A Flexible Framework for Query-oriented Interactive Community Search

Longxu Sun[1], Xin Huang[1], Jiannan Wang[2,3], Jianliang Xu[1]
[1]Hong Kong Baptist University
[2]Huawei Technologies, [3]Simon Fraser University
{cslxsun, xinhuang, xujl}@comp.hkbu.edu.hk, jnwang@sfu.ca

Abstract

Community search finds query-dependent communities over graphs, which has been investigated broadly. In this work, we focus on the task of returning only a single connected community containing all user input query vertices. Most existing studies in the literature only propose *a single and static model* based on *a particular subgraph* (e.g., k-core, k-truss, quasi-clique, and learning-based component). These fixed models are hard to find exact community answers on all datasets and fit with different underlying desires of users and queries. This implies that the *community search* task needs human-in-loop *interactions*, which allows users to give feedback and dynamically advise community refinement.

To tackle the above issues, we formulate and study the problem of *interactive community search*, which allows users to add/delete vertices for improving community answers in a few rounds of interactions. We first summarize dozens of existing community models and develop an integrated notation system $\mathbb{M}(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P})$ to describe them all. Then, we propose a flexible approach to interactive community search over graphs called GICS-framework. The successful principle of GICS-framework lies on three key components: *personalized adding/deleting recommendation*, *parameter auto-tuning*, and *fast partial refinement*. We develop efficient algorithms and successfully deploy three community models on our GICS-framework. We further analyze algorithm complexity of GICS-framework by illustrating one instance model in detail. Extensive experiments on ground-truth communities demonstrate that our interaction of GICS-framework improves F1-score accuracy by 22% against state-of-the-art competitors, and gives users real-time responses within one second.

PVLDB Reference Format:
Longxu Sun[1], Xin Huang[1], Jiannan Wang[2,3], Jianliang Xu[1]. A Flexible Framework for Query-oriented Interactive Community Search. PVLDB, 18(6): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SunLongxu/interactiveCommunity.

Proceedings of the VLDB Endowment, Vol. 18, No. 6 ISSN 2150-8097.
doi:XX.XX/XXX.XX

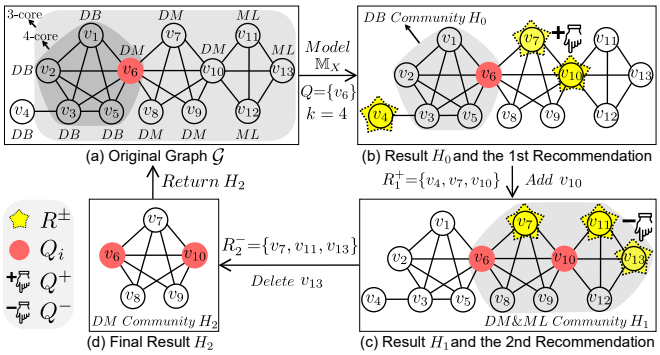


Figure 1: An example of interactive community search.

1 Introduction

Community search is a fundamental graph analytics task, which finds a closely connected subgraph containing a given set of queries in large graphs [1–3, 6, 8, 9, 17, 18, 20, 21, 24, 25, 27, 28, 31, 36, 37, 39, 40, 54, 55, 68, 70, 72, 73, 76]. Different from general community detection tasks [59, 66] which finds all communities in the graph without user input query vertex, community search supports query-oriented personalized search by locally exploring the graph to identify the community in a candidate subgraph quickly [24, 54]. It has several applications in real-world networks, such as social circles discovery in social networks [49], finding research groups in collaboration networks [46], and protein-protein structures identification in biology networks [11].

However, most existing community search studies focus only on the *static* problem setting, which returns community results in *one interactive round* after receiving user query inputs. The answer may be far more satisfactory from the users' perspective and cannot support further revision with more user inputs. We illustrate three reasons why one static community model cannot find good communities. First, a fixed model struggles to capture the diverse properties of all communities in real-life graphs, as no single model can accurately represent these varying connections. Second, several different parameters are difficult for users to configure. Query-oriented community search tasks always ask users to provide query vertices, attributes, and other parameters (e.g., density, size, diameter, etc.). Third, overlapping communities exist in real-life applications, leading to different communities depending on user input query vertices. Initially, users are uncertain about their desired communities, often realizing this only after several interactions. Motivated by these observations, this paper studies a different problem of *interactive community search*, which allows users to *continuously interact* with the community model to refine community results until they are satisfied.

Existing interactive community search ICS-GNN [23] considers user feedback to refine the community search result by labeling a positive vertex and a negative vertex in the candidate subgraph, which increases the size of the labeled vertices of the training set iteratively to improve the effectiveness of the results. Unfortunately, it still suffers three drawbacks. First, it uses a *fixed model* that does not adjust the size parameter k during the interaction phase. Second, it refines the answer by *re-running* search algorithms entirely, which is inefficient. Third, it lacks *recommendations* and *scalability* to other community models, thereby limiting its applicability and extension to other community models. These limitations highlight the importance of developing more flexible and efficient interactive community search solutions.

Challenges. To address these gaps, we intend to develop a novel interactive community search framework, which must overcome three key technical challenges as follows.

- *Challenge 1: the fitness of diverse community models.* The aim of a general interactive community search framework needs to fit many different community models. However, it is a significant challenge to fit multiple community search models with various features of dense subgraph patterns (e.g., k-core, k-truss, learning-based component, and so on), structural metrics, attributes metrics, and parameters.
- *Challenge 2: the design of user-friendly interactions.* Many choices of node insertion/deletion exit in a large-scale graph, which brings challenges for users to select in the interactive process. It seeks an effective recommendation schema by reducing such choices and auto-adjusting parameters to help users make fast and good community refinement.
- *Challenge 3: the real-time response of interactions.* The interactive community search seeks real-time responses. Thus, it is challenging to develop an efficient algorithm of community refinement, avoiding the computation from scratch.

To address the above challenges, we propose a novel GICS-framework of interactive community search. Leveraging a generalized notation system of community models, our GICS-framework equips with three useful components for fast and effective interactive actions, including *personalized recommendation*, *fast refinement*, and *parameter auto-adjustment*. A motivating example shown in Figure 1 introduces the key idea of our solution. Given a collaboration network $\mathcal{G}$ in Figure 1(a), a k-core community model with the smallest diameter, a query vertex $Q = \{v_6\}$, and parameter $k = 4$. The initial community H_0 in Figure 1(b) is a database community of 4-core containing v_6, which has the smallest diameter of 1. Then, our solution recommends three vertices v_4, v_7, and v_{10} for users to add to H_0. After performing the user action of adding v_{10}, our solution tunes parameter $k = 3$ as the v_{10} only appears in the 3-core. The answer H_0 is corresponding refined as H_1 in Figure 1(c), which is a 3-core containing the new query $\{v_6, v_{10}\}$ with the diameter of 2. Next, the user deletes v_{13} and finally obtains H_2 in Figure 1(d), which is shown to be the desired data mining community containing $\{v_6, v_{10}\}$. The proposed techniques of interactive community search in GICS-framework are easily extended to support many community models [1–3, 6, 8, 9, 17, 18, 20, 21, 24, 25, 27, 28, 31, 36, 37, 39, 40, 54, 55, 68, 70, 72, 73, 76]. In summary, we make the following contributions:

- We identify key characteristics of community models, including dense subgraph patterns, structural, and attribute metrics. Based on these model criteria, we propose a novel notation system to generalize existing community search models in a unified way. We illustrate the usefulness of our notation system using three particular instances of community search models, and formulate the problem of interactive community search (Section 3).
- We develop a fast and effective framework for interactive community search, incorporating three well-designed components of personalized recommendation, fast refinement, and parameter auto-turning. Moreover, we identify the scope of our interactive GICS-framework that can do and cannot do (Section 4).
- We design personalized recommendation algorithms to suggest a small-sized set of high-quality vertices, which are easy for users to decide insertion/deletion for further community refinement. To support effective personalization and diverse recommendations, we develop several strategies including directed candidate graph reconstruction, maximum coverage-based greedy solutions, and also the extension techniques of handling attributes and large-scale graphs (Section 5).
- We develop two important phases of community parameter auto-tuning and fast interactive refinement, which provide good community results in a fast and user-friendly way (Section 6).
- We implement three particular community search models into our interactive community search system. Experiments on ground-truth communities and user studies validate the usability, efficiency, and effectiveness of our methods (Section 7).

We review related work in Section 2 and conclude paper in Section 8.

2 Related Work

Structural community search. Table 1 summarizes various community models that employ *dense subgraph patterns*, *structural metrics*, and *attributed metrics*. Simple structural community search problems first explored by Sozio et al. [54] are to identify densely interconnected subgraphs that include specified query vertices [19, 26]. To constraint community density, key dense subgraph patterns were proposed include k-core [3, 9, 54, 68], k-truss [24, 27], k-clique [8, 70], the densest subgraph model [10, 32], and learning-based weighted components [5, 23, 30, 33]. Community models also utilize structural metrics to measure the community structure characteristics such as vertex count [9, 23, 24, 68], query distance [25], and graph diameter [2, 27, 36].

Complex community search. Recent studies have explored community search within complex graphs, including directed graphs [7, 20, 36], keyword graphs [6, 12, 18, 25, 72], spatial graphs [17, 31, 40], weighted graphs [55, 73], heterogeneous information networks (HINs) [21, 29, 74], multilayer graphs [4, 56], and dynamic graphs [28, 57, 64]. As summarized in Table 1, various attribute metrics are employed to assess attribute cohesion. For instance, Liu et al. [37] introduced an attribute score based on Jaccard distance for ensuring cohesiveness, while Zhou et al. [74] focused on identifying communities adhering to the meta-path-based $(k, \mathcal{P})$-core condition in HINs. Moreover, recent research in dynamic community search [28, 57, 64] emphasizes the efficient updating of indices in response to changes in graph topology.

Table 1: A summary of existing community search problems

Graph Type	Papers	Dense Subgraph Patterns				Structural Metrics	Attribute Metrics	Interactive Search
		k-core	k-truss	Clique	Learning-based component			
Simple	[3, 9, 68]	✓				$\|V(H)\|$	×	×
	[54]	✓				$\text{dist}(Q,H)$		
	[24]		✓			$\|V(H)\|$		
	[27]		✓			$\text{dist}(Q,H)$		
	[8, 10, 70]			✓		$\|V(H)\|$		
Attributed	[18, 21, 28]	✓				$\|V(H)\|$	$\text{attr}(H)$	
	[72]	✓				$\text{dist}(Q,H)$	$\text{attr}(H)$	
	[17, 39, 40]	✓				$\|V(H)\|$	$\text{geodiam}(H)$	
	[55, 73]	✓				$\|V(H)\|$	$\text{weight}(H)$	
	[34, 41, 74]	✓				$\|V(H)\|$	$\text{influ}(H)$	
	[37, 76]		✓			$\|V(H)\|$	$\text{attr}(H)$	
	[2, 36]		✓			$\text{diam}(H)$	$\text{attr}(H)$	
	[25]		✓			$\text{dist}(Q,H)$	$\text{attr}(H)$	
	[5, 30, 33, 38]				✓	$\|V(H)\|$	$\text{gnn}(H)$	
	[23]				✓	$\|V(H)\|$	$\text{gnn}(H)$	✓
All types	**Ours**	✓	✓	✓	✓	**All types**	**All types**	✓

Interactive graph search. Several studies investigate user interaction with graphs to enhance search outcomes [43–45]. Perozzi et al. [48] investigate community and outlier detection problems guided by user-provided nodes. Fan et al. study graph incremental computation problems and the unbounded update properties of common graph queries such as graph traversal [15]. Du et al. propose interactive algorithms for subgraph matching following query graph revisions [13]. Tao et al. focus on interactive graph search in directed acyclic graphs (DAGs) [58]. The GMine system facilitates interactive visualization for large graphs [50].

Gao et al. introduce ICS-GNN which focuses on interactive community search to identify a connected k-sized community maximizing the GNN score [23]. This approach iteratively enhances the training set by labeling additional vertices to boost effectiveness. However, extending ICS-GNN to other community models, such as k-core and k-truss models, presents challenges due to the lack of training objectives and ground-truth labels. Furthermore, ICS-GNN does not support user recommendations, which limits its usability. In contrast to ICS-GNN and other non-interactive community search methods [4, 21, 27, 29, 31, 36, 68, 70], we propose a scalable interactive community search approach that includes *fast refinement with automatic parameter tuning* and *high-quality recommendations* for enhanced usability.

3 Preliminaries

In this section, we identify key criteria of communities and propose a notation system $\mathbb{M}(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P})$ to generalize community models in a unified way. After that, we formulate the problem of *interactive community search*.

3.1 Community Criterion

We introduce elements of a typical community model, including *dense subgraph patterns*, *structural metrics*, and *attribute metrics*.

Basic graph notions and notations. Given an attributed graph $G(V, E, A)$, the vertex set and edge set are V and E, respectively. Each vertex $v \in V$ is depicted by the set of attributes $A(v)$. For a subgraph $H \subseteq G$, the neighbors of $v \in V$ is defined as $\mathsf{N}_H(v) = \{u : (v,u) \in E(H)\}$. The degree of v is the number of edges incident to v in H denoted by $\deg_H(v) = |\mathsf{N}_H(v)|$. A triangle $\triangle_{uvw}$ is a 3-clique of three vertices v, u, and w. For an edge $e = (u, v) \in E(H)$, the support of e is the number of triangles containing e in H, denoted as $\sup_H(e)$, where $\sup_H(e) = |\{\triangle_{uvw} : w \in \mathsf{N}_H(v) \cap \mathsf{N}_H(u)\}|$. The edge density of H is denoted as $\text{den}(H) = \frac{|E(H)|}{|V(H)|}$. For two vertices u, v, the shortest distance between u and v is the smallest length of all paths connecting u, v in graph H, denoted as $\text{dist}_H(u, v) \in \mathbb{N}$. Moreover, graph H is connected denoted as $\text{conn}(H) = 1$, otherwise $\text{conn}(H) = 0$. Query vertices are denoted as $Q \subseteq V(H)$. Throughout this paper, w.l.o.g., we regard the communities H to be connected and containing all query vertices, i.e., $\text{conn}(H) = 1$ and $Q \subseteq V(H)$.

3.1.1 Dense subgraph patterns. Most community models adopt a dense subgraph pattern to depict structural connections among community members. We summarize five frequently used dense subgraph models: k-core [3, 9, 17, 18, 20, 21, 28, 31, 39, 40, 54, 55, 68, 72, 73], k-truss [2, 24, 25, 27, 36, 37, 76], cliques [8, 70], the densest subgraphs [10, 32], and GNN-based models [23, 33].

k-core. A subgraph $H \subseteq G$ is k-core if every vertex has at least k neighbors in H, i.e., $\deg_H(v) \geq k$. *Coreness* is frequently used to denote the largest number $k \in \mathbb{N}$ such that H is a k-core, corresponding to the minimum degree of vertices in H, i.e., $\text{core}(H) = min_{v \in H} \deg_H(v)$. Obviously, the larger coreness $\text{core}(H)$, the stronger cohesiveness H. k-core is frequently used as the basis of community models, which applies on the constraints to satisfy $\text{core}(H) \geq k$ for a specific k [9, 17, 18, 20, 21, 28, 31, 39, 40, 55, 72] or optimization criteria to maximize coreness $\text{core}(H)$ [3, 9, 54, 68]. In summary, we can represent a k-core community H as $\text{core}(H) \geq k$.

k-truss. A subgraph $H \subseteq G$ is k-truss if every edge $e \in E(H)$ has at least $k-2$ triangles in H, i.e., $\sup_H(e) \geq k-2$. Similar to k-core, we evaluate the structural cohesiveness of k-truss H using the minimum support of edges $E(H)$, named as the *trussness*, i.e., $\text{truss}(H) = \min_{e \in E(H)} \sup_H(e) + 2$. The larger trussness truss (H), the cohesive community H. Several community models [2, 24, 25, 27, 36, 37, 76] require k-truss to be a dense subgraph pattern. In summary, we can represent a k-truss community H as $\text{truss}(H) \geq k$.

Cliques. A clique is a complete graph where every pair of nodes has an edge [8, 70]. A k-clique $H \subseteq G$ is the clique of exact k vertices with the largest edge density, i.e., $|V(H)| = k$ and $\text{den}(H) = \frac{k-1}{2}$. A similar relaxation of k-clique is γ-quasi-k-clique [8], which allows some missed edges, i.e., $|V(H)| = k$ and $\text{den}(H) \geq \gamma \times \frac{k-1}{2}$.

Densest subgraph. A subgraph $H \subseteq G$ is the densest subgraph if there exists no other subgraph $H' \subseteq G$ having a higher density than H, i.e., $H = \arg\max_{H \subseteq G} \text{den}(H)$. Besides this classical definition of edge density $\text{den}(H) = \frac{|E(H)|}{|V(H)|}$, the densest subgraph may adopt other density metrics, e.g., the fraction of edge $\frac{2|E(H)|}{|V(H)|(|V(H)|-1)}$,

the modularity based density [10, 32], and so on. The densest subgraph can be formulated to $\arg\max_{H\subseteq G} \mathsf{den}(H)$.

Learning-based component. The learning-based subgraph H is usually a connected k-sized subgraph with vertices assigned learned weights, denoted by $\mathsf{conn}(H) = 1$ and $|V(H)| = k$. A well-known GNN-based community contains a set of k vertices with the highest GNN scores, i.e., $\mathsf{gnn}(H) = \sum_{v\in V(H)} \mathsf{weight}(v)$, based on their structural and feature similarity to query vertices [23]. These weight-based connected community search problems [5, 23, 30, 33, 38] could be regarded as a 1-core with attribute score optimization, i.e., $\arg\max_{H\subseteq G} \mathsf{weight}(H)$, where $\mathsf{core}(H) \geq 1$.

3.1.2 Structural metrics. *Community distance* measures the communication cost between community members. Given a community H, common graph distance-based metrics include the longest length of shortest paths as *diameter* $\mathsf{diam}(H) = \max_{u,v\in V(H)} \mathsf{dist}_H(u,v)$ [2, 4, 27, 36], *radius* $\mathsf{radius}(H) = \min_{v\in V(H)}\{\max_{u\in V(H)} \mathsf{dist}_H(u,v)\}$, and also *query distance* $\mathsf{dist}(Q,H) = \max_{u\in Q, v\in V(H)} \mathsf{dist}_H(u,v)$ where Q is a set of query vertices [25, 54]. A small graph distance of H indicates the closeness of community members. *Community size* counts on the *number of vertices and edges* in H , denoted by $|V(H)|$ and $|E(H)|$, respectively [3, 8, 9, 20, 21, 23, 24, 28, 37, 64, 68, 74].

3.1.3 Attribute metrics. The *attribute similarity* computes the homogeneity of nodes' associated attributes, e.g., categorized keywords [2, 6, 12, 30, 72], spatial locations [17, 31, 40], and influential importance [34, 41, 74]. The *common attribute* of community H is $\mathsf{attr}(H) = \bigcap_{v\in V(H)} A(v)$, where $A(v)$ represents the categorized keywords of v [2, 18]. In geo-social networks, the *spatial diameter* of H is $\mathsf{geodiam}(H) = \max_{u,v\in V(H)} \mathsf{geod}_H(u,v)$ where $\mathsf{geod}(u,v)$ represents the distance between u and v in the spatial space [17, 40]. In influential social networks, the community *influence* measures the minimum influence of nodes in H $\mathsf{influ}(H) = \min_{v\in V(H)} A(v)$ where $A(v)$ here represents the influential importance of v [34]. The *group weight* in the weighted graph can be defined as $\mathsf{weight}(H) = \sum_{e\in E(H)} w(e)$ where $w(e)$ represents the edge weight [73]. All attribute metrics above can be calculated incrementally during interactive community search.

3.2 Community Notation Systems and Instances

Based on the above dense subgraph patterns, structural, and attribute metrics, we present a community notation system $\mathbb{M}(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P})$ to depict arbitray community model and illustrate the instances. We first present three key elements of communities as the metrics $\mathcal{M}$, operations $\mathcal{O}$, and parameters $\mathcal{P}$.

Metrics $\mathcal{M}$. We use a set of metrics $\mathcal{M}$ to describe the properties of vertices, edges, and subgraphs in community H. Therefore, we represent the metrics as $\mathcal{M} = \{\mathsf{core}(.), \mathsf{truss}(.), \mathsf{den}(.), \mathsf{conn}(.), \mathsf{diam}(.), \mathsf{radius}(.), \ldots, \mathsf{attr}(.), \mathsf{gnn}(.), \mathsf{geodiam}(.), \mathsf{influ}(.)\}$.

Operations $\mathcal{O}$. We use a set of operations $\mathcal{O}$ to optimize objectives and limit constraints, where $\mathcal{O} = \{>, <, \geq, \leq, =, \neq, \max(.), \min(.), \arg, \subseteq, \supseteq, \cup, \cap, \ldots, |.|, +, -, *, /, \sum, \Pi\}$. Here, $\max(.)/\min(.)$ represents the maximum/minimum optimization of functions, $|.|$ indicates the set cardinality, and $\leq/\geq$ represent the inequality constraints.

Parameters $\mathcal{P}$. We use a set of parameters $\mathcal{P}$ to indicate the community requirements quantified. First, query vertices are usually required to appear in community answers, i.e., $\Phi(Q,H) = |Q|$. Second, dense subgraph models involve additional parameters of k, d, and others to constrain cohesiveness or communication cost, i.e., $\mathsf{core}(H) \geq k$. Third, community size constraints can be an upper or lower bound or belong to a range, i.e., $a \leq |V(H)| \leq b$.

Based on the querying graph $\mathcal{G}$ and three elements $\mathcal{M}, \mathcal{O}, \mathcal{P}$ above, the problem of existing community search models can be formulated to find the answer of community H as

$$H \leftarrow \mathbb{M}(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P}).$$

Therefore, we derive a unified community notation system $\mathbb{M}(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P})$ for describing any community models. To illustrate the usefulness of proposed notation system, we apply $\mathbb{M}(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P})$ on three representative instances, including the *closest truss community search* (CTC) [27], *attributed community search* (ACQ) [18], and *GNN-based interactive community search* (ICS-GNN) [23].

Instance-1 $\mathbb{M}_{CTC}$. The CTC model [27] finds a connected k-truss community H containing query vertices Q with the smallest diameter. The objective of CTC can formulated as a minimization problem to find a community $H \in \Omega_1$ where the search space Ω_1 is constrained by $\mathcal{G}_1, \mathcal{M}_1, \mathcal{O}_1, \mathcal{P}_1$, such that

$$\mathbb{M}_{CTC}(\mathcal{G}_1, \mathcal{M}_1, \mathcal{O}_1, \mathcal{P}_1) = \arg\min_{H\in\Omega_1} \mathsf{diam}(H)$$

where $\mathcal{G}_1 = (V_1, E_1)$, $\mathcal{M}_1 = \{\mathsf{diam}(.), \mathsf{truss}(.), \mathsf{conn}(.), \Phi(.)\}$, $\mathcal{O}_1 = \{\subseteq, \min(.), \geq, =, |.|\}, \mathcal{P}_1 = \{Q, k\}$, and $\Omega_1 = \{H | H \subseteq \mathcal{G}_1, \mathsf{truss}(H) \geq k, \mathsf{conn}(H) = 1, \Phi(Q,H) = |Q|\}$.

Throughout the remaining of this paper, we omit the constraints $\mathsf{conn}(H) = 1$ for a connected community, $\Phi(Q,H) = |Q|$ for query-dependent communities, and also parameters $\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P}$ as model $\mathbb{M}(.)$, for simplicity. We can rewrite the above model as $\mathbb{M}_{CTC}(.) = \arg\min_{H\in\Omega_1} \mathsf{diam}(H)$, where $\Omega_1 = \{H | H \subseteq \mathcal{G}_1, \mathsf{truss}(H) \geq k\}$.

Instance-2 $\mathbb{M}_{ACQ}$. The ACQ model [18] finds a connected k-core containing query vertex q with the maximum common attributes over attributed graph $\mathcal{G}_2 = (V_2, E_2, A_2)$, w.r.t., the parameter query q and input query attributes S. The ACQ model can be formulated as $\mathbb{M}_{ACQ}(.) = \arg\max_{H\in\Omega_2} |S \cap \mathsf{attr}(H)|$, where $\Omega_2 = \{H | H \subseteq \mathcal{G}_2, \mathsf{core}(H) \geq k\}$.

Instance-3 $\mathbb{M}_{GNN}$. The ICS-GNN model [23] finds a k-sized connected subgraph containing query vertices Q with the largest GNN-based score over attributed graph $\mathcal{G}_3 = (V_3, E_3, A_3)$. The ICS-GNN model can be formulated as $\mathbb{M}_{GNN}(.) = \arg\max_{H\in\Omega_3} \mathsf{gnn}(H)$, where $\Omega_3 = \{H | H \subseteq \mathcal{G}_3, |V(H)| = k\}$.

3.3 Problem Formulation

In the following, we formulate the problem of interactive community search (ICS-problem) for any community model $\mathbb{M}_x \in$ CSM. In the interactive setting, we allow users to modify community search result $H_0 \xrightarrow{Q^{\pm}} \hat{H}$ by clicking vertices $Q^{\pm}$ to adding/removing into/from the community, where H_0 and $\hat{H}$ are the initial and optimal communities, respectively.

[Interactive community search problem (ICS-problem)]. Given a graph $\mathcal{G} = (V, E, A)$, a community model $\mathbb{M}_x(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P}_0) \in$ CSM, a given set of queries $Q_0 \in$ parameter $\mathcal{P}_0$, an initial result

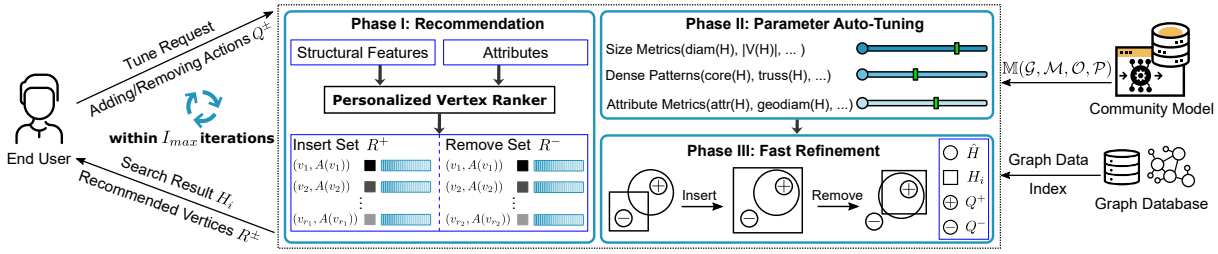


Figure 2: The GICS for interactive community search.

of community $H_0 = \mathbb{M}_x(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P}_0)$, and the maximum interactive round as $I_{max} \in \mathbb{Z}^+$, the ICS-problem is to iteratively revise community H^{i-1} to H^i at the i-th round, where $1 \le i \le I_{max}$, $H^i = \mathbb{M}_x(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P}_i)$, the query $Q_i = Q_{i-1} \cup Q^{\pm}$ by a sequence of actions to add/remove vertices $Q^{\pm} \subseteq V$, and $Q_i \in \mathcal{P}_i$.

The objective of interactive community search is to compute the community H_i close to the user target community $\hat{H}$ guide by user-provided interactive actions $Q^{\pm}$ within I_{max} iterations, such that $\lim_{i \to I_{max}} |H_i - \hat{H}| = 0$, where $0 \le i \le I_{max}$. In the i-th round of interaction, users can choose to add/remove operations to a specific vertex or stop the search. There are three kinds of relationships between H_i and $\hat{H}$: (1) $H_i \subset \hat{H}$; (2) $H_i \supset \hat{H}$; (3) $H_i \cap \hat{H} \neq \emptyset$. Case (1) requires to add vertices to H_i while Case (2) requires to remove vertices from H_i. Case (3) requires both adding and removing actions.

Discussion on convergency and optimality. Given a ground-truth community $\hat{H}$, it takes $O(n)$ interactions to converge to $\hat{H}$ in worst cases, where $n = |V|$. The rationale is as follows. At each i-th iteration, users can mark the label of insertion/deletion for one unlabeled vertex $v \in V$, i.e., insert $v \in V(\hat{H})$ into H_i and delete $v \in V(H_i)$ from H_i. After at most n iterations by marking all vertices in G, the optimal community $\hat{H}$ is identified. Unfortunately, there seems to be no optimal sequence of refinements to achieve $\hat{H}$, as the interaction process involves human willingness, that is, users may perform different actions at the same round of interaction. Therefore, a good solution for interactive community search should admit at least two requirements: 1) Fast refinement. It takes a small number i of actions to reach the target community, i.e., $H_i = \hat{H}$; 2) High-quality recommendation for user-friendly actions. It recommends a small number of high-quality vertices $R^{\pm}$ to add/remove, i.e. $Q^{\pm} \subseteq R^{\pm}$, instead of the whole vertex set such as $Q^{\pm} \subseteq V$, which saves the user efforts and provides user-friendly service.

Example 1. Figure 1 gives an example of interactive community search. Consider a collaboration network $\mathcal{G}$ in Figure 1(a) with 13 vertices and 26 edges. Each scholar vertex has an attribute of research topics in $\{DB, DM, ML\}$. $\mathbb{M}_x$ model finds a connected k-core community with the smallest diameter, i.e., $\mathbb{M}_x = \arg\min_{H \in \Omega} \text{diam}(H)$, where $\Omega = \{H | H \subseteq G, \text{core}(H) \ge k\}$. Given community model $\mathbb{M}_x$, query vertex $Q = \{v_6\}$, $k = 4$, and $I_{max} = 2$, the initial community search result H_0 is a 4-core community with $\text{diam}(H) = 1$ shown in Figure 1(b). After adding v_{10}, there is no 4-core containing v_{10}, thus we auto-tune by decreasing k to 3 and refine the answer to H_1 in Figure 1(c). H_1 is a 3-core community with $\text{diam}(H) = 2$. After deleting v_{13}, the refined community H_2 has been extracted in Figure 1(d), which is a data mining (DM) group.

Algorithm 1 Interactive Community Search Framework

Input: Graph $\mathcal{G} = (V, E, A)$, community model $\mathbb{M}_x \in \mathcal{CSM}$, query vertices Q_0, interactive budget I_{max}
Output: Refined community H^*

```
Compute initial community H_0 ← M_x(G, M, O, P_0) ;
i ← 1;
while H_i = Ĥ or 1 ≤ i ≤ I_max do
    Recommend interactivevertices R^± by Algorithm 2;
    End-user specifies Q^± from recommended vertex set R^±;
    Tune parameters P_{i-1} to P_i by Algorithm 6;
    Refine and find answer H_i ← M_x(G, M, O, P_{i+1}) by Algorithm 7;
    i ← i + 1;
Return a refined community H_i;
```

4 The Proposed GICS-Framework

We present an overview of GICS-framework in Figure 2 to be adapted into arbitrary community model, which consists of three phases in Algorithm 1.

- **Phase I: personalized adding/deleting recommendation.** This module first provides a user-friendly service to list a small number of recommended add/delete nodes $R^{\pm}$ for community refinement. This gives users quick ideas to select candidates for revising the community to the right answer. Specifically, we propose new techniques to recommend a set of nodes that have high community relevance and large coverages over our reconstructed directed graphs.
- **Phase II: auto-tuning parameter mechanism.** We construct an auto-adjusted parameter mechanism to infer a proper parameter setting for new queries and underlying community. It adjusts proper parameter values $\mathcal{P}_i$ based on the existing community H_{i-1} and the user's new action on $Q_i^{\pm}$. This phase maintains all kinds of structural and attributed parameters based on user's adding/removing actions.
- **Phase III: fast refinement.** We propose efficient refinement algorithms to quickly identify answers based on the existing answer H_{i-1} and new queries Q_i, which provides short response time in a user-friendly interaction. To ensure fast community refinement, it constructs a small candidate subgraph to support the community's partial update.

The scope of our ICS problem and framework. We identify the scope of our interactive community search problem and the proposed solution GICS-framework, that can do and cannot do.

- **Our scope**. This work focuses on a specific problem of *graph query processing*, i.e., community search, which aims to identify *a single connected community containing all user input query vertices* meanwhile satisfying the predefined constraints and

optimizations. We develop a unified notation system CSM using $\mathbb{M}_x(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P}_0)$ to represent various community search models [1–3, 6, 8, 9, 17, 18, 20, 21, 24, 25, 27, 28, 31, 36, 37, 39, 40, 54, 55, 68, 70, 72, 73, 76], which can be formulated to tackle in our ICS-problem. Note that similar graph query processing tasks to *community search* can also be solved by our GICS-framework. We highlight three studies [51] [63] [48] as follows. First, the minimum Wiener connector problem [51] finds a connected subgraph $H \subseteq G$ containing query vertices Q that minimizes the Wiener index, the total of all pairwise distances in H. Our GICS first recommends vertices for user insertion/deletion. It is unnecessary to tune parameters since no input parameters are required except Q. It then finds the shortest path to connect inserted vertex to H and expands it to a candidate subgraph. It finally iteratively peels vertices to minimize the Wiener index. Second, Whang et al. [63] propose a seed set expansion approach to find overlapping communities with low conductance. Assume that there is one connected community containing all seeds. The seeds act as query nodes in GICS, supporting recommendations on node adjustments. It tends to refine a low conductance community from candidate subgraphs and then expand to the answer by connecting "whiskers" via "bridges" [63]. Third, the focused attributed graph clustering [48] finds clusters based on attributes of user-provided exemplar vertices. Assume that there is one connected cluster containing all user-provided exemplar vertices, which we treat as query vertices in GICS. GICS refines communities based on FOCUSCO model [48]. It iteratively finds a low-weighted conductance community by adding vertices from the candidate subgraph, ensuring both dense structure and similar attributes to exemplars. Motivated by [15], GICS suggests those querying tasks equipped with *node queries* and *graph localizable algorithms* that can find the updated answer in a local small graph may also be supported by GICS-framework.

- **Out of our scope**. However, except for community search, our GICS cannot address a broader area of graph analytics tasks, especially *those tasks without the input queries*, such as subgraph pattern counting [16] (e.g., triangle counting [47], rectangle counting [60]), graph decomposition [69], graph summarization [52, 75], graph traversal [14], and graph clustering [42]. Consider the task of community detection that identifies all communities $\{C_1, \ldots, C_k\}$ for $k \in \mathbb{Z}^+$ in a graph, which is hard to handle by our GICS in an interactive way. The difficulties lie in two aspects. First, GICS supports a simple and user-friendly action of adding/deleting nodes, which is a binary decision. However, it is hard to apply GICS for community detection, by changing decisions to be multiple choices as determining from vertex $x \in C_i$ to $x \in C_j$, where $1 \leq j \leq k$ and $i \neq j$. In other ways, asking users whether $x \in C_j$ holds for $1 \leq j \leq k$ at each interaction, which may involve too many $k-1$ rounds of interactions for a large k. This is not user-friendly. Second, community detection usually has no query vertices, leading to communities that are not user-oriented and prioritize global optimization over local search objectives that GICS supports. Additionally, GICS may struggle with interactive graph query processing tasks that involve complex optimizations and constraints, such as pattern matching and keyword searches.

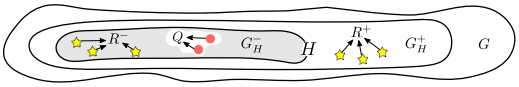


Figure 3: Recommendation candidates R^- and R^+.

5 Personalized Community Recommendation

In this section, we introduce personalized recommendation algorithms for interactive community search. We start with a straightforward recommendation solution by finding the maximum coverage vertex set. To enhance recommendation quality, we propose a personalized maximum coverage approach that considers vertex community relevance, highlighting varying importance of vertices.

5.1 Max-coveraged Recommendation

The problem setting of recommendation is as follows. Let community search iteration number i correspond to the returned community H_i. The objective is to recommend vertices $R_i^{\pm} = R_i^+ \cup R_i^-$ to help users in revising H_i to H_{i+1}, where the insertion candidate $|R_i^+| = r$ and the deletion candidate $|R_i^-| = r$, thus $|R_i^{\pm}| = 2r$ for a small integer $r \in \mathbb{Z}^+$. User can choose to add a vertex $v \in R_i^+$ or delete $u \in R_i^-$, leading to $Q_i^+ = \{v\}$ and $Q_i^- = \{u\}$. Thus, the new query $Q_i = Q_{i-1} \cup Q_i^+$ or $Q_i = Q_{i-1} \setminus Q_i^-$. For simplicity, we can omit subscripts and use $H, R^{\pm}, R^+, R^-, Q^{\pm}, Q^+, Q^-$ to represent H_i, $R_i^{\pm}, R_i^+, R_i^-, Q_i^{\pm}, Q_i^+, Q_i^-$, respectively.

Recommendation candidates R^- and R^+. The deletion candidate set R^- must include vertices in current answer H but exclude any previously confirmed query vertices Q_{i-1}, i.e. $R^- \subseteq V(H) \setminus Q^+$. We define the search region of R^- as $G_H^- = (V(G_H^-), E(G_H^-))$, where $V(G_H^-) = V(H) \setminus Q^+$, $E(G_H^-) = \{(v,u) \in E : v,u \in V(G_H^-)\}$. The insertion candidate R^+ admits both of $R^+ \subseteq V$ and $R^+ \cap V(H) = \emptyset$, indicating that vertices not in H could be further suggested to be added into community. Given that a graph can have millions of vertices while human cognitive capacity for suggestions is limited to tens or hundreds, the vertices in R^+ need to be highly correlated with community H. Thus, we focus on a small candidate region of $G_H^+ \subseteq G$, which is a d-hop neighborhood subgraph of H, $d \in \mathbb{Z}^+$, where the vertex set $V(G_H^+) = \{u \in V \setminus V(H) : \text{dist}(v,u) \leq d, v \in V(H)\}$, the edge set $E(G_H^+) = \{(v,u) \in E : v,u \in V(G_H^+)\}$. We set a very small number d as $d = 1$ or $d = 2$, due to consideration of efficiency in a possible small world in social networks. Although it expands only d-hops each time, we can still explore those candidate vertices far from initial query nodes Q_0 using a few more iterations. As a result, we consider the scope of potential candidates as $R^+ \subseteq V(G_H^+)$ and $R^- \subseteq V(G_H^-)$.

Small and diverse recommendation. In terms of users' aspects, the recommended vertices should satisfy two good properties: i) the size of $R^{\pm}$ shall be not too large for easily selection. This could be flexibly adjusted by a parameter $r = \frac{|R^{\pm}|}{2}$; and ii) the diversity of $R^{\pm}$ shall be large, i.e., not all recommended vertices are similar. In other words, the neighborhood subgraph of $R^{\pm}$ could differ, leading to a large union neighborhood of the coverage of $R^{\pm}$. This benefits users by identifying those vertices that could become community members. Next, we give a definition of vertex coverage.

Definition 1 (Vertex Coverage)**.** Given a community answer H and a candidate graph G_H, the coverage of a vertex $x \in V(G_H)$ is

Algorithm 2 Personalized Recommendation Solution (PADR)

Input: Candidate graph G_H, community H, query vertices Q
Output: Recommendation vertices R

1: Construct directed graph G_D from G_H in Algorithm 3;
2: Compute all personalized community relevance $\text{PCR}(v)$ in Algorithm 4;
3: Ranking to find out top r vertices R in Algorithm 5;
4: **return** Recommendation vertices R;

defined as $\text{cov}(x) = |N^*(x)|$, which includes the set of x's neighbors and x itself as $N^*(x) = \{u \in V(G_H) : (x,u) \in E(G_H)\} \cup \{x\}$.

The coverage of a vertex counts on the size of 1-hop neighborhood vertices. The larger coverage of recommended vertices, the higher possibility of offering a better recommendation choice. Note that we set $G_H = G_H^+$ for adding recommendations and $G_H = G_H^-$ for deleting recommendations separately. For a given set of vertices S in graph G_H, the coverage of S is $\text{cov}(S) = |\bigcup_{x \in S} N^*(x)|$.

Problem statement. Given a current answer H and a candidate subgraph G_H, an integer $r \in \mathbb{Z}^+$, the objective of recommendation is to find a small-sized set of $2r$ vertices $R^{\pm} = R^+ \cup R^-$ such that it achieves the maximum coverage, i.e., $\max_{R^{\pm}=R^+\cup R^-}(\text{cov}(R^+) + \text{cov}(R^-))$ where $R^+ \subseteq V(G_H^+), R^- \subseteq V(G_H^-), |R^+| = |R^-| = r$. This problem can be shown to be NP-Hard by the reduction of a classical problem of maximum coverage problem in an intuitive way. To select the best recommendation of top-r vertices, one straightforward method adopts a greedy strategy to select the vertex with the largest coverage in each round until $|R^{\pm}| = 2r$.

5.2 Personalized Recommendation PADR

Motivations. The probability of a vertex being part of a community can vary significantly. Straightforwardly, given the current answer H, vertices $V \setminus V(H)$ closer to query nodes Q are more likely to be a target community member than those vertices of graph periphery farther away from Q, due to the common properties of dense structure and community connectivity.

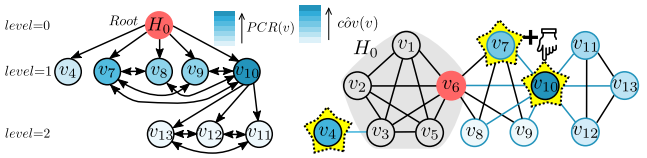
An overview of PADR. We propose an optimization problem to find r-sized recommended vertices to maximize the total personalized community relevance score of covered vertices. We present the personalized adding/deleting recommendation solution (PADR) in Algorithm 2. Specifically, PADR involves three main steps. First, we construct a directed graph G_D from the existing candidate subgraph G_H for further relevance calculation (Algorithm 3). Second, we compute the personalized community relevance of all vertices in G_D (Algorithm 4). Finally, we develop a greedy algorithm to find r-sized diverse vertices to maximize the total importance of covered vertices (Algorithm 5).

Step-1: Directed candidate graph reconstruction. In the original candidate graph G_H, it does not distinguish source vertices and the edges' influential direction that propagates the community relevance from Q. Therefore, we transform an undirected graph G_H into a directed graph G_D. The key idea is to set "correct" community vertices Q as the initial seeds and generate directed edges from Q to others vertices $V(G_H)$ by expanding from neighbors level by level. Specifically, we define three cases of directed edges. Consider a pair of vertices v and u with $(v,u) \in E(G_H)$, if v and u have the same level, i.e., $level(v) = level(u)$, we create two directed edges $\langle v,u\rangle$ and $\langle u,v\rangle$; if v has a smaller level than u, i.e., $level(v) < level(u)$, we create one directed edge from v to u, i.e., $\langle v,u\rangle$; otherwise, if $level(v) > level(u)$, we create a directed edge $\langle u,v\rangle$.

Algorithm 3 Directed Candidate Graph Reconstruction

Input: Candidate graph G_H, query vertices Q
Output: Directed graph $G_D = (V(G_D), E(G_D))$

1: Create a set of isolated vertices $V(G_D) = V(H)$;
2: **for** each vertex $v \in Q$ **do** Set $level(v) \leftarrow 0$;
3: Mark all vertices Q as visited and add Q into a queue $D_q = Q$;
4: **while** $D_q \neq \emptyset$ **do**
5: Pop a vertex v from D_q;
6: **for** each unvisited $u \in N_{G_H}(v)$ **do**
7: $level(u) \leftarrow level(v) + 1$;
8: Mark u as visited and $D_q \leftarrow D_q \cup \{u\}$;
9: **for** each $u \in N_{G_H}(v)$ **do**
10: **if** $level(u) > level(v)$ **then**
11: Add a new edge $\langle v,u\rangle$ into graph G_D;
12: **else if** $level(u) < level(v)$ **then**
13: Add a new edge $\langle u,v\rangle$ into graph G_D;
14: **else if** $level(u) = level(v)$ **then**
15: Add two new edges $\langle v,u\rangle$ and $\langle u,v\rangle$ into G_D;
16: **return** directed graph G_D;



(a) Construct G_D (b) Personalized recommendation

Figure 4: An example of top-r personalized recommendation.

Algorithm 3 outlines the details of directed candidate graph reconstruction. Specifically, deleting recommendation takes an input of graph G_H^- and the updated i-th round query Q_i^+ regarded to be fallen into community, which finally produces a directed graph G_D; adding recommendation takes an input of graph G_H^+ and the source vertices $Q = V(H)$, which finally produces a directed graph G_D. It first constructs G_D with all isolated vertices of $V(H)$ (line 1). Then, it starts a queue of all query nodes Q by marking them as visited and setting their level as 0, which treats Q as the root of a directed graph G_D (lines 2-3). Then, it performs a BFS search to expand the graph construction of G_D (lines 4-15). In each iteration, it pops a vertex v from D_q (line 5). For each unvisited neighbor $u \in N_{G_H}(v)$, it sets $level(u)$ by increasing $level(v)$ by one and add u into D_q (lines 6-8). Then, it adds a new directed edge $\langle v,u\rangle$ for nodes from an early level to a later level, i.e., $level(v) \leq level(u)$. If $level(v) = level(u)$, we add two new directed edges $\langle v,u\rangle$ and $\langle u,v\rangle$ into graph G_D. The time complexity of Algorithm 3 is $O(|E(G_H)|)$ by scanning the whole graph G_H once.

Step-2: Personalized community relevance computation. In this step, we compute the personalized community relevance (PCR) for all vertices w.r.t. the refined query Q over the constructed graph G_D. Assume the number of vertices in G_D is denoted as $n_d = |V(G_D)|$. For a vertex v, the set of in-neighbors and out-neighbors are defined as $N_{G_D}^+(v) = \{u \in V(G_D) : \langle u,v\rangle \in E(G_D)\}$ and $N_{G_D}^-(v) = \{u \in V(G_D) : \langle v,u\rangle \in E(G_D)\}$, respectively. Inspired by personalized PageRank computation [62, 71], we intend to calculate the personalized community relevance of a vertex v w.r.t. multiple personalized source vertices Q. Note that we do not calculate the

Algorithm 4 Personalized Community Relevance Computation

Input: Directed graph G_D, query vertices Q, a damping constant parameter $\alpha \in [0, 1]$, iteration $l_{max} \in \mathbb{Z}^+$, a constant tolerance $\delta \in \mathbb{R}^+$
Output: Personalized Community Relevance $\mathrm{PCR}(v)$

1: Initialization: $n_D \leftarrow |V(G_D)|$;
2: **for** each vertex $v \in V(G_D) \setminus Q$ **do** $\mathrm{PCR}(v) \leftarrow 0$;
3: **for** each vertex $v \in Q$ **do** $\mathrm{PCR}(v) \leftarrow |N^+_{G_D}(v)|$;
4: **for** $i \leftarrow 1$ to l_{max} **do**
5: Assign $change \leftarrow 0$;
6: **for** each vertex $v \in V(G_D) \setminus Q$ **do**
7: $\mathrm{PCR}'(v) \leftarrow \frac{1-\alpha}{n_D} + \alpha \cdot \sum_{u \in N^+_{G_D}(v)} \frac{\mathrm{PCR}(u)}{|N^-_{G_D}(u)|}$;
8: $change \leftarrow change + |\mathrm{PCR}(v) - \mathrm{PCR}'(v)|$;
9: $\mathrm{PCR}(v) \leftarrow \mathrm{PCR}'(v)$;
10: **if** $change < \delta$ **then break**;
11: **for** each vertex $v \in Q$ **do**
12: Assign the maximum PCR value: $\mathrm{PCR}(v) \leftarrow 1$;
13: **return** all Personalized Community Relevance $\mathrm{PCR}(v)$;

accumulated PCR values propagated between two vertices in Q. Therefore, we treat the whole set of vertices Q as *one virtual source vertex*. We define the personalized community relevance of vertex $v \in V(G_D) \setminus Q$ by $\mathrm{PCR}(v)$ as follows.

$$\mathrm{PCR}(v) \leftarrow \frac{1-\alpha}{n_D} + \alpha \cdot \sum_{u \in N^+_{G_D}(v)} \frac{\mathrm{PCR}(u)}{|N^-_{G_D}(u)|} \tag{1}$$

Algorithm 4 outlines the procedure of computing personalized community relevance for all vertices in directed graph G_D. At the initialization stage, it assigns $\mathrm{PCR}(v)$ as the number of out neighbors of v for $v \in Q$ and $\mathrm{PCR}(v) = 0$ for $v \in V(G_D) \setminus Q$. In this way, the PCR value is transferred from Q to vertices outside evenly (lines 1-3). Next, it iteratively computes PCR values by at most l_{max} iterations. At each round, it first assigns an initial value of total change as 0 (line 5). Then, it updates PCR value via Eq. 1 for each vertex $v \in V(G_D) \setminus Q$ and accumulates the total change between two rounds' PCR values (lines 6-9). When the total change reaches a predefined constant tolerance δ, the algorithm early terminates (line 10); otherwise, it continues until repeating l_{max} times.

Remarks. Unlike traditional multi-source personalized PageRank computation, the key of our personalized community relevance computation in Algorithm 4 is to reconstruct a small directed graph G_D rather than using the original graph G. Note that our computation of PCR is a heuristic approach, which runs in a finite number of iterations and provides no accuracy guarantee of the resulting scores. Fortunately, it provides a relative measure of importance prioritizing proximity to source vertices Q. Thus, we can distinguish different importance of vertices, which is vital to offer a fast and relative ranking for recommendation. In addition, we develop techniques to accelerate PCR in two-fold aspects. First, we ignore the edges between community members in graph G_D, as all these community members of Q already appeared in answers. Second, we set the iteration parameter l_{max} as the number of levels in G_D, where l_{max} is a small number in practice. This can lead to accelerating our PCR computation significantly. Further accelerating PCR computation could be extended by approximate multi-source PPR computation [22] and other techniques [53] [35] [61] [62] [71] [67].

Step-3: Personalized community relevance based maximum coverage. In this step, we intend to find a set of r-sized vertices

Algorithm 5 Personalized Maximum Coverage Recommendation

Input: Directed graph G_D, all $\mathrm{PCR}(v)$ values, query vertices Q
Output: Recommendation vertices R

1: Construct a weighted directed graph $\hat{G}_D$ based on G_D and $\mathrm{PCR}(v)$ for $v \in V(G_D)$;
2: Initialize $R \leftarrow \emptyset$;
3: Initialize a priority queue $PQ \leftarrow \emptyset$;
//Pick the top-r insertion candidates;
4: **for** each vertex $v \in \hat{G}_D \setminus Q$ **do**
5: Push vertex v with its weighted $\hat{\mathrm{cov}}(v)$ into PQ;
6: **while** $|R| < r$ **do**
7: Pop out a vertex u^* with the largest coverage $\hat{\mathrm{cov}}(u^*)$;
8: **if** $\hat{\mathrm{cov}}(u^*)$ is not up-to-date **then**
9: Recalculate $\hat{\mathrm{cov}}(u^*)$ and push u^* into PQ again;
10: **else** $R \leftarrow R \cup \{u^*\}$;
11: **return** the recommended vertices R;

that maximizes coverage based on personalized community relevance in G_D. Specifically, we first convert directed graph G_D into a weighted directed graph $\hat{G}_D$, assigning each vertex v a weight of community relevance $\mathrm{PCR}(v)$. The coverage of a vertex v is defined as the accumulated score of community relevance in the neighborhood of v, i.e., $\hat{\mathrm{cov}}(v) = \sum_{u \in N^*_{\hat{G}_D}(v)} \mathrm{PCR}(u)$. For a set of vertices S, coverage is $\hat{\mathrm{cov}}(S) = \sum_{u \in \bigcup_{v \in S} N^*_{\hat{G}_D}(v)} \mathrm{PCR}(u)$. Therefore, the objective of finding a diverse and important vertex set in our new recommendation problem is formulated as

$$\max_{R^{\pm} = R^+ \cup R^-} (\hat{\mathrm{cov}}(R^+) + \hat{\mathrm{cov}}(R^-)),$$

where $R^+ \subseteq V(\hat{G}_D) \setminus V(H), R^- \subseteq V(H) \setminus Q^+, |R^+| = |R^-| = r$.

We propose a greedy selection method in Algorithm 5 to find top-r vertices by picking up a vertex with the largest weighted coverage score in each iteration, reflecting its neighborhood's community relevance. Moreover, the problem of finding the optimal solution is NP-hard, which can be reduced from a classical NP-hard maximum coverage problem. Thus, our greedy solution gives a promising recommendation answer. Specifically, it first constructs a weighted directed graph $\hat{G}(D)$ and initializes recommendation answers as empty sets, i.e., $R \leftarrow \emptyset$ (lines 1-2). Next, it uses a priority queue to maintain vertices v associated with weighted coverage scores $\hat{\mathrm{cov}}(v)$ (lines 3-5). It then iteratively selects the vertex with the largest coverage one by one into answer R (lines 6-10). Finally, it returns the recommendation answer R. Specifically, we set $G_D = G^+_D$, $Q = V(H)$ for adding recommendation, and $G_D = G^-_D$, $Q = Q^+$ for deleting recommendation respectively. After running Algorithm 5 twice to obtain R^+ and R^-, it returns the answer $R^{\pm} = R^+ \cup R^-$. This algorithm can be extended to handle attributed graphs by calculating attribute coverage, which can then be combined with structural coverage to produce an integrated score.

5.3 Complexity Analysis

For graph $G(V, E)$, we denote the number of vertices and edges as $n = |V|$ and $m = |E|$. In the interactive process, a candidate subgraph G_H of G has $\bar{n}$ vertices and $\bar{m}$ edges. W.L.O.G., we assume that $\bar{n} - 1 \leq \bar{m}$ for a connected graph G_H, i.e., $O(\bar{n}) \subseteq O(\bar{m})$ [27].

Theorem 5.1. *The recommendation phase in Algorithm 2 takes* $O(\min\{\bar{m}, r\bar{d}_{max} + \bar{n}\} \cdot \log \bar{n})$ *time and* $O(\bar{m})$ *space.*

Proof. The recommendation phase in Algorithm 2 consists of three steps: directed graph construction in Algorithm 3, personalized community relevance computation in Algorithm 4, and personalized maximum coverage recommendation in Algorithm 5. We analyze the time complexity of three steps one by one.

First, Algorithm 3 constructs a directed candidate graph G_D by scanning the entire candidate graph G_H once, which takes $O(\bar{m} + \bar{n}) = O(\bar{m})$ time. Second, Algorithm 4 computes personalized community relevance of each vertex in G_D iteratively in a total of l_{max} rounds (lines 4-10 of Alg. 4), which takes $O(l_{max} \cdot \bar{m})$ time. Thirds, Algorithm 5 first computes the coverage score $\hat{\text{cov}}(v)$ for each vertex v by access all v's neighbors (line 1 of Alg. 5), which takes $O(\bar{m})$ time. Then, it creates a priority queue PQ to store all vertices in decreasing order of $\hat{\text{cov}}(v)$ (lines 2-5 of Alg. 5), which takes $O(\bar{n} \log \bar{n})$ time. After that, it takes r iterations to pick top-r vertices u^* with the largest $\hat{\text{cov}}(u^*)$ for recommendation (lines 6-10). After select u^* into R, it needs to at most update $\hat{\text{cov}}(v)$ for $v \in N(u^*)$ in PQ, which takes $O(|N(u^*)| \log \bar{n})$ time. Assume that the maximum degree $\bar{d}_{max} = \max_{u \in V(G_D)} |N(u^*)|$. Thus, the time of updating PQ in r iterations is $O(\min\{r\bar{d}_{max}, \bar{m}\} \log \bar{n})$ in worst case. Overall, Algorithm 5 takes $O(\bar{m} + \bar{n} \log \bar{n} + \min\{r\bar{d}_{max}, \bar{m}\} \log \bar{n})$ $= O(\min\{\bar{m}, r\bar{d}_{max} + \bar{n}\} \log \bar{n})$ time and $O(\bar{m})$ space.

6 Fast Community Refinement

We introduce two important phases of *community parameter auto-tuning* and *fast interactive refinement*, which provide good community results in a fast and user-friendly way.

6.1 Parameter Auto-Tuning

Given a community model $\mathbb{M}_x(\mathcal{G}, \mathcal{M}, \mathcal{O}, \mathcal{P})$, parameters $\mathcal{P}$ can be complex, as summarized in Table 1. Generally, four kinds of parameters are required in a community search model, including query vertices, density, community size, and attributes. Some models $\mathbb{M}_x$ require a single parameter, while others may require users to set up multiple parameters for $\mathcal{P}$. Parameters $\mathcal{P}$ can be further categorized into numbers X, ranges $[X, Y]$, keywords $\{w_1, \ldots, w_l\}$, and tuples. Although parameters $\mathcal{P}$ strongly affect community search results, it is challenging for users to update parameters by themselves during the interactive iterations. Thus, the parameter auto-turning is important for interactive community search. We identify parameters in different categories and then provide the corresponding strategies for interactive-driven auto-tuning.

Auto-tuning rules. In the process of I_{max} interactive actions, users add/delete nodes $Q^{\pm}$ into query $Q_i^{\pm}$ where $0 \le i < I_{max}$. The objective of parameter tuning is to adjust parameter values $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ such that the search result H_{i+1} is close to the optimal community $\hat{H}$. The auto-tuning strategy guideline depends on the current community H_i and the user-provided interactive actions $Q^{\pm}$, which provide the current information of local search area, structural density indexes (e.g., coreness, trussness), and attributes.

Parameter auto-tuning algorithm. Algorithm 6 outlines the procedure of parameter auto-tuning in the i-th iteration. The input includes community model $\mathbb{M}_x$, the current community H_i, parameters $\mathcal{P}_i = \{Q_i, A_i, k_i, S_i\}$, and also a pair of user selected adding/deleting vertices $Q^{\pm}$. The algorithm outputs a modified parameter $\mathcal{P}_{i+1}$. Here, the new query vertices $Q_{i+1}^{\pm} = Q_i^{\pm} \cup Q^{\pm}$. Assume

Algorithm 6 Parameter Auto-Tuning

Input: $\mathbb{M}_x$, H_i, $\mathcal{P}_i = \{Q_i^{\pm}, A_i, k_i, S_i\}$, interaction $Q^{\pm}$, ratio $\lambda \in (0, 1)$
Output: parameter $\mathcal{P}_{i+1} = \{Q_{i+1}^{\pm}, A_{i+1}, k_{i+1}, S_{i+1}\}$

1: Initialization: $\mathcal{P}_{i+1} \leftarrow \mathcal{P}_i$;
2: **if** users click on an inserting action **do**
3: $\quad Q_{i+1}^{+} = Q_i^{+} \cup Q^{+}$;
4: $\quad$ Let k_{Q^+} as the density of Q^+ (e.g., coreness);
5: $\quad$ **if** $k_{Q^+} < k_i$ **do** $k_{i+1} \leftarrow k_{Q^+}$;
6: $\quad$ **else if** $H_i = \emptyset$ and $k_i > 1$ **do** $k_{i+1} \leftarrow k_i - 1$;
7: $\quad$ **else** $S_{i+1} \leftarrow (1 + \lambda) \cdot S_i$;
8: $\quad$ **if** $\text{attr}(Q^+) \not\subseteq A_i$ **do** $A_{i+1} \leftarrow A_i \cup \text{attr}(Q^+)$;
9: **if** users click on a removing action **do**
10: $\quad Q_{i+1}^{-} \leftarrow Q_i^{-} \cup Q^{-}$;
11: $\quad$ **if** $H_i = \emptyset$ and $k_i > 1$ **do** $k_{i+1} \leftarrow k_i - 1$;
12: $\quad$ **else** $S_{i+1} \leftarrow (1 - \lambda) \cdot S_i$;
13: **return** $\mathcal{P}_{i+1} = \{Q_{i+1}^{\pm}, A_{i+1}, k_{i+1}, S_{i+1}\}$;

that the model $\mathbb{M}_x$ is the k-core-based attribute community search model. Then we have A_i as the set of query attributes for some attributed models, k_i as the density constraint of H, i.e., $\text{core}(H) \ge k$, and S_i is the community size constraint of H. Algorithm 6 presents two parameter tuning procedures for the insertion Q^+ action (lines 1-8) and the deleting Q^- action (lines 9-12), respectively. First, it adjusts the important density parameter k_i first to avoid bad community results. A poor setting of parameter k can lead to an empty search result H for two cases. The one is that the query vertex can not meet the density constraint to k_i, which violates the query containment constraints. The other one is that query vertices are disconnected from each other in the search result, violating the connectivity requirement. Thus, if the density of inserted vertex Q^+ is less than k_i, it decreases k_i (lines 4-5). Otherwise, if community H_i is empty, we also decrease k_i by 1 as the updated parameter k_{i+1} (line 6). We slightly adjust the community size and the query attributes via a predefined ratio $\lambda \in (0, 1)$ (lines 7-8). For the deletion case, we remove a community node $Q_{i+1}^{-} = Q_i^{-} \cup Q^{-}$ and update k_{i+1}, S_{i+1} accordingly (lines 9-12). Finally, the algorithm returns the updated parameter $\mathcal{P}_{i+1}$.

6.2 Partial Refinement

Interactive community refinement updates a partial community Δ_H with new parameters $\mathcal{P}_{i+1}$ at a low cost, resulting in $H_{i+1} = H_i + \Delta_H$. However, due to the irregular community shapes and large subgraph search spaces, it is challenging to produce Δ_H efficiently over large graphs. The key idea of our fast community refinement is to leverage local search strategy to construct a high-quality, small candidate subgraph based on H_i and $\mathcal{P}_{i+1}$, and then rerun community search algorithms to get H_{i+1}.

Algorithm. Algorithm 7 provides an efficient approach to update community H_i. First, it initializes a candidate subgraph C_H as the induced subgraph of G by the current community H_i (line 1). Second, it updates C_H by adding potentially qualified neighbors of $u \in V(C_H)$ if density or attribute constraints change in the previous interaction (lines 2-6). For insertion, it also inserts relevant vertices between C_H and the user-clicked vertex Q^+ along a path from Q^+ to C_H (lines 7-9). For deletion, it adjusts C_H by deleting Q^- and its relevant edges (lines 10-11). After that, it computes a refined community H_{i+1} from C_H (line 12). If $H_{i+1} = \emptyset$ for the bad

Algorithm 7 Fast Community Refinement

Input: Model $\mathbb{M}_x$, community H_i, interaction $Q^{\pm}$, parameters $\mathcal{P}_{i+1}$
Output: Community H_{i+1}

1: Construct a candidate subgraph C_H as the induced subgraph of G by previous community H_i;
2: **if** $\mathcal{P}_{i+1} \neq \mathcal{P}_i$ **then**
3: **for** each vertex v in C_H **do**
4: **for** each vertex $u \in N(v)$ **do**
5: **if** u satisfies the new constraint **then**
6: Add u and its incident edges to C_H;
7: **if** users click on an inserting action **then**
8: Find a path P from Q^+ to any vertex in C_H;
9: Reconstruct C_H by inserting all vertices $u \in P$ and u's neighbors;
10: **if** users click on a removing action **then**
11: Delete Q^- and its incident edges from C_H;
12: Run the $\mathbb{M}_x$ community search algorithm to compute H_{i+1} on C_H;
13: **while** $H_{i+1} = \emptyset$ **do**
14: Invoke Algorithm 7 to auto-tune parameters $\mathcal{P}_{i+1}$ and run this refinement algorithm again;
15: **if** $Q^+ \notin V(H_{i+1})$ or $Q^- \in V(H_{i+1})$ **then**
16: Force insert/delete Q^+/Q^- to/from H_{i+1} and maintain the connectivity of H_{i+1} accordingly;
17: **return** community H_{i+1};

parameter setting or strict community model, it auto-tunes parameters $\mathcal{P}_{i+1}$ again by invoking Algorithm 6 and reruns this algorithm; It enforces the insertion/deletion of $Q^{\pm}$ in H_{i+1} and maintains the community connectivity by removing disqualified vertices (lines 15-16). Finally, it returns the updated community H_{i+1}.

6.3 Complexity Analysis

We analyze the complexity of Algorithm 6, Algorithm 7, and our GICS-framework in Algorithm 1. For a given community model $\mathbb{M}_x$, we assume that the original $\mathbb{M}_x$ search algorithm takes $O(\mathcal{T}_x(H))$ time and $O(\mathcal{S}_x(H))$ space, where H is the querying graph.

Theorem 6.1. *The refinement phase with parameter auto-tuning in Algs. 6 and 7 take $O(\mathcal{T}_x(G_H))$ time and $O(\mathcal{S}_x(G_H))$ space.*

Proof. First, Algorithm 6 of parameter auto-tuning takes constant $O(1)$ time. Second, Algorithm 7 first constructs a connected candidate subgraph C_H based on parameters $\mathcal{P}$ and users' interactive actions $Q^{\pm}$, where C_H is a subgraph of G_H, i.e., $C_H \subseteq G_H$. Thus, the community refinement in Algorithm 7 invoking original search algorithms take $O(\mathcal{T}_x(G_H))$ time and $O(\mathcal{S}_x(G_H))$ space.

Theorem 6.2. *Algorithm 1 of* GICS *takes $O(I_{max}(\min\{\bar{m}, r\bar{d}_{max}+\bar{n}\} \cdot \log \bar{n} + \mathcal{T}_x(G_H)))$ time and $O(m + \mathcal{S}_x(G))$ space.*

Proof. We analyze the complexity of our GICS-framework in Algorithm 1, which has three phases of recommendation in Alg. 2, parameter tuning in Alg. 6, and community refinement in Alg. 7. By Theorems 5.1& 6.1, it infers that GICS takes $O(\min\{\bar{m}, r\bar{d}_{max} + \bar{n}\} \cdot \log \bar{n} + \mathcal{T}_x(C_H))$ time in one round of user interaction. Overall, GICS-framework in Algorithm 1 takes $O(I_{max} \cdot (\min\{\bar{m}, r\bar{d}_{max} + \bar{n}\} \cdot \log \bar{n} + \mathcal{T}_x(C_H)))$ time and $O(m + \mathcal{S}_x(G))$ space, where I_{max} is the maximum round of user interactions and $\mathcal{S}_x(G)$ represents the storage of G and the index of community model $\mathbb{M}_x$.

Consider an instance of k-core-based community model $\mathbb{M}_{WCS}$ [55], which takes $O(|Q|m \log n)$ time and $O(m)$ space for a set of query nodes Q on graph G. Thus, our GICS-framework equipped with $\mathbb{M}_{WCS}$ takes $O(I_{max} \cdot (\min\{\bar{m}, r\bar{d}_{max} + \bar{n}\} \cdot \log \bar{n} + |Q|\bar{m} \log \bar{n})) = O((I_{max} + |Q|)\bar{m} \log \bar{n})$ time and $O(m + m) = O(m)$ space.

Table 2: Network Statistics

Datasets	$\|V\|$	$\|E\|$	$core(G)$
Amazon	334,863	925,872	6
DBLP	317,080	1,049,866	113
Youtube	1,134,890	2,987,624	51
LiveJournal	3,997,962	34,681,189	360
Orkut	3,072,441	117,185,083	253

7 Experiments

Datasets and queries. We use five real-world datasets with ground-truth communities from SNAP[1] shown in Table 2. Ground-truth communities are generated from [65], where community members share a common external property or function. For example, the LiveJournal dataset is a free online blogging community with user-defined groups considered as ground-truth communities. For each dataset, we use the recommended top 5,000 high-quality ground-truth communities in experiments. We randomly generate the queries and a few attributes for those vertices without attributes by following the setting of previous works [23, 25, 55]. In each test, we report the average result of 100 queries.

Compared methods. We compare all interactive community search methods, in terms of *recommendation* and *community refinement*. We implement three baseline methods of community models ICS-GNN [23], ACQ [18], WCS [55], and our method as follows.

- GNN-baseline: is an interactive communtiy search method of ICS-GNN that finds a k-sized community with the highest GNN scores [23].
- ACQ-baseline: interactively finds a k-core community for new queries with the largest number of common attributes [18].
- WCS-baseline: interactively finds a k-core community for new queries with the smallest weight [55].
- Ours: is an integrated method by implementing three models ICS-GNN, ACQ, and WCS under the proposed GICS-framework in Algorithm 1, which equips with the recommendation in Algorithm 2 parameter auto-tuning in Algorithm 6, and community refinement in Algorithm 7.

Note that there is no recommendation phase for [23] [18] [55], as their models are either *static without any interaction* or *interactive without any recommendation*. To fit with an interactive setting, all three baselines GNN-baseline, ACQ-baseline, and WCS-baseline are supported with a phase of *random recommendation* by randomly selecting $R^{\pm}$ vertices from candidate subgraphs. For fairness, we also comprehensively test the recommendations in Exp-3 and Exp-5. GNN-baseline is provided with its well-designed scheme of community refinement [23]. For the vertex deletion, ACQ-baseline [18] and WCS-baseline [55] update the original graph by removing vertices and their incident edges and then rerun the search algorithm.

Evaluation metrics. We evaluate competitors on three aspects: *interactive efficiency*, *search quality*, and *recommendation quality*. For algorithm efficiency, we report the average *running time* of interactive search and recommendation, respectively. In terms of community quality, we compare the search results and ground-truth communities using *F1-score*. Recommendation quality is assessed

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

Table 3: Interactive efficiency, search quality and recommendation quality evaluation on three models and five datasets

Datasets			Amazon		DBLP		Youtube		LiveJournal		Orkut	
Models	Phases	Matrics	Baseline	Ours	Baseline	Ours	Baseline	Ours	Baseline	Ours	Baseline	Ours
GNN	Interactive Efficiency	Total Time (s)	0.38	0.88	0.15	0.78	0.22	0.56	1.05	2.89	5.51	10.14
		Interactive Search Time (s)	0.38	0.81	0.15	0.66	0.22	0.28	1.05	2.24	5.51	9.74
		Recommendation Time (s)	\	0.07	\	0.12	\	0.28	\	0.65	\	0.40
	Community Quality	Initial F1-score	0.61	\	0.08	\	0.10	\	0.30	\	0.37	\
		Interactive F1-score	0.67	**0.92**	0.08	**0.20**	0.12	**0.14**	0.31	**0.44**	0.37	**0.65**
	Recommendation Quality	Success Ratio	0.26	**0.72**	0.51	**0.94**	0.61	**0.73**	0.50	**0.82**	0.72	**0.79**
		Accuracy	0.03	**0.29**	0.14	**0.52**	0.24	**0.46**	0.14	**0.38**	0.37	**0.42**
ACQ	Interactive Efficiency	Total Time (s)	0.00	0.03	0.00	0.03	0.00	0.03	0.00	0.03	0.01	0.04
		Interactive Search Time (s)	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.01
		Recommendation Time (s)	\	0.02	\	0.02	\	0.02	\	0.02	\	0.03
	Community Quality	Initial F1-score	0.55	\	0.42	\	0.13	\	0.48	\	0.19	\
		Interactive F1-score	0.59	**0.86**	0.43	**0.76**	0.13	**0.78**	0.50	**0.88**	0.19	**0.38**
	Recommendation Quality	Success Ratio	0.10	**0.26**	0.26	**0.39**	0.13	**0.36**	0.12	**0.27**	0.31	**0.52**
		Accuracy	0.01	**0.15**	0.07	**0.24**	0.02	**0.21**	0.02	**0.16**	0.06	**0.19**
WCS	Interactive Efficiency	Total Time (s)	0.19	0.50	0.42	0.33	8.02	3.93	10.04	8.87	220.76	13.27
		Interactive Search Time (s)	0.19	0.49	0.42	0.33	8.02	3.92	10.04	8.85	220.76	13.26
		Recommendation Time (s)	\	0.00	\	0.00	\	0.01	\	0.02	\	0.01
	Community Quality	Initial F1-score	0.79	\	0.86	\	0.50	\	0.65	\	0.31	\
		Interactive F1-score	0.80	**0.93**	0.86	**0.93**	0.51	**0.68**	0.67	**0.78**	0.31	**0.39**
	Recommendation Quality	Success Ratio	0.24	**0.81**	0.22	**0.68**	0.45	**0.49**	0.40	**0.73**	0.68	**0.91**
		Accuracy	0.14	**0.65**	0.15	**0.48**	0.35	**0.44**	0.24	**0.51**	0.34	**0.52**

through *accuracy* and *success ratio* per iteration. The accuracy of insertions and deletions are defined as: $accuracy^{+} = \frac{|(\hat{H} \setminus H^{*}) \cap R^{+}|}{|R^{+}|}$ and $accuracy^{-} = \frac{|(H^{*} \setminus \hat{H}) \cap R^{-}|}{|R^{-}|}$, where $\hat{H}$ is ground-truth community, H^{*} is search result, R^{+}/R^{-} are recommended vertex sets. A recommendation is considered successful if a user accepts a vertex from $R^{\pm}$, resulting in a success ratio of one; otherwise, it fails with a ratio of zero. We report the average accuracy and success ratio for evaluating recommendation quality.

Parameter setting. By default, we set the maximum interactive iteration as $I_{max} = 10$. For each iteration, we recommend ten vertices for insertion and ten for deletion, i.e., $|R^{+}| = |R^{-}| = 10$. The recommendation candidate subgraph size is 1000. The adjustment ratio is $\lambda = 0.1$ to slightly modify community size. In PCR computation, the damping factor $\alpha = 0.85$, with a constant tolerance of $\delta = 0.0001$ and maximum iterations l_{max} equal to the number of levels in G_D. For a query vertex v, the input parameter $k \in [1, core(v)]$ for ACQ and WCS models. For ICS-GNN, we follow its default setting and set community and subgraph sizes to 100 and 1000, respectively.

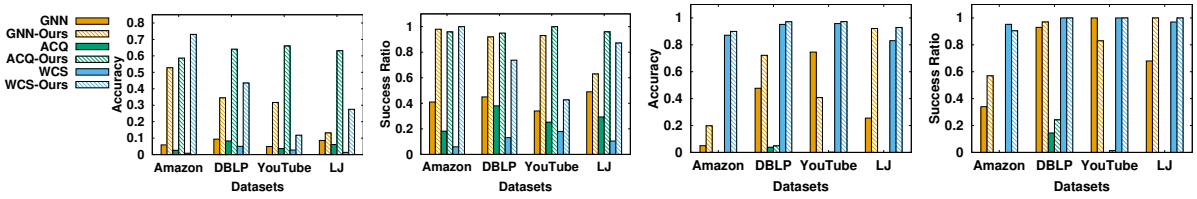
Exp-1: Interactive efficiency evaluation. Table 3 reports the total running time of all methods, and also their breakdown components of interactive search time and recommendation time. Here, the interactive search time includes the parameter auto-tuning and community refinement time, where the community refinement takes the majority cost and depends on its specific community model $\mathbb{M}_x$. Overall, the interactive search time of our GICS is very competitive to baseline methods on all datasets. GICS using the ICS-GNN model takes longer than GNN-baseline, due to an enhanced recommendation scheme for quality improvement. GICS runs faster than WCS-baseline, especially on larger datasets LiveJournal and Orkut, thanks to the effectiveness of our fast partial refinement strategy. In terms of recommendation time, the random recommendation taken by baseline methods is millisecond-scale and omitted from Table 3. The recommendation time of our Algorithm 2 consistently takes less than one second across all models, which is practically useful to allow real-time user interactions.

Exp-2: Search quality evaluation on ground-truth communities. Table 3 reports the F1-score of interactive quality. We compare three baselines and ours. We calculate the initial F1-score of an initial community and the final F1-score of revised communities after ten rounds of interactive iterations, respectively. Our algorithm always achieves a better performance than the baseline on all datasets and all three models, which indicates the effectiveness and scalability of our interactive framework.

Exp-3: Recommendation quality evaluation. Table 3 reports the recommendation quality of all methods, in terms of the accuracy and success ratio. We calculate the average result of inserting and deleting recommendation quality. The results show that our PADR always achieves a better quality than the baseline random recommendation. Figure 5 evaluates the recommendation quality of the first round recommendation for insertion and deletion separately. As shown in Figure 5, our PADR for insertion consistently outperforms than baselines for both accuracy and success ratio. Moreover, our PADR recommendation can achieve success around two times of baselines in Figure 5(b). Figure 5(c) and 5(d) show that the recommendation quality of baselines and ours are similar, as a small size of candidate search subgraph.

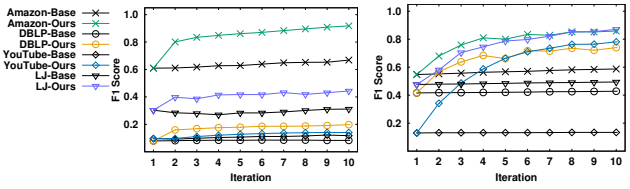
Exp-4: Community quality evaluation by varying interactions. Figure 6 reports the F1-score of interactive community search results for GNN and ACQ community models on four datasets, respectively. We vary the number of iterations to test community quality. With the increasing number of iterations, most methods improve better and better, indicating the usefulness of user interactions. We can see that all our methods achieve higher F1-scores than baseline methods on both models. Our GICS method can obtain good F1-scores significantly in the first three iterations, reflecting the effective refinement of our techniques.

Exp-5: Ablation study. We test the performance of three phases in GICS, i.e., Phase-I of recommendation, Phase-II of parameter auto-tuning, and Phase-III of partial refinement, which are denoted by *P-I*, *P-II*, *P-III*, respectively. We implement GICS on WCS model, and compare GICS with its four variants by removing *P-I*, *P-II*, *P-III*, and *P-II&III*, respectively. WCS-baseline is implemented by rerunning original WCS model algorithms [55] and applying random recommendations in each iteration. Table 4 illustrates that GICS achieves the highest F1-scores and decreases with smaller F1-scores with the removal of any phase. In terms of efficiency, GICS becomes much slower when Phase-III of partial refinement is omitted, as shown by comparing the running time of GICS and GICS *without P-III*. Moreover, comparing WCS-baseline to GICS *without P-II &*



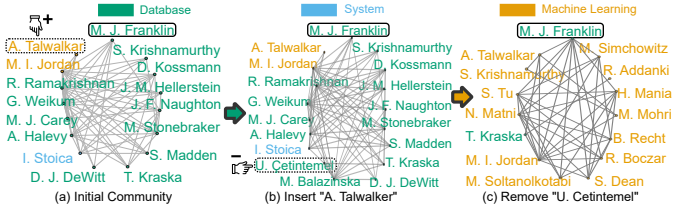
(a) Insertion Accuracy (b) Insertion Success Ratio (c) Deletion Accuracy (d) Deletion Success Ratio

Figure 5: Recommendation quality evaluation on four datasets and three models.



(a) GNN (b) ACQ

Figure 6: F1-score evaluation by varying l iterations.



(a) Initial Community (b) Insert "A. Talwalker" (c) Remove "U. Cetintemel"

Figure 7: Case study on DBLP network.

P-III demonstrates that even added Phase-I personalized recommendation alone outperforms WCS-baseline in F1-scores. As a result, it clearly demonstrates the usefulness and indispensability of all three proposed phases in GICS.

Exp-6: User study on DBLP network. We conduct a user study on DBLP network to evaluate user satisfaction with interactive search through three questions: Q1 regarding response time, Q2 about final result, and Q3 concerning insertion/deletion recommendation. We test on two models ICS-GNN and ACQ by comparing the baseline and the GICS method. We collect 50 sample results by inviting 25 participants to test GNN model and another 25 for ACQ model. To maintain the integrity of the results, participants were not informed about the specific algorithms utilized, ensuring their responses remained unbiased. The statistical analysis results, as summarized in Table 5, indicate that GICS achieves high satisfaction rates across all questions on both models. The T-test results suggest that there is a slight difference between the satisfaction of final results and recommendation. The correlation between final results and recommendation is strong, indicating a better recommendation result is crucial.

Exp-7: Case study of personalized academic group search. We use a research collaboration network from Aminer [12] with 144,334 vertices, 1,821,930 edges, and 7 vertex labels like research topics of published papers, e.g., "DB", "ML". Our case study aims to identify a notable machine learning ("ML") research group formed by "M. J. Franklin", a famous database researcher. To find who collaborated with "M. J. Franklin" in ML area, we intend to use our interactive community search to find this personalized answer in Figure 7, where colors indicate different research topics. First, we

Table 4: Ablation study of three phases of GICS on WCS model: P-I for recommendation, P-II for parameter auto-tuning, and P-III for partial refinement.

	Datasets	GICS	GICS without P-I	GICS without P-II	GICS without P-III	GICS without P-II&P-III	WCS-baseline
Time(s)	Amazon	0.50	0.09	0.63	0.50	3.28	0.19
	DBLP	0.33	0.11	0.27	0.69	1.32	0.42
	Youtube	3.93	0.56	0.50	11.37	18.93	8.02
	LiveJournal	8.87	4.58	5.21	15.16	24.89	10.04
F1-score	Amazon	**0.93**	0.82	0.92	0.90	0.83	0.80
	DBLP	**0.93**	0.89	0.92	0.9	0.90	0.86
	Youtube	**0.68**	0.67	0.57	0.52	0.52	0.51
	LiveJournal	**0.78**	0.69	0.76	0.72	0.71	0.67

Table 5: Statistical analysis of user study on DBLP.

Models and Methods	GNN			GNN-Ours			ACQ			ACQ-Ours		
Questions	Q1	Q2	Q3	Q1	Q2	Q3	Q1	Q2	Q3	Q1	Q2	Q3
Count Satisfied	25	4	2	24	22	22	23	2	0	25	20	23
Count Dissatisfied	0	21	23	1	3	3	2	23	25	0	5	2
T-test (Q2&Q3)	\	0.39		\	1		\	0.16		\	0.23	
Correlation (Q2&Q3)	\	0.68		\	1		\	\		\	0.59	
Average Satisfaction	**100%**	16%	8%	96%	**88%**	**88%**	92%	8%	0%	**100%**	**80%**	**92%**
Overall Satisfaction	8%			**84%**			0%			**80%**		

query "M. J. Franklin" and obtain a database-focused community (Figure 7(a)), but involving very few ML researchers. By adding ML researcher "A. Talwalker", we achieve a mixed collaboration group containing both ML and DB researchers in Figure 7(b). After removing "U. Cetintemel", our interactive system finally refines the result to ML collaboration group in Figure 7(c). This case study illustrates the usefulness of our interactive community search for excellent recommendations and user-friendly interaction.

8 Conclusions

This paper proposes GICS for interactive community search, enabling users to refine community results by adding/ deleting vertices in a user-friendly way. GICS can be extended to support existing query-oriented community search models in the interactive setting but cannot support a broad range of graph clustering tasks that find numerous communities globally without input queries. Equipped with three well-designed components of personalized recommendation, fast refinement, and parameter auto-turning, GICS improves the quality of community answers efficiently against the state-of-the-art methods on ground-truth datasets. User studies and case studies validate the efficiency, effectiveness, and usability of proposed GICS.

Acknowledgments

This work was supported by Hong Kong RGC Projects Nos. 12200424, 12202221,12202024, C2003-23Y, RIF R1015-23, and Guangdong Basic and Applied Basic Research Foundation (Project No. 2023B1515130002). Xin Huang is the corresponding author.

References

[1] Esra Akbas and Peixiang Zhao. 2017. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB* 10, 11 (2017), 1298–1309.

[2] Ahmed Al-Baghdadi and Xiang Lian. 2020. Topic-based community search over spatial-social networks. *PVLDB* 13, 12 (2020), 2104–2117.

[3] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *DMKD* 29, 5 (2015), 1406–1433.

[4] Ali Behrouz, Farnoosh Hashemi, and Laks VS Lakshmanan. 2022. FirmTruss Community Search in Multilayer Networks. *PVLDB* 16, 3 (2022), 505–518.

[5] Jiazun Chen, Yikuan Xia, and Jun Gao. 2023. CommunityAF: An Example-Based Community Search Method via Autoregressive Flow. *PVLDB* 16, 10 (2023), 2565–2577.

[6] Lu Chen, Chengfei Liu, Kewen Liao, Jianxin Li, and Rui Zhou. 2019. Contextual community search over large social networks. In *ICDE*. IEEE, 88–99.

[7] Yankai Chen, Jie Zhang, Yixiang Fang, Xin Cao, and Irwin King. 2021. Efficient community search over large directed graphs: An augmented index-based approach. In *IJCAI*. 3544–3550.

[8] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *SIGMOD*. 277–288.

[9] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *SIGMOD*. 991–1002.

[10] Yizhou Dai, Miao Qiao, and Lijun Chang. 2022. Anchored densest subgraph. In *SIGMOD*. 1200–1213.

[11] Antonio Del Sol, Hirotomo Fujihashi, and Paul O'Meara. 2005. Topology of small-world networks of protein–protein complex structures. *Bioinformatics* 21, 8 (2005), 1311–1315.

[12] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *PVLDB* 14, 11 (2021), 2006–2018.

[13] Boxin Du, Si Zhang, Nan Cao, and Hanghang Tong. 2017. First: Fast interactive attributed subgraph matching. In *SIGKDD*. 1447–1456.

[14] Wenfei Fan, Tao He, Longbin Lai, Xue Li, Yong Li, Zhao Li, Zhengping Qian, Chao Tian, Lei Wang, Jingbo Xu, et al. 2021. GraphScope: a unified engine for big graph processing. *PVLDB* 14, 12 (2021), 2879–2892.

[15] Wenfei Fan and Chao Tian. 2022. Incremental graph computations: Doable and undoable. *TODS* 47, 2 (2022), 1–44.

[16] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Incremental graph pattern matching. *TODS* 38, 3 (2013), 1–47.

[17] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *PVLDB* 10, 6 (2017), 709–720.

[18] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *PVLDB* 9, 12 (2016), 1233–1244.

[19] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDBJ* 29, 1 (2020), 353–392.

[20] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2018. Effective and efficient community search over large directed graphs. *TKDE* 31, 11 (2018), 2093–2107.

[21] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *PVLDB* 13, 6 (2020), 854–867.

[22] Denis Gallo, Matteo Lissandrini, Yannis Velegrakis, et al. 2020. Personalized page rank on knowledge graphs: Particle Filtering is all you need!. In *EDBT*. 447–450.

[23] Jun Gao, Jiazun Chen, Zhao Li, and Ji Zhang. 2021. ICS-GNN: lightweight interactive community search via graph neural network. *PVLDB* 14, 6 (2021), 1006–1018.

[24] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322.

[25] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *PVLDB* 10, 9 (2017), 949–960.

[26] Xin Huang, Laks VS Lakshmanan, Jianliang Xu, and HV Jagadish. 2019. *Community search over big graphs*. Vol. 14. Springer.

[27] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *PVLDB* 9, 4 (2015), 276–287.

[28] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *PVLDB* 13, 10 (2020), 1723–1736.

[29] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective community search over large star-schema heterogeneous information networks. *PVLDB* 15, 11 (2022), 2307–2320.

[30] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang. 2022. Query driven-graph neural networks for community search: from non-attributed, attributed, to interactive attributed. *PVLDB* 15, 6 (2022), 1243–1255.

[31] Junghoon Kim, Tao Guo, Kaiyu Feng, Gao Cong, Arijit Khan, and Farhana M Choudhury. 2020. Densely connected user community and location cluster search in location-based social networks. In *SIGMOD*. 2199–2209.

[32] Junghoon Kim, Siqiang Luo, Gao Cong, and Wenyuan Yu. 2022. DMCS: Density modularity based community search. In *SIGMOD*. 889–903.

[33] Ling Li, Siqiang Luo, Yuhai Zhao, Caihua Shan, Zhengkui Wang, and Lu Qin. 2023. COCLEP: Contrastive Learning-based Semi-Supervised Community Search. In *ICDE*. IEEE, 2483–2495.

[34] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential community search in large networks. *PVLDB* 8, 5 (2015), 509–520.

[35] Wenqing Lin. 2019. Distributed Algorithms for Fully Personalized PageRank on Large Graphs. In *WWW*. 1084–1094.

[36] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *SIGMOD*. 2183–2197.

[37] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: vertex-centric attributed community search. In *ICDE*. IEEE, 937–948.

[38] Dongsheng Luo, Yuchen Bian, Yaowei Yan, Xiao Liu, Jun Huan, and Xiang Zhang. 2020. Local community detection in multiple networks. In *KDD*. 266–274.

[39] Jiehuan Luo, Xin Cao, Xike Xie, and Qiang Qu. 2019. Best Co-Located Community Search in Attributed Networks. In *SIGKDD*. 2453–2456.

[40] Jiehuan Luo, Xin Cao, Xike Xie, Qiang Qu, Zhiqiang Xu, and Christian S Jensen. 2020. Efficient Attribute-Constrained Co-Located Community Search. In *ICDE*. IEEE, 1201–1212.

[41] Wensheng Luo, Xu Zhou, Kenli Li, Yunjun Gao, and Keqin Li. 2021. Efficient influential community search in large uncertain graphs. *TKDE* 35, 4 (2021), 3779–3793.

[42] Kasper Overgaard Mortensen, Fatemeh Zardbani, Mohammad Ahsanul Haque, Steinn Ymir Agustsson, Davide Mottin, Philip Hofmann, and Panagiotis Karras. 2023. Marigold: Efficient k-Means Clustering in High Dimensions. *PVLDB* 16, 7 (2023), 1740–1748.

[43] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, Themis Palpanas, et al. 2017. New trends on exploratory methods for data analytics. *PVLDB* 10, 12 (2017), 1977–1980.

[44] Davide Mottin and Emmanuel Müller. 2017. Graph exploration: From users to large graphs. In *SIGMOD*. 1737–1740.

[45] Davide Mottin and Emmanuel Müller. 2018. Graph Exploration: Let me Show what is Relevant in your Graph. *SIGKDD* (2018).

[46] Mark EJ Newman. 2001. Scientific collaboration networks. I. Network construction and fundamental results. *Physical review E* 64, 1 (2001), 016131.

[47] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *PVLDB* 6, 14 (2013), 1870–1881.

[48] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused clustering and outlier detection in large attributed graphs. In *SIGKDD*. 1346–1355.

[49] Xueming Qian, He Feng, Guoshuai Zhao, and Tao Mei. 2013. Personalized recommendation combining user interest and social circle. *TKDE* 26, 7 (2013), 1763–1777.

[50] José F Rodrigues Jr, Hanghang Tong, Agma JM Traina, Christos Faloutsos, and Jure Leskovec. 2006. GMine: a system for scalable, interactive graph visualization and mining. In *VLDB*. 1195–1198.

[51] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2015. The minimum wiener connector problem. In *SIGMOD*. 1587–1602.

[52] Tara Safavi, Caleb Belth, Lukas Faber, Davide Mottin, Emmanuel Müller, and Danai Koutra. 2019. Personalized knowledge graph summarization: From the cloud to your pocket. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 528–537.

[53] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Real-time Top-k Personalized PageRank over Large Graphs on GPUs. *PVLDB* 13, 1 (2019), 15–28.

[54] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*. 939–948.

[55] Longxu Sun, Xin Huang, Ronghua Li, Byron Choi, and Jianliang Xu. 2020. Index-based Intimate-Core Community Search in Large Weighted Graphs. *TKDE* 34, 9 (2020), 4313–4327.

[56] Longxu Sun, Xin Huang, Zheng Wu, and Jianliang Xu. 2024. Efficient Cross-layer Community Search in Large Multilayer Graphs. In *ICDE*. IEEE, 2959–2971.

[57] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jiajie Xu, and Chengfei Liu. 2022. Reliable community search in dynamic networks. *PVLDB* 15, 11 (2022), 2826–2838.

[58] Yufei Tao, Yuanbing Li, and Guoliang Li. 2019. Interactive graph search. In *SIGMOD*. 1393–1410.

[59] Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. 2011. Community discovery using nonnegative matrix factorization. *DMKD* 22, 3 (2011), 493–521.

[60] Jia Wang, Ada Wai-Chee Fu, and James Cheng. 2014. Rectangle counting in large bipartite graphs. In *IEEE BigData*. 17–24.

[61] Runhui Wang, Sibo Wang, and Xiaofang Zhou. 2019. Parallelizing approximate single-source personalized pagerank queries on shared memory. *VLDBJ* 28, 6 (2019), 923–940.

[62] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient algorithms for approximate single-source personalized pagerank queries. *TODS* 44, 4 (2019), 1–37.

[63] Joyce Jiyoung Whang, David F Gleich, and Inderjit S Dhillon. 2013. Overlapping community detection using seed set expansion. In *CIKM*. 2099–2108.

[64] Tianyang Xu, Zhao Lu, and Yuanyuan Zhu. 2022. Efficient Triangle-Connected Truss Community Search in Dynamic Graphs. *PVLDB* 16, 3 (2022), 519–531.

[65] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *SIGKDD*. 1–8.

[66] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *ICDM*. IEEE, 1151–1156.

[67] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. 2024. Efficient Algorithms for Personalized PageRank Computation: A Survey. *TKDE* 36, 9 (2024), 4582–4602.

[68] Kai Yao and Lijun Chang. 2021. Efficient Size-Bounded Community Search over Large Networks. *PVLDB* 14, 8 (2021), 1441–1453.

[69] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2016. I/O efficient ECC graph decomposition via graph reduction. *PVLDB* 9, 7 (2016), 516–527.

[70] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *TKDE* 30, 5 (2017), 922–935.

[71] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized pagerank on dynamic graphs. In *SIGKDD*. 1315–1324.

[72] Zhiwei Zhang, Xin Huang, Jianliang Xu, Byron Choi, and Zechao Shang. 2019. Keyword-centric community search. In *ICDE*. IEEE, 422–433.

[73] Dong Zheng, Jianquan Liu, Rong-Hua Li, Cigdem Aslay, Yi-Cheng Chen, and Xin Huang. 2017. Querying intimate-core groups in weighted graphs. In *IEEE ICSC*. IEEE, 156–163.

[74] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential Community Search over Large Heterogeneous Information Networks. *PVLDB* 16, 8 (2023), 2047–2060.

[75] Xuliang Zhu, Xin Huang, Byron Choi, and Jianliang Xu. 2020. Top-k graph summarization on hierarchical DAGs. In *CIKM*. 1903–1912.

[76] Yuanyuan Zhu, Jian He, Junhao Ye, Lu Qin, Xin Huang, and Jeffrey Xu Yu. 2020. When Structure Meets Keywords: Cohesive Attributed Community Search. In *CIKM*. 1913–1922.