorted neighbor set $N(v)$ (line 1). The Tag equals to 'False', indicating that the algorithm uses the binary deletion; otherwise, it uses the careful deletion (line 2). Then, it computes the minimal weight $\chi_{G_Q}(v)$ for all nodes with the help of WC-index (lines 4-6). It sorts nodes in the descending order of minimal weights (line 7). Based on the graph size, the algorithm chooses to use binary deletion (lines 8-17) or careful deletion (lines 18-25). For a large graph with more than γ nodes, it applies the binary deletion, which deletes a half of nodes with the largest minimal weights (lines 8-15). After deletion, the algorithm maintains the remaining graph as a connected k-core containing Q (line 12). If the binary deletion is not applicable to current graph, we set the Tag as 'True' (lines 16-17). For a small graph with no greater than γ nodes, it deletes one node from graph at each iteration (lines 18-25).

Summary. We give a summary of the proposed techniques of WC-index, WC-index-based expansion, and WC-index-based refinement. Three new techniques play essentially important roles in our LEKS framework. WC-index integrates an advanced index of node corenesses and edge weights, which offers the efficient retrieval for two important phases of expansion and refinement in LEKS. In the tree-to-graph expansion phase, WC-index-based expansion can find a small candidate of intimate-core group using a two-level expansion of k-breadth and 1-depth. In the refinement phase, WC-index-based refinement removes nodes using a new metric of minimal weight and develops two deletion strategies of coarse-grained/fine-grained refinements for large/small graphs. Equipped with these new techniques, our LEKS framework can deal with intimate-group queries efficiently and find high-quality answers.

Algorithm 7. WC-index-Based Refinement

Input: WC-index, an integer k, a set of query vertices Q, candidate subgraph G_Q, protected vertices V_p
Output: The intimate core group H
1: Get the sorted neighbors $N(v)$ using WC-index;
2: $Tag \leftarrow$ False;
3: **while** there are nodes to be removed **do**
4: **for** each $v \in V_{G_Q} \setminus V_p$ **do**
5: Let u be the first node in $N_{G_Q}(v)$;
6: $\chi_{G_Q}(v) \leftarrow w(v,u)$;
7: Sort all nodes in $V_{G_Q} \setminus V_p$ in the descending order by $\chi_{G_Q}(v)$;
8: **if** $|V_{G_Q} \setminus V_p| > \gamma$ and Tag = False **then**
9: Copy the range $(0, |V_{G_Q} \setminus V_p|/2)$ of $V_{G_Q} \setminus V_p$ to V_{delete};
10: **for** each $v \in V_{delete}$ **do**
11: Remove v from G_Q;
12: Maintain G_Q as a connected k-core containing Q;
13: **if** $G_Q = \emptyset$ **then**
14: Restore the graph G_Q;
15: $V_p \leftarrow V_p \cup \{v\}$;
16: **if** the number of removed nodes in V_{delete} is less than $|V_{delete}|/2$ **then**
17: Tag $\leftarrow$ True;
18: **if** $|V_{G_Q} \setminus V_p| \leq \gamma$ or Tag $=$ True **then**
19: **for** each $v \in V_{G_Q} \setminus V_p$ **do**
20: Remove v from G_Q;
21: Maintain G_Q as a connected k-core containing Q;
22: **if** $G_Q = \emptyset$ **then**
23: Restore the graph G_Q;
24: $V_p \leftarrow V_p \cup \{v\}$;
25: **else goto** Step 3;
26: $H \leftarrow G_Q$;
27: **return** H;

6 EXPERIMENTS

In this section, we evaluate the performance of our proposed algorithms. All algorithms are implemented in Java.

Datasets. We use six real-world datasets in experiments. The wiki-vote and Flickr datasets are publicly available from [37]. The edge weight represents the existence probability of an edge. A smaller weight indicates a higher possibility of the edge to existing. The other four ground-truth datasets are from Stanford Large Network Dataset Collection.[1] We assign each edge within communities with a random edge weight in $(0, 0.2]$, and each edge outside communities with a random edge weight in $[0.3, 0.9]$. The statistics of all datasets are shown in Table 4. The maximum coreness $\delta_{max} = \max_{v \in V} \delta(v)$.

Algorithms. We compare four algorithms as follows.

- ICG-M: is the state-of-the-art approach for finding intimate-core group using bulk deletion [9].
- LEKS-tree: is our index-based search framework in Algorithm 2 using Algorithm 3 for tree generation.

TABLE 4
Network Statistics

Datasets	$\|V\|$	$\|E\|$	δ_{max}	Ground-Truth
wiki-vote	7,115	103,689	56	No
Flickr	24,125	300,836	225	No
com-Amazon	334,863	925,872	6	Yes
com-DBLP	317,080	1,049,866	113	Yes
com-Youtube	1,134,890	2,987,624	51	Yes
com-LiveJournal	3,997,962	34,681,189	360	Yes

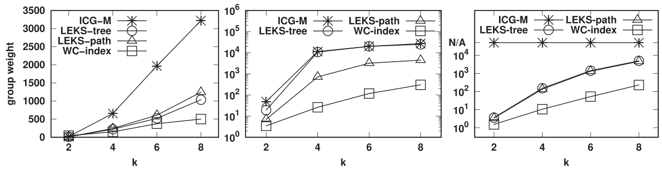
- LEKS-path: is our index-based search framework in Algorithm 2 using Algorithm 4 for tree generation.
- WC-index: is our WC-index-based search framework in Algorithm 2 using Algorithm 4 for tree generation, Algorithm 6 for expansion and Algorithm 7 for refinement.

We first evaluate all algorithms by comparing the running time and intimate-core group weight. The less running time costs, the more efficient the algorithm is. Smaller the group weight of the answer, better effectiveness is. To further evaluate the approaches, we apply them on datasets with ground-truth communities and discuss the precision, recall, and F1-score of the results. We also compare the size and construction time to analyze the k-core index and WC-index.

Queries and Parameters. For real-life datasets, we evaluate all competitive approaches by varying parameters k and $|Q|$. We randomly generate 100 sets of queries with different k and $|Q|$. We set $|Q| = 5$ and $k = 6$ by default. For ground-truth datasets, the queries are randomly generated from ground-truth communities. We set different k for different sized graphs with ground-truth. For three small graphs com-Amazon, com-DBLP, and com-Youtube, k is set as the smallest value of coreness of the query nodes. For a large graph com-LiveJournal and the generated dense graphs, we apply Algorithm 4 to generate spanning-tree T_Q and select the smallest coreness of the top ten nearest neighbors of nodes in T_Q as the value of k. Moreover, we set the parameter $\gamma = 100$ to decide coarse-grained/fine-grained deletions for WC-index by default. Note that γ calculates the number of vertices in the candidate graph excluding the query nodes and protected vertices. We treat the running time as infinite and the group weight as N/A if one algorithm cannot finish within 24 hours.

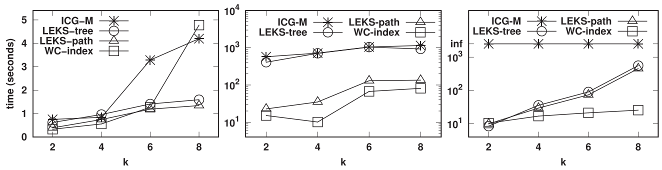
Exp-1: Varying k. Fig. 6 shows the group weight of four algorithms by varying parameter k on three datasets. We vary the number of query nodes k in {2, 4, 6, 8}. The results show that our WC-index method always gets a smallest group weight. The local search methods LEKS-tree and LEKS-path can find intimate groups with lower group weights than ICG-M, for different k. LEKS-path performs better than LEKS-tree on com-Youtube. LEKS-path and LEKS-tree achieve similar performances on Flickr and com-LiveJournal. Fig. 7 shows that LEKS-path has a good performance for most cases, and runs significantly faster than ICG-M. WC-index may take more time than other methods on small graphs since the fine-grained deletion strategy requires more iterations to get high-quality answers with smaller group weights. However, WC-index is clearly much more efficient than all other methods on large graphs due to its coarse-grained deletion strategy and two-level expansion strategy. As ICG-M takes more than

1. https://snap.stanford.edu/data/



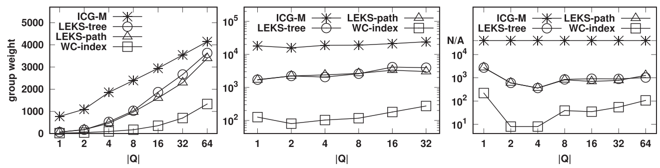
(a) Flickr (b) com-Youtube (c) com-LiveJournal

Fig. 6. Effectiveness evaluation of all methods by varying k.



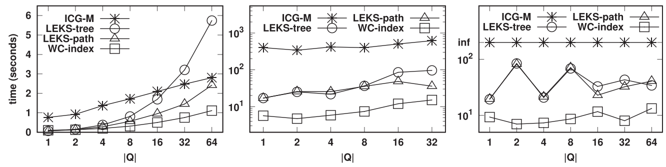
(a) Flickr (b) com-Youtube (c) com-LiveJournal

Fig. 7. Efficiency evaluation of all methods by varying k.



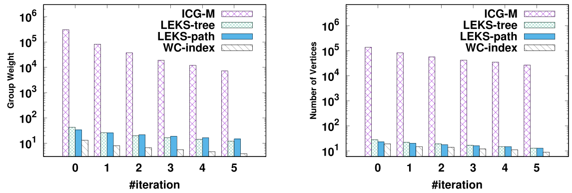
(a) Flickr (b) com-Youtube (c) com-LiveJournal

Fig. 8. Effectiveness evaluation of all methods by varying $|Q|$.



(a) Flickr (b) com-Youtube (c) com-LiveJournal

Fig. 9. Efficiency evaluation of all methods by varying $|Q|$.



(a) Group weight (b) Number of vertices

Fig. 10. The size and weight of intimate-groups on com-DBLP varied by #iterations.

24 hours for one query on com-LiveJournal, we denote its running time as infinity.

Exp-2: Varying $|Q|$. We evaluate the efficiency and quality performance of all algorithms for different queries by varying $|Q|$. We vary the number of query nodes $|Q|$ in {1, 2, 4, 8, 16, 32, 64}. Fig. 8 reports the group weight results. With the increased $|Q|$, WC-index can always achieve smaller group weights than other methods, especially on large graphs com-Youtube and com-LiveJournal as shown in Figs. 8b and 8c. LEKS-tree and LEKS-path methods have similar performances, which are better than ICG-M. Fig. 9 shows the results of running time by varying $|Q|$. WC-index always outperforms other methods using a smaller running time for different $|Q|$ in most cases. Moreover, the efficiency performance of WC-index is slightly stable. With the increased Q, all methods generally cost more running time on Flickr in Fig. 9a. ICG-M cannot finish the query processing tasks within 24 hours on the large graph com-LiveJournal in Fig. 9c. WC-index may take extra efforts for fine-grained node deletion in small graphs and run fast on large graphs by two-level expansion strategy and coarse-grained deletion. Note that we test $|Q|$ from 1 to 32 on com-Youtube as there are no ground-truth communities with enough vertices.

Exp-3: Quality Evaluation of Candidate Intimate-Core Groups. This experiment evaluates the subgraphs of candidate intimate-core groups by all methods, in terms of vertex size and group weight. ICG-M takes the maximal connected k-core subgraph containing query nodes as an initial candidate, and iteratively shrinks it. LEKS-tree and LEKS-path both generate an initial candidate subgraph locally expanded from a tree, and then iteratively shrink the candidate by ICG-M. WC-index method uses the WC-index-based two-level expansion strategy to construct the candidate subgraphs and then refines the graph depend on minimal weight. We report the results of the first 5 removal iterations and the initial candidate at the #iteration of 0. Fig. 10a shows that the group weight of candidates by our methods is much smaller than ICG-M. Fig. 10b reports the vertex size of all candidates at each iteration. The number of vertices in the candidate group by our methods at the #iteration of 0, is even less than the vertex size of candidate group by ICG-M at the #iteration of 5. The WC-index can always achieve good candidate graphs with the smallest group weights.

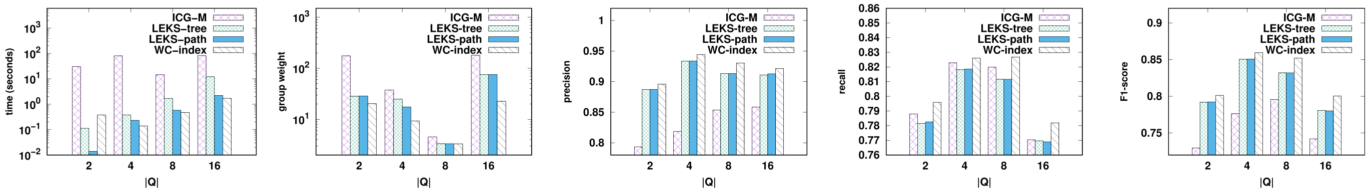
Exp-4: Running Time and Quality Evaluation on Ground-Truth Datasets. We compare the four algorithms on four large graphs with ground-truth communities by setting default $|Q| = 8$. As shown in Table 5, ICG-M algorithm always gets the worst results both on running time and quality. ICG-M cannot finish within 24 hours on large graph com-LiveJournal. LEKS-tree achieves similar F1-scores to LEKS-path but takes longer time. The WC-index method achieves the best performance especially for large graphs in terms of efficiency and effectiveness.

Exp-5: Evaluation of All Methods on Com-DBLP by Varying $|Q|$. Figs. 11a and 11b respectively show the running time and group weight comparison on all algorithms by varying $|Q|$ in {2, 4, 8, 16}. Our methods always have a better performance than ICG-M. WC-index takes less time than other methods in most cases but takes more time when $k = 2$ since it needs more cost to obtain high-quality answers. WC-index can always get answers with the smallest group weights for different k. Figs. 11c, 11d, and 11e report the quality evaluation by comparing the precision, recall, and F1-score. Our methods always have higher precision and F1-score than ICG-M. The WC-index method has the largest value on all these three matrices than others.

Exp-6: Index Construction. We evaluate the construction of the simple k-core index and the WC-index by comparing the index size and generation time in Table 6. Here, we use *k-core* to indicate the k-core index, which only keeps the coreness of all vertices. The indexes are built off-line and stored in memory. We can see that the size of WC-index is about 1.4 times of the original graph size $|G|$, which is compact and very competitive. This confirms that WC-index has $O(m)$ space complexity. On the other hand, the construction

TABLE 5
Running Time and Quality Evaluation on Ground-Truth Datasets

Dataset	Algorithm	Precision	Recall	F1-score	Running time	Group Weight
com-Amazon	ICG-M	0.945	0.718	0.782	75.544	4.36
	LEKS-tree	0.953	0.718	0.788	0.052	4.16
	LEKS-path	0.953	0.719	0.789	0.050	4.16
	WC-index	0.954	0.716	0.789	0.049	4.17
com-DBLP	ICG-M	0.853	0.819	0.796	144.038	4.52
	LEKS-tree	0.913	0.812	0.832	1.693	3.33
	LEKS-path	0.913	0.812	0.832	0.581	3.33
	WC-index	0.930	0.827	0.852	0.472	3.3
com-Youtube	ICG-M	0.451	0.564	0.339	1414.724	6510.34
	LEKS-tree	0.515	0.565	0.412	131.174	1933.76
	LEKS-path	0.527	0.567	0.423	115.599	2670.66
	WC-index	0.658	0.571	0.510	40.587	587.02
com-LiveJournal	ICG-M	–	–	–	–	–
	LEKS-tree	0.822	0.614	0.634	216.709	2455.67
	LEKS-path	0.821	0.616	0.634	110.641	2354.19
	WC-index	0.873	0.620	0.658	12.524	1513.66



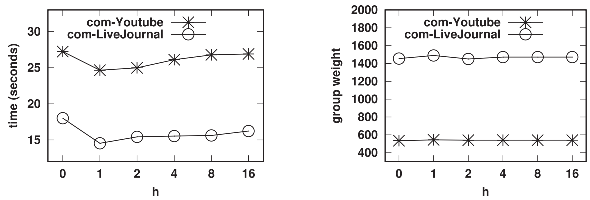
(a) Running time (b) Group weight (c) Precision (d) Recall (e) F1-score

Fig. 11. Evaluation of all methods on com-DBLP by varying $|Q|$.

time of WC-index takes a little bit longer time than the construction time of k-index, which is is about 1.3 times of k-core index construction. Therefore, as shown in Figs. 7, 9 and Table 5, WC-index-based approach is several orders of magnitude faster than ICG-M without using any index and also k-core index-based LEKS-tree and LEKS-path, especially on large graphs.

Exp-7: Varying h for the h-depth Expansion. To validate the effectiveness of 1-depth expansion strategy used in WC-index, we evaluate WC-index using h-depth expansion, which explores h vertices with the smallest edge weights at each level of h-depth from queries' neighbors. We vary the parameter h from 0 to 16. For $h=0$, WC-index does not invoke any depth expansion and only uses the k-breadth expansion. Fig. 12 reports the results of running time and group weight by varying the h-depth of WC-index on two ground-truth datasets com-Youtube and com-LiveJournal. As we can see, WC-index uses no depth expansion for $h=0$ takes more time than using the h-depth expansion for $h \geq 1$. This verifies that the depth-based expansion can identify a connected k-core easily using less iterative exploration of vertices, which saves much time in practice. For $h \geq 1$, the running time increases a little bit with increasing h. On the other hand, the group weight of all results keeps stable for different depth expansions as they use the same refinement strategy. As a result, we use the 1-depth expansion in the implementation of WC-index.

Exp-8: Varying γ in WC-index *Refinement Phase.* In this experiment, we evaluate the performance of WC-index by varying γ on large graphs com-Youtube and com-LiveJournal. We vary γ from 0 to 400. The results are reported in Fig. 13. When $\gamma = 0$, WC-index achieves a larger group weight in Fig. 13b. This because that WC-index only uses the coarse-grained deletion for $\gamma = 0$, which verifies the

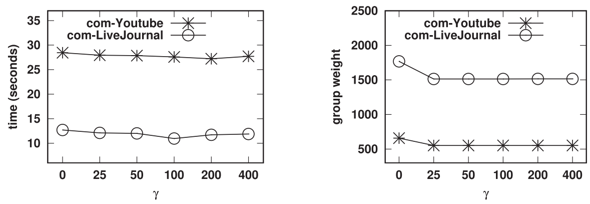


(a) Running Time (b) Group Weight

Fig. 12. Evaluating h-depth expansion by varying h.

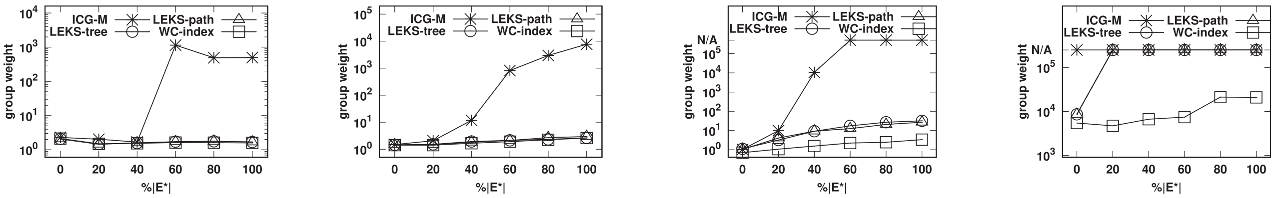
TABLE 6
Index Size (in Megabytes) and Index Construction Time (in Seconds)

Datasets	Graph Size	Index Size		Index Time	
		k-core	WC-Index	k-core	WC-Index
wiki-vote	2.05**MB**	0.06**MB**	3.97**MB**	0.31	0.43
Flickr	5.57**MB**	0.19**MB**	8.43**MB**	0.77	1.1
com-Amazon	16.7**MB**	2.88 **MB**	24**MB**	3.25	3.95
com-DBLP	18.6**MB**	2.73**MB**	26.4**MB**	3.53	4.52
com-Youtube	52.1**MB**	10.1**MB**	76.4**MB**	14.8	18.39
com-LiveJournal	639**MB**	38.3**MB**	839**MB**	24.58	27.27



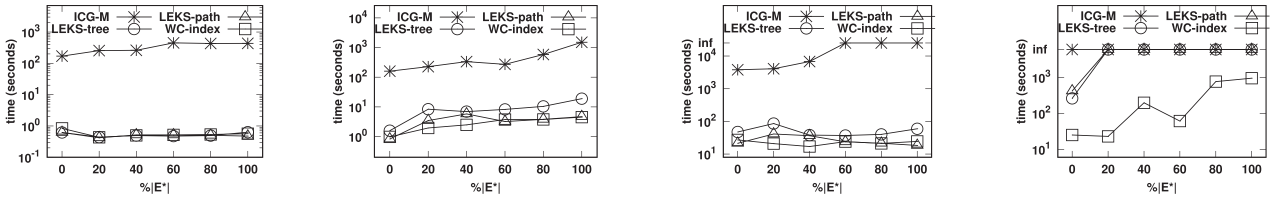
(a) Running Time (b) Group Weight

Fig. 13. Varying γ on com-Youtube and com-LiveJournal.



(a) com-Amazon (b) com-DBLP (c) com-Youtube (d) com-LiveJournal

Fig. 14. Effectiveness evaluation on ground-truth graphs for different graph density. Here, $\%|E^*|$ is the percentage of new adding edges.



(a) com-Amazon (b) com-DBLP (c) com-Youtube (d) com-LiveJournal

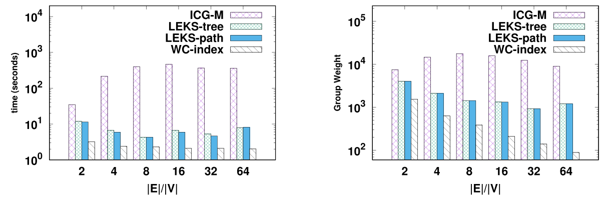
Fig. 15. Efficiency evaluation on ground-truth graphs for different graph density. Here, $\%|E^*|$ is the percentage of new adding edges.

effectiveness of fine-grained deletion strategy. When γ increases from 25 to 400, both the corresponding results of running time and group weight remain stable. This indicates that our default choice of $\gamma = 100$ is suitable on these real datasets.

Exp-9: Scalability Test on Dense Graphs With Ground-Truth Communities. To evaluate the efficiency and quality robustness, we conduct the scalability test of all algorithms on graphs with different densities. We use four graphs with ground-truth communities and increase the density by adding a percentage of new edges. We add $0\% \sim 100\%$ percentage new edges w.r.t. the original graph size $|E|$ into graphs, where the new edges are proportionally distributed over the inside-community and outside-community. Figs. 14 and 15 report the results of group weight and running time, respectively. It shows that our WC-index algorithm has better performance than others as the graph density increases. LEKS-path and LEKS-tree can achieve competitive efficiency results to WC-index in Figs. 15a, 15b, and 15c but they cannot finish the query task within 24 hours on the large graph comp-LiveJournal in Fig. 15d.

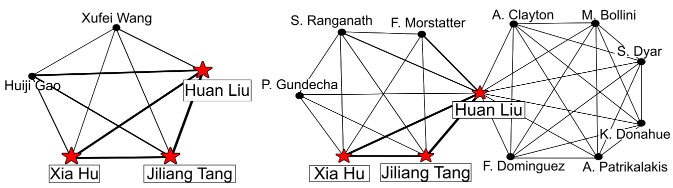
Exp-10: Scalability Test on Power-Law Graphs by Varying Graph Densities. We also conduct the scalability test of all algorithms on pow-law graphs with different densities. We randomly generate a series of power-law graphs, which have a graph density $\frac{|E|}{|V|}$ from 2 to 32. Each graph has 100,000 vertices. For queries, we set the parameter $k = 4$ and $|Q| = 1$. Fig. 16 reports the running time and group weight results of four methods. Our approach WC-index consistently achieves the smallest running time and group weight among all methods, indicating that WC-index can well handle different densities of graphs in a robust way. Fig. 16b shows one interesting phenomenon that WC-index achieves smaller group weights with the increased density, which may be caused by the existence of tightly dense communities.

Exp-11: Case Study on the DBLP Network. We conduct a case study of intimate-core group search on the collaboration DBLP network [9]. Each node represents an author, and an edge is added between two authors if they have co-authored papers. The weight of an edge (u, v) is the reciprocal of the number of papers they have co-authored. The smaller weight of (u, v), the closer intimacy between authors u and v. We use the query $Q =$ {"Huan Liu", "Xia Hu", "Jiliang Tang"} and $k = 4$. We apply LEKS-path and ICG-M to find 4-core intimate groups for Q. The results of LEKS-path and ICG-M are shown in Figs. 17a and 17b respectively. The bolder lines of an edge represent a smaller weight, indicating closer intimate relationships. Our LEKS method discovers a compact 4-core with 5 nodes and 10 edges in Fig. 17a, which has the group weight of 1.6, while ICG-M finds a subgraph with 12 nodes, which has a larger group weight of 16.7 in Fig. 17b. We can see that nodes on the right side of Fig. 17b has no co-author connections with two query nodes "Xia Hu" and "Jiliang Tang" at all. This case study verifies that our LEKS-path can successfully find a better intimate-core group than ICG-M.



(a) Running Time (b) Group Weight

Fig. 16. Evaluation on power-law graphs for different graph density.



(a) LEKS-path (b) ICG-M

Fig. 17. Case study of intimate-core group search on the DBLP network. Here, query $Q =$ {"Huan Liu", "Xia Hu", "Jiliang Tang"} and $k = 4$.

7 Conclusion

This paper presents a local exploration k-core search (LEKS) framework for efficient intimate-core group search. LEKS generates a spanning tree to connect query nodes in a compact structure, and locally expands it for intimate-core group refinement. Moreover, we design a WC-index, which keeps the node corenesses and edge weights. Based on WC-index, we propose two WC-index-based algorithms of expansion and refinement in LEKS, which accelerates the search efficiency of intimate-core group search over large graphs. Extensive experiments on real-world weighted graphs show that our approaches achieves a higher quality of answers using less running time, in comparison with state-of-the-art methods.

Acknowledgments

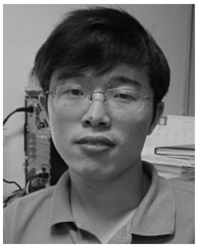
This paper was supported by NSFC 61702435, 62072034, 61772346, HK RGC Grants Nos. 22200320, 12201518, 12200817, 12232716, HK RGC CRF C6030-18G, and Guangdong Basic and Applied Basic Research Foundation (Project No. 2019B1515130001).

References

[1] M. E. Newman, "Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality," *Phys. Rev. E*, vol. 64, no. 1, 2001, Art. no. 016132.
[2] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Soc. Netw.*, vol. 32, no. 3, pp. 245–251, 2010.
[3] M. E. Newman, "Analysis of weighted networks," *Phys. Rev. E*, vol. 70, no. 5, 2004, Art. no. 056131.
[4] X. Huang, L. V. Lakshmanan, and J. Xu, *Community Search over Big Graphs*. San Rafael, CA, USA: Morgan & Claypool, 2019.
[5] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, "Index-based densest clique percolation community search in networks," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 2161–2162.
[6] Y. Fang *et al.*, "A survey of community search over big graphs," in *Proc. IEEE Symp. Very Large-Scale Data Anal. Vis. J.*, vol. 29, no. 1, 2020, pp. 353–392.
[7] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1311–1322.
[8] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2010, pp. 939–948.
[9] D. Zheng, J. Liu, R.-H. Li, C. Aslay, Y.-C. Chen, and X. Huang, "Querying intimate-core groups in weighted graphs," in *Proc. IEEE Int. Conf. Semantic Comput.*, 2017, pp. 156–163.
[10] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," 2003, *arXiv cs/0310049*.
[11] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek, "Streaming algorithms for k-core decomposition," *Proc. VLDB Endowment*, vol. 6, no. 6, pp. 433–444, 2013.
[12] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 812–823, 2012.
[13] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1233–1244, 2016.
[14] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proc. VLDB Endowment*, vol. 8, no. 5, pp. 509–520, 2015.
[15] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *Data Mining Knowl. Discov.*, vol. 29, no. 5, pp. 1406–1433, 2015.
[16] W. Zhu, C. Chen, X. Wang, and X. Lin, "K-core minimization: An edge manipulation approach," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, 2018, pp. 1667–1670.
[17] S. Medya, T. Ma, A. Silva, and A. Singh, "A game theoretic approach for k-core minimization," 2019, in *Proc. 19th Int. Conf. Auton. Agents MultiAgent Syst.*, 2020, pp. 1922–1924.
[18] L. Sun, X. Huang, R.-H. Li, and J. Xu, "Fast algorithms for intimate-core group search in weighted graphs," in *International Conference on Web Information Systems Engineering*. Springer, 2019, pp. 728–744.
[19] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," *Int. J. Very Large Data Bases*, vol. 25, no. 2, pp. 171–196, 2016.
[20] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proc. VLDB Endowment*, vol. 9, no. 4, pp. 276–287, 2015.
[21] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 991–1002.
[22] X. Huang and L. V. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.
[23] F. Bi, L. Chang, X. Lin, and W. Zhang, "An optimal and progressive approach to online search of top-k influential communities," *Proc. VLDB Endowment*, vol. 11, no. 9, pp. 1056–1068, 2018.
[24] Z. Zheng, F. Ye, R.-H. Li, G. Ling, and T. Jin, "Finding weighted k-truss communities in large networks," *Inf. Sci.*, vol. 417, pp. 344–360, 2017.
[25] D. Duan, Y. Li, Y. Jin, and Z. Lu, "Community mining on dynamic weighted directed graphs," in *Proc. ACM Int. Workshop Complex Netw. Meet Inf. Knowl. Manage.*, 2009, pp. 11–18.
[26] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu, "Effective and efficient attributed community search," *Int. J. Very Large Data Bases*, vol. 26, no. 6, pp. 803–828, 2017.
[27] Y. Li, C. Sha, X. Huang, and Y. Zhang, "Community detection in attributed graphs: An embedding approach," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 338–345.
[28] Y. Zhang *et al.*, "Community detection in networks with node features," *Electron. J. Statist.*, vol. 10, no. 2, pp. 3153–3178, 2016.
[29] G. Zhang, D. Jin, J. Gao, P. Jiao, F. Fogelman-Soulié, and X. Huang, "Finding communities with hierarchical semantics by distinguishing general and specialized topics," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 3648–3654.
[30] B. Tripathi, S. Parthasarathy, H. Sinha, K. Raman, and B. Ravindran, "Adapting community detection algorithms for disease module identification in heterogeneous biological networks," *Front. Genet.*, vol. 10, 2019, Art. no. 164.
[31] U. Gargi, W. Lu, V. Mirrokni, and S. Yoon, "Large-scale community detection on youtube for topic discovery and exploration," in *Proc. 5th Int. AAAI Conf. Weblogs Soc. Media*, 2011, pp. 486–489.
[32] N. Ruchansky, F. Bonchi, D. García-Soriano, F. Gullo, and N. Kourtellis, "The minimum wiener connector problem," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1587–1602.
[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2009.
[34] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Finding critical users for social network engagement: The collapsed k-core problem," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 245–251.
[35] 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.
[36] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin, "OLAK: An efficient algorithm to prevent unraveling in social networks," *Proc. VLDB Endowment*, vol. 10, no. 6, pp. 649–660, 2017.
[37] X. Huang, W. Lu, and L. V. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 77–90.

Longxu Sun received the MS degree in computer science from Hong Kong Baptist University, Hong Kong, in 2019. She is currently working toward the PhD degree at Hong Kong Baptist University, Hong Kong. Her research interests include community search and data mining.

Xin Huang received the PhD degree from the Chinese University of Hong Kong (CUHK), Hong Kong, in 2014. He is currently an assistant professor at Hong Kong Baptist University, Hong Kong. His research interests mainly focus on graph data management and mining.



Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong, Hong Kong, in 2013. He is currently an associate professor with the Beijing Institute of Technology (BIT), Beijing, China. Before joining BIT, in 2018, he was an assistant professor at Shenzhen University, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.

Byron Choi received the bachelor of engineering degree in computer engineering from the Hong Kong University of Science and Technology (HKUST), Hong Kong, in 1999, and the MSE and PhD degrees in computer and information science from the University of Pennsylvania, Philadelphia, Pennsylvania, in 2002 and 2006, respectively. He is an associate professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong.

Jianliang Xu received the PhD degree from The Hong Kong University of Science and Technology, Hong Kong. He is currently a professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. His research interests include big data management, mobile computing, and data security and privacy. He has published more than150 technical papers in these areas. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering*.