An Optimal and Progressive Approach to Online Search of Top-K Influential Communities

Fei Bi[†], Lijun Chang[§], Xuemin Lin[†], Wenjie Zhang[†]
[†] University of New South Wales, Australia [§] The University of Sydney, Australia
f.bi@student.unsw.edu.au, Lijun.Chang@sydney.edu.au
{lxue,zhangw}@cse.unsw.edu.au

ABSTRACT

Community search over large graphs is a fundamental problem in graph analysis. Recent studies propose to compute top-k influential communities, where each reported community not only is a cohesive subgraph but also has a high influence value. The existing approaches to the problem of top-k influential community search can be categorized as index-based algorithms and online search algorithms without indexes. The index-based algorithms, although being very efficient in conducting community searches, need to pre-compute a special-purpose index and only work for one built-in vertex weight vector. In this paper, we investigate online search approaches and propose an *instance-optimal* algorithm LocalSearch whose time complexity is linearly proportional to the size of the smallest subgraph that a correct algorithm needs to access without indexes. In addition, we also propose techniques to make LocalSearch *progressively* compute and report the communities in decreasing influence value order such that k does not need to be specified. Moreover, we extend our framework to the general case of top-k influential community search regarding other cohesiveness measures. Extensive empirical studies on real graphs demonstrate that our algorithms outperform the existing online search algorithms by several orders of magnitude.

PVLDB Reference Format:
Fei Bi, Lijun Chang, Xuemin Lin, Wenjie Zhang. An Optimal and Progressive Approach to Online Search of Top-K Influential Communities. *PVLDB*, 11 (9): 1056-1068, 2018.
DOI: https://doi.org/10.14778/3213880.3213881

1. INTRODUCTION

Community search is a fundamental problem in graph analysis, and has been receiving increasing interest in recent years (see a recent tutorial in [23] and references therein). Existing works on community search mainly focus on the cohesiveness of structural connections among members of a community while ignoring other aspects of communities, *e.g.*, influence. As a result, an enormous number of overlapping communities may be reported, and also a single community can be of a large size. However, in many application domains, we are usually only interested in the most influential communities [27]. Motivated by this, top-k influential community search is recently proposed and studied in [10, 27, 28]. It has many important applications such as extracting backbone structures (*i.e.*, being both cohesive and influential) from biology networks [5], and detecting cohesive communities consisting of influential people in social networks [26]. Moreover, it can also be used to identify the most influential community in a collaboration network of database researchers for organizing a workshop [27], and to compute the most influential community in a friendship network of hikers to plan a hiking event [28]. Besides, computing influential communities also greatly refines communities to their core members [27].

Here, the graph $G = (V, E)$ is associated with a vertex weight vector $\omega(\cdot)$ assigning an influence value to every vertex in V. Each community of G, called *influential γ-community*, besides being a cohesive subgraph (*i.e.*, with minimum degree at least γ), has *an influence value that equals the minimum vertex weight of the community* [27]. As a result, members in a high influential γ-community are highly connected to each other, and moreover each member is also an influential individual. Formally speaking, a connected subgraph g of G is an influential γ-community [27] if 1) its minimum vertex degree is at least γ, and 2) it is the maximal one among all such subgraphs of G with the same influence value as g. For example, consider the graph in Figure 1 where vertex weights are shown beside the vertices, and $\gamma = 3$. There are two influential γ-communities: the subgraphs induced by vertices $\{v_0, v_1, v_5, v_6\}$ and vertices $\{v_3, v_4, v_7, v_8, v_9\}$ that, respectively, have influence values 10 and 13. The subgraph induced by vertices $\{v_3, v_4, v_7, v_8\}$ also has an influence value 13; however, it is not an influential γ-community since it is not maximal. *The problem of top-k influential community search is to compute the k influential γ-communities of a graph with the highest influence values, for a user-specified query consisting of γ and k.*

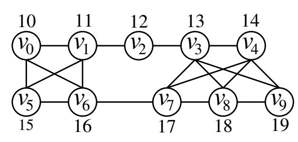


Figure 1: An example graph

Existing Approaches and Their Deficiencies. The existing approaches to top-k influential community search can be categorized as *index-based algorithms* and *online search algorithms*.

(1) Index-based Algorithms. Li et al. [27] proposed an index-based algorithm IndexAll to efficiently retrieve the top-k influential γ-communities from a pre-built special-purpose index that essentially materializes all influential γ-communities of a graph in a compact form for all possible γ values. However, the special-purpose index adds a large burden to the graph processing system, as it is time-consuming to update the index when the graph changes. Moreover,

Proceedings of the VLDB Endowment, Vol. 11, No. 9

DOI: https://doi.org/10.14778/3213880.3213881

IndexAll cannot process queries that impose vertex weight vectors different from the one used in the index.

(2) Online Search Algorithms. Online search algorithms without pre-computing indexes are investigated in [10, 27]. Firstly, Li et al. [27] proposed an OnlineAll algorithm, which online computes all influential γ-communities in a graph in increasing influence value order. OnlineAll iteratively applies the following three subroutines: 1) reduce the current graph to its γ-core (*i.e.*, maximal subgraph with minimum degree at least γ); 2) identify the connected component of the resulting graph containing the vertex with the minimum weight, which is the next influential γ-community; and 3) remove the minimum-weight vertex from the graph. During this process, the last k identified influential γ-communities are the results. Among the above three subroutines of OnlineAll, the second one is the most time-consuming due to the overlapping nature of the influential γ-communities [10]. In view of this, Chen et al. [10] proposed a Forward algorithm which conducts the second subroutine of OnlineAll (*i.e.*, connected component computation) only for the last k iterations; as a result, Forward improves upon OnlineAll. Nevertheless, both OnlineAll and Forward are global search algorithms that need to traverse the entire graph for finding just the top-k influential γ-communities.

Challenges and Our Online Local Search Approach. In this paper, we aim to compute the top-k influential γ-communities by conducting a *local search* on the graph G *without pre-computing indexes*, to overcome the deficiencies of the existing algorithms. The benefits of local search without indexes are two-fold.

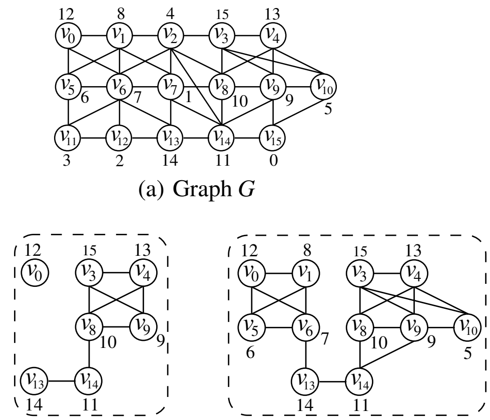
- It does not incur any burden to the graph data management system, regarding index construction and index maintenance.
- It can efficiently process a query by visiting only a small portion of the graph G.

However, there are three challenges to tackle to achieve this.

- It is challenging to determine whether a given subgraph of G is sufficient for processing a query.
- It is challenging to choose a proper subgraph to process.
- It is challenging to carry out the ideas efficiently for real-time query processing over large graphs.

Note that, the Backward algorithm proposed in [10] tried to conduct a local search for computing top-k influential γ-communities, but it fails by having a quadratic time complexity and is outperformed by Forward when γ is large [10].

We propose a local search framework to tackle the above challenges, based on the following ideas. Firstly, we prove that *if the subgraph $G_{\geq\tau}$ of G contains at least k influential γ-communities, then the top-k influential γ-communities in $G_{\geq\tau}$ is the query result*, where $G_{\geq\tau}$ denotes the subgraph of G induced by all vertices with weights at least τ. Thus, our goal is to find the smallest subgraph $G_{\geq\tau^*}$ of G containing at least k influential γ-communities. Secondly, we prove that *the number of influential γ-communities in a subgraph $G_{\geq\tau}$ of G is non-decreasing when τ decreases*. Thus, we can find the target subgraph $G_{\geq\tau^*}$ by iteratively decreasing the value of τ until reaching the target value. Thirdly, to efficiently implement the above ideas, we propose to only process (*i.e.*, count the number of influential γ-communities for) the subgraphs $G_{\geq\tau_1}, G_{\geq\tau_2}$, ..., such that the size of $G_{\geq\tau_i}$ is around twice the size of $G_{\geq\tau_{i-1}}$ for every $i > 1$. For example, to compute the top-2 influential γ-communities in the graph in Figure 2(a) with $\gamma = 3$, we first count the number of influential γ-communities in the subgraph $G_{\geq 9}$ as shown in Figure 2(b), which is 1. Thus, we need to find another smaller τ such that the size of $G_{\geq\tau}$ is around twice the size of $G_{\geq 9}$; we obtain $\tau_2 = 5$ and $G_{\geq 5}$ is shown in Figure 2(c). As there are three influential γ-communities in $G_{\geq 5}$ — the subgraphs induced by vertices $\{v_0, v_1, v_5, v_6\}$, $\{v_3, v_4, v_8, v_9\}$ and $\{v_3, v_4, v_8, v_9, v_{10}\}$, respectively — the top-2 are the result.



(a) Graph G (b) Subgraph $G_{\geq 9}$ (c) Subgraph $G_{\geq 5}$

Figure 2: An example of our local search framework

As a critical subroutine in our local search framework, we propose a linear-time algorithm to count the number of influential γ-communities in an arbitrary given subgraph of the graph G. As a result, we prove that the time complexity of our local search algorithm LocalSearch is linear to the size of the largest subgraph that it accesses. We also show that the subgraph that LocalSearch accesses is at most a constant (specifically, 3) times larger than the smallest subgraph $G_{\geq\tau^*}$ that an online search algorithm without indexes needs to access for correctly computing the top-k influential γ-communities. Thus, LocalSearch is instance-optimal among the class of online search algorithms without indexes.

Moreover, we propose techniques to make LocalSearch progressively compute and report the influential γ-communities in decreasing influence value order such that k does not need to be specified in the query. The user can terminate the algorithm at any time once determining that enough influential γ-communities have been reported. Our instance-optimality result of LocalSearch also carries over to the progressive approach. It is worth noting that the existing global search algorithms OnlineAll and Forward are only able to report the k communities at the end of the algorithm.

Finally, we also extend our local search framework to the case of non-containment community search and to the case of top-k influential community search regarding other cohesiveness measures.

Contributions. Our main contributions are summarized as follows.

- We propose an instance-optimal algorithm LocalSearch, whose time complexity is linearly proportional to the size of the smallest subgraph that a correct algorithm without indexes needs to access, for computing the top-k influential γ-communities (Section 3).
- We propose techniques to make LocalSearch progressively compute and report the influential γ-communities in decreasing influence value order (Section 4).
- We extend our local search framework to the general case of top-k influential community search regarding other cohesiveness measures (Section 5).

Extensive experimental results in Section 6 show that our local search algorithms outperform the existing online search algorithms by several orders of magnitude. Some of the proofs are omitted from this paper due to limit of space, and can be found in the full version [3].

Related Works. Besides top-k influential community search as discussed above, other related works are categorized as follows.

(1) Community Detection. Community detection is a long-studied problem [6, 16], which aims to find all communities in a graph for a given community definition. A community is a group of vertices that are similar to each other and dissimilar to vertices outside the community. The existing community definitions can be categorized as, *(1) graph partitioning* that divides the vertices of a graph into k groups of predefined size such that the number of inter-group edges is minimized [2, 25, 38], *(2) hierarchical clustering* that reveals the multi-level structure of the graph by computing the similarity for each pair of vertices [19, 29], *(3) partitional clustering* that divides vertices into k clusters such that the cost function defined on distances/disimilarities between vertices is minimized [20, 31, 34], and *(4) spectral clustering* that partitions the graph by using the eigenvectors of the matrix derived from the graph [15, 30, 36]. Due to inherent problem natures, these techniques cannot be used to compute top-k influential communities studied in this paper.

(2) Cohesive Subgraph Computation. Computing cohesive subgraphs in a graph has been extensively studied in [8, 9, 12, 18, 32, 33, 35, 40], where a cohesive subgraph can be regarded as a community. The cohesiveness of a graph is measured by the minimum degree (aka, k-core) [33, 35], the average degree (aka, edge density) [9, 18], the minimum number of triangles each edge participates in (aka, k-truss) [12, 32], or the edge connectivity (aka, k-edge connected components) [8, 40]. These works focus on computing all maximal subgraphs whose cohesiveness is no smaller than a user-given threshold. Due to different problem definitions, these techniques cannot be applied to the problem studied in this paper.

(3) Community Search. Recently, cohesive community search is receiving increasing interests (see [23] and references therein). Given one query vertex or a set of query vertices, cohesive community search is to find a subgraph such that (1) it contains all query vertices and (2) its cohesiveness is no smaller than the user given threshold. For example, k-core-based community search is studied in [1, 17, 37], edge density-based community search is studied in [39], k-truss-based community search is studied in [22, 24], and edge connectivity-based community search is studied in [7, 21]. As influences of vertices are not considered in these works, these techniques cannot be applied to the problem of top-k influential community search.

2. PRELIMINARIES

In this paper, we focus on a *vertex-weighted undirected graph* $G = (V, E, \omega)$, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, and ω is a weight vector that assigns each vertex $u \in V$ a weight denoted by $\omega(u)$. Here, the weight $\omega(u)$ represents the *influence* of vertex u, which can be its PageRank value, centrality score, h-index, social status, and etc; the larger the value, the more influential the vertex is. Following the existing works [10, 27], we assume that the weights of vertices are pre-given[1], and each vertex has a distinct weight (*i.e.*, $\omega(u) \neq \omega(v), \forall u \neq v$). For a given value τ, we use $V_{\geq\tau}$ to denote the subset of V consisting of all vertices with weights no less than τ (*i.e.*, $V_{\geq\tau} = \{u \in V \mid \omega(u) \geq \tau\}$). In the following, for ease of presentation we simply refer to a vertex-weighted undirected graph as a graph when the context is clear.

We denote the size of a graph G by $\texttt{size}(G)$, which is the summation of the number of vertices and the number of edges in G; that is, $\texttt{size}(G) = |V| + |E|$. The set of neighbors of $u \in V$ in G is denoted by $N(u) = \{v \in V \mid (u, v) \in E\}$, and the degree of u is denoted by $d(u)$, which is the number of neighbors of u (*i.e.*, $d(u) = |N(u)|$). Given a subset $S \subseteq V$ of vertices, the subgraph of G induced by S is denoted by $G[S]$, which consists of all edges of G whose both end-points are in S; that is, $G[S] = (S, \{(u, v) \in E \mid u, v \in S\}, \omega)$. For presentation simplicity, we use $G_{\geq\tau}$ to denote the subgraph of G induced by vertices $V_{\geq\tau}$ (*i.e.*, $G_{\geq\tau} = G[V_{\geq\tau}]$).

[1]Note that, the techniques proposed in this paper can be extended to the case that the weights of vertices are computed online based on a query, *e.g.*, the weight of a vertex is the reciprocal of the shortest distance to query vertices as studied in closest community search [24]. We will analyze the time complexities of such extensions in our future work.

Influential Community. This paper aims to identify influential communities from a given large graph G, where each community is a cohesive subgraph of G and has an influence value. The influence value of a subgraph is defined in below, which is shown to be robust to outliers as discussed in [27].

Definition 2.1: [27] Given a subgraph $g = (V(g), E(g), \omega)$ of G, the ***influence value*** of g, denoted by $f(g)$, is defined as the minimum weight of the vertices in g (*i.e.*, $f(g) = \min_{u \in V(g)} \omega(u)$).

For the cohesiveness measure, many definitions have been proposed and studied in the literature, *e.g.*, k-core [33, 35], edge density [9, 18], k-truss [12, 32], edge connectivity [8, 11, 40]. Among them, the k-core-based cohesiveness measure has been widely adopted, due to its simplicity and fast computability. Thus, we mainly focus on k-core-based community search in the following, and will extend our techniques to other cohesiveness measures in Section 5.

Definition 2.2: [27] Given a graph G and an integer γ, an ***influential γ-community*** is a *vertex-induced subgraph* g of G such that the following constraints are satisfied.

- ***Connected:*** g is a connected subgraph;
- ***Cohesive:*** each vertex u in g has a degree at least γ, *i.e.*, the minimum degree of g is at least γ;
- ***Maximal:*** there exists no other subgraph g' of G such that (1) g' is a supergraph of g with $f(g') = f(g)$, and (2) g' is also connected and cohesive.

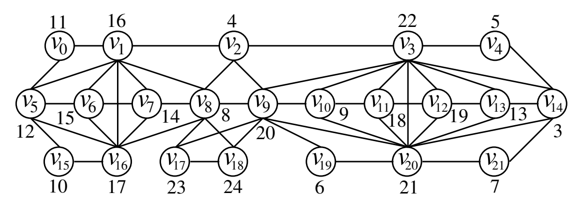


Figure 3: A graph

Example 2.1: Consider the graph in Figure 3 and $\gamma = 3$. The subgraph g_1 induced by vertices $\{v_3, v_{10}, v_{11}, v_{12}, v_{20}\}$ is connected, and has a minimum degree 3 and an influence value 9. However, it is not an influential γ-community because it is not maximal; that is, the subgraph g_2 induced by vertices $\{v_3, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{20}\}$ is an influential γ-community with the same influence value as g_1. Note that, the subgraph induced by vertices $\{v_3, v_{11}, v_{12}, v_{20}\}$ is also an influential γ-community; this is because, although it is a subgraph of g_2, it has a larger influence value (*i.e.*, 18) than g_2. □

In the following, for presentation simplicity, we simply refer to an influential γ-community by the set of vertices from which the influential γ-community is induced.

Problem Statement. Given a graph $G = (V, E, \omega)$, and two query parameters γ and k, the problem of top-k influential community search is to extract the k influential γ-communities with the highest influence values from G.

For example, consider the graph in Figure 3 with $\gamma = 3$ and $k = 4$. The top-4 influential γ-communities are $\{v_3, v_{11}, v_{12}, v_{20}\}$, $\{v_1, v_6, v_7, v_{16}\}$, $\{v_3, v_{11}, v_{12}, v_{13}, v_{20}\}$ and $\{v_1, v_5, v_6, v_7, v_{16}\}$ with influence values 18, 14, 13 and 12, respectively.

3. A LOCAL SEARCH APPROACH

In the following, we first develop a local search framework for efficient top-k influential community search in Section 3.1, and then present our approach in Section 3.2, while the instance-optimality of our local search approach is illustrated in Section 3.3.

3.1 The Framework

Properties of Influential γ-community. Firstly, we prove some important properties of influential γ-community in the following lemmas and theorems.

LEMMA 3.1: *For any two values $\tau_1 \leq \tau_2$, every influential γ-community in $G_{\geq\tau_2}$ is also an influential γ-community in $G_{\geq\tau_1}$. Note that, $G_{\geq\tau_1}$ is a supergraph of $G_{\geq\tau_2}$.*

LEMMA 3.2: *For any two values $\tau_1 \leq \tau_2$ and an influential γ-community g in $G_{\geq\tau_1}$, if the influence value of g is no smaller than τ_2, then g is also an influential γ-community in $G_{\geq\tau_2}$.*

THEOREM 3.1: *Let τ^* be the largest value such that $G_{\geq\tau^*}$ contains at least k influential γ-communities. Then, the set of top-k influential γ-communities in $G_{\geq\tau^*}$ is the set of top-k influential γ-communities in G.*

PROOF: First of all, we assume that G contains at least k influential γ-communities; otherwise, τ^* in the statement of the theorem is not properly defined. Let τ_{min} be the minimum vertex weight in G, then $G_{\geq\tau_{min}}$ is the same as G and moreover $\tau_{min} \leq \tau^*$. From Lemma 3.1, we know that each influential γ-community in $G_{\geq\tau^*}$ is also an influential γ-community in G. It is easy to see that all influential γ-communities in $G_{\geq\tau^*}$ have influence values at least τ^*. From Lemma 3.2, we also know that each influential γ-community in G that is not contained in $G_{\geq\tau^*}$ must have an influence value smaller than τ^*. Thus, the theorem holds. □

The Framework. Following Theorem 3.1, to compute the top-k influential γ-communities in G, we can first identify the largest influence value τ^* such that $G_{\geq\tau^*}$ contains at least k influential γ-communities, and then return the set of top-k influential γ-communities in $G_{\geq\tau^*}$ as the result. In this way, we only need to work on the subgraph $G_{\geq\tau^*}$ which can be much smaller than G. For example, $\frac{\texttt{size}(G_{\geq\tau^*})}{\texttt{size}(G)}$ is smaller than 0.073% across all the graphs tested in our experiments for $k = 10$ and $\gamma = 10$. However, it is non-trivial to obtain the appropriate influence value τ^*.

From Lemma 3.1, we know that the number of influential γ-communities in the subgraph $G_{\geq\tau}$ increases along with the decreasing of τ. Thus, one possible way to computing τ^* is conducting a binary search on the sequence of all possible vertex weights in G. However, it is time consuming to count the number of influential γ-communities in a graph, which takes linear time to the size of the graph (see Section 3.2.1), and the size of the first subgraph of G tested by the binary search may be as large as half of $\texttt{size}(G)$. Thus, binary search does not save the computational cost.

In this paper, we propose to use the exponential growth strategy for computing the target τ value; that is, we iteratively increase the size of the graph $G_{\geq\tau}$, with a growing ratio of δ, for processing. The proper setting of δ will be discussed in Section 3.3. The pseudocode of our framework is shown in Algorithm 1. We first heuristically compute the largest τ_1 value such that $G_{\geq\tau_1}$ would contain at least k influential γ-communities (Line 1). For example, τ_1 could be set as the $(k+\gamma)$-th largest vertex weight in G; that is, the k influential γ-communities contain at least $k+\gamma$ distinct vertices. Then, as long as $G_{\geq\tau_i}$ contains less than k influential γ-communities (*i.e.*, $\mathsf{CountIC}(G_{\geq\tau_i}) < k$) and $G_{\geq\tau_i}$ is not the same as G (Line 3), we find the next largest τ_{i+1} value such that the size of $G_{\geq\tau_{i+1}}$ is at least δ times the size of $G_{\geq\tau_i}$ (Line 4), and increment i by 1 (Line 5); note that, if $\texttt{size}(G)$ is smaller than $\delta\cdot\texttt{size}(G_{\geq\tau_i})$, then we set τ_{i+1} as the smallest vertex weight τ_{min} in G. Finally, we compute and return the top-k influential γ-communities in $G_{\geq\tau_i}$, which is obtained by invoking $\mathsf{EnumIC}(G_{\geq\tau_i})$, as the result (Line 6).

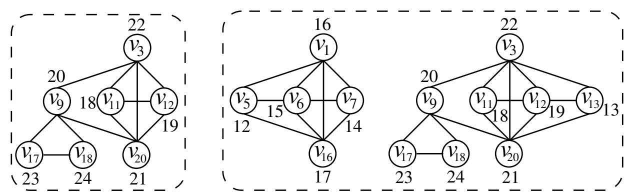
Algorithm 1: LocalSearch

Input: A graph $G = (V, E, \omega)$, and two integers k and γ
Output: Top-k influential γ-communities in G

1 $\tau_1 \leftarrow$ the largest τ value such that $G_{\geq\tau}$ would contain at least k influential γ-communities;
2 $i \leftarrow 1$;
3 **while** $\mathsf{CountIC}(G_{\geq\tau_i}, \gamma) < k$ ***and*** $G_{\geq\tau_i} \neq G$ **do**
4 $\quad\tau_{i+1} \leftarrow \max\{\{\tau \mid \texttt{size}(G_{\geq\tau}) \geq \delta\cdot\texttt{size}(G_{\geq\tau_i})\} \cup \{\tau_{min}\}\}$; `/*` τ_{min} `is the smallest vertex weight in` G `*/`;
5 $\quad i \leftarrow i + 1$;
6 **return** top-k communities in $\mathsf{EnumIC}(G_{\geq\tau_i})$;

u	v_{18}	v_{17}	v_3	v_{20}	v_9	v_{12}	v_{11}	v_{16}	v_1	v_6	v_7
$\omega(\cdot)$	24	23	22	21	20	19	18	17	16	15	14
u	v_{13}	v_5	v_0	v_{15}	v_{10}	v_8	v_{21}	v_{19}	v_4	v_2	v_{14}
$\omega(\cdot)$	13	12	11	10	9	8	7	6	5	4	3

(a) Vertices in decreasing weight order



(b) $G_{\geq\tau_1}$ ($\tau_1 = 18$) (c) $G_{\geq\tau_2}$ ($\tau_2 = 12$)

Figure 4: Running example of our local search framework

Graph Organization. As we will show in Section 3.2.1 that computing the number of influential γ-communities in a graph g (*i.e.*, $\mathsf{CountIC}(g)$) can be conducted in linear time to the size of g, which is measured by the number of vertices and the number of edges in it. Consequently, we also need efficient techniques to retrieve the induced subgraph $G_{\geq\tau}$ in linear time to its size. To do so,

★★ we assume the vertices of G are *pre-sorted* in decreasing order with respect to their weights.

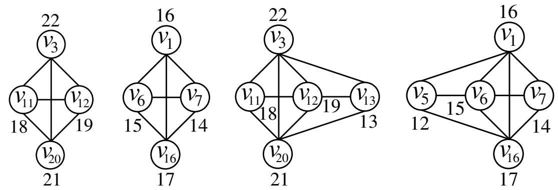
Thus, regarding a τ, the subset $V_{\geq\tau}$ of vertices can be trivially retrieved in $O(|V_{\geq\tau}|)$ time. To also retrieve the induced edges in $G_{\geq\tau}$ in linear time,

★★ we *pre-partition* the adjacent neighbors $N_G(u)$ of each vertex u into two disjoint sets: $N_G^{\geq}(u)$ contains all neighbors of u whose weights are no smaller than $\omega(u)$, and $N_G^{<}(u)$ contains the neighbors of u whose weights are smaller than $\omega(u)$.

These will support efficient online/ad-hoc queries across every k and γ, while avoiding the maintenance of indexes [27]. Thus, to construct $G_{\geq\tau}$, we only need to retrieve the set $N_G^{\geq}(u)$ of neighbors for each $u \in V_{\geq\tau}$, which can be conducted in linear time.

Based on our graph organization, we can efficiently implement Line 4 of Algorithm 1 (*i.e.*, enlarging $G_{\geq\tau_i}$ to obtain $G_{\geq\tau_{i+1}}$ whose size is at least $\delta\cdot\texttt{size}(G_{\geq\tau_i})$) as follows. We first let $G_{\geq\tau_{i+1}}$ be the same as $G_{\geq\tau_i}$, and then iteratively add into $G_{\geq\tau_{i+1}}$ the highest-weighted vertex u in $G\backslash G_{\geq\tau_{i+1}}$ and also an undirected edge between u and each of its neighbors in $N_G^{\geq}(u)$, until the obtained subgraph has a size at least $\delta\cdot\texttt{size}(G_{\geq\tau_i})$. It is easy to see that $G_{\geq\tau_{i+1}}$ is obtained from $G_{\geq\tau_i}$ in time linear to $\texttt{size}(G_{\geq\tau_{i+1}}) - \texttt{size}(G_{\geq\tau_i})$.

EXAMPLE 3.1: Consider the graph G in Figure 3, with $\gamma = 3$ and $k = 4$. The vertices of G in decreasing weight order are shown in



(a) Top-1 (b) Top-2 (c) Top-3 (d) Top-4

Figure 5: Top-4 influential γ-communities

Figure 4(a). Initially, we set τ_1 to be the weight of the 7-th vertex (*i.e.*, v_{11}) since the top-4 influential γ-communities will contain at least $k + \gamma = 7$ distinct vertices. Thus, $\tau_1 = 18$ and the subgraph $G_{\geq\tau_1}$ is shown in Figure 4(b). By invoking CountIC on $G_{\geq\tau_1}$, we know that $G_{\geq\tau_1}$ contains only one influential γ-community.

Then, we need to find the largest τ_2 value such that the size of $G_{\geq\tau_2}$ is at least δ times the size of $G_{\geq\tau_1}$; assume $\delta = 2$. As $G_{\geq\tau_1}$ has 7 vertices and 11 edges, the size of $G_{\geq\tau_1}$ is 18. We iteratively add the next highest-weight vertex into the subgraph $G_{\geq\tau_1}$. Firstly, we add v_{16} which has no edges to the subgraph. Secondly, we add v_1 with one edge to v_{16} to the subgraph. So on so forth. Until after adding v_5 to the subgraph, the size of the subgraph becomes 36. Thus, $\tau_2 = \omega(v_5) = 12$ and $G_{\geq\tau_2}$ is shown in Figure 4(c).

By invoking CountIC on $G_{\geq\tau_2}$, we know that $G_{\geq\tau_2}$ has four influential γ-communities. Thus, EnumIC computes the top-4 influential γ-communities in $G_{\geq\tau_2}$ as shown in Figure 5, which is outputted as the result of the query. □

Remark. In our framework in Algorithm 1, the graph G can be either main memory resident, or disk resident, or stored in a database. The only requirement is that there is an interface to retrieve the vertices (together with their neighbors $N^{\geq}_{G}(\cdot)$) in decreasing weight order. For example, if G is stored on disk, then Algorithm 1 can work in an I/O-efficient manner in a similar way to the semi-external algorithm in [28], as follows. It assumes that the main memory is large enough to store constant information regarding vertices as well as a subset of all edges of G, and it sorts edges in decreasing weight order in a preprocessing step, where the weight of an edge equals the minimum weight of its two end-points [28]. Thus, the neighbors in $N^{\geq}_{G}(v)$ of v are stored consecutively on disk, and to construct $G_{\geq\tau_{i+1}}$ from $G_{\geq\tau_i}$, the edges of $G_{\geq\tau_{i+1}}$ that are not in $G_{\geq\tau_i}$ are loaded sequentially from disk to main memory; then, the computations regarding $G_{\geq\tau_{i+1}}$ are conducted in main memory.

In the following, we assume that G is stored in main memory for presentation simplicity; nevertheless, we also evaluate our algorithm for the scenario that G is stored on disk in Section 6.

3.2 Our Approach

In Algorithm 1, CountIC can be achieved by invoking EnumIC. However, it is expected that counting the number of influential γ-communities in a graph would be easier than enumerating them. This is because that, the total size of influential γ-communities in a graph can be much larger than the size of the graph, since they may overlap with each other [10, 27]. Nevertheless, the existing algorithms do not count the influential γ-communities in a graph without enumerating them, and they take time at least linear to the size of the top-k influential γ-communities. Thus, we propose new algorithms for counting, as well as enumerating, influential γ-communities in a graph in the following two subsections.

3.2.1 *Influential γ-community Counting*

We first define the notion of keynode regarding influential γ-community in the following.

Definition 3.1: A vertex u in a graph G is a ***keynode*** regarding a γ value if there exists a subgraph g of G such that g has an influence value $\omega(u)$ and the minimum vertex degree of g is at least γ; note that, this subgraph g must contain u according to the definition of influence value.

For example, v_7 in Figure 3 is a keynode regarding $\gamma = 3$, since the subgraph induced by vertices $\{v_1, v_6, v_7, v_{16}\}$ has an influence value $\omega(v_7) = 14$ and a minimum degree at 3. It can also be verified that v_6 is not a keynode regarding $\gamma = 3$. In the following, for presentation simplicity we simply call a vertex a keynode without referring to the γ value which can be inferred from the context.

In order to efficiently count the number of influential γ-communities in a graph, we prove the following lemmas regarding keynode.

Lemma 3.3: *Given a graph G and a value τ, there is at most one influential γ-community in G with influence value τ.*

Lemma 3.4: *There is a one-to-one correspondence between influential γ-communities in a graph G and keynodes in G. Thus, the number of keynodes in G equals the number of influential γ-communities in G.*

In the following, given an influential γ-community g, we use key(g) to denote the unique corresponding keynode in g, according to Lemma 3.4; that is, the vertex in g with the minimum weight. Note that, an influential γ-community may contain multiple keynodes, but it is uniquely determined by the keynode with the smallest weight (*i.e.*, key(g)). For example, v_{11}, v_7, v_{13} and v_5 are keynodes for the graph in Figure 3 with $\gamma = 3$, and they correspond to the four influential γ-communities shown in Figures 5(a), 5(b), 5(c), and 5(d), respectively.

Algorithm 2: CountIC

```
Input: A graph g and an integer γ
Output: The number of influential γ-communities in g

1  g ← compute the γ-core of g;
2  keys ← ∅;
3  cvs ← ∅;
4  while g ≠ ∅ do
5      u ← arg min_{v∈g} ω(v);
6      Append u to the end of keys;
7      Remove(u, g, cvs);   /* Compute the γ-core of g\u */;
8  return |keys|;

   Procedure Remove(u, g, cvs)
9  Initialize a queue Q by u;
10 while Q ≠ ∅ do
11     Pop a vertex v from Q;
12     for each neighbor v' of v in g do
13         if the degree of v' in g is γ then
14             Push v' into Q;
15     Remove v from g and append v to the end of cvs;
```

The Algorithm CountIC**.** Following Lemma 3.4, we count the number of influential γ-communities in a graph by computing the set of keynodes in the graph. The pseudocode is shown in Algorithm 2. Given a graph g, we first reduce g to its γ-core (Line 1), which is the maximal subgraph with minimum degree at least γ [35], and initialize a sequence keys of keynodes and a sequence cvs of vertices to be empty (Lines 2–3). cvs will be used in influential γ-community enumeration and will be discussed in Section 3.2.2; we ignore cvs for the current being. Then, while the graph g is not empty (Line 4), we get the vertex u with the minimum weight in g

(Line 5), which is a keynode (Line 6), and then remove the keynode u from g and reduce the resulting graph to its γ-core (Line 7).

The procedure Remove computes the γ-core of $g \backslash u$ (*i.e.*, the resulting graph by removing u from g). Note that, as input to Remove, the graph g itself is a γ-core, but $g \backslash u$ may not be. Thus, we only need to invoke Remove for u, which will then recursively remove all vertices whose degrees become less than γ as a result of removing vertices. This is achieved by the queue Q and checking that the degree of vertex v' before removing v is γ (Line 13); thus, each vertex is pushed into the queue Q at most once.

In Algorithm 2, we omit the details of computing γ-core of g at Line 1. This actually can be achieved by invoking the procedure Remove for each vertex in g whose degree is smaller than γ.

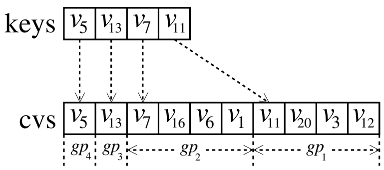


Figure 6: Running example of CountIC

Example 3.2: Consider running CountIC on the subgraph $G_{\geq \tau_2}$ shown in Figure 4(c) for $\gamma = 3$. Initially, we reduce the subgraph to its γ-core, which removes vertices $\{v_9, v_{17}, v_{18}\}$. Then, we iteratively pick the vertex u with the minimum weight from the remaining graph, add u to keys, and remove u from the graph and also maintain the γ-core. Firstly, we add v_5 to keys, whose removal does not make other vertices' degrees to be smaller than γ; thus, the procedure Remove merely removes v_5 from the graph. Secondly, we similarly add v_{13} to keys and remove it from the graph. Thirdly, we add v_7 to keys and remove it from the graph. The removal of v_7 makes the degrees of v_1, v_6, v_{16} become smaller than than γ; thus, they are all removed from the graph. Similarly, in the fourth step, we add v_{11} to keys and remove all remaining vertices from the graph; the algorithm terminates. The results of keys and cvs are shown in Figure 6. As there are four vertices in keys, we conclude that there are four influential γ-communities in $G_{\geq \tau_2}$. □

Time Complexity and Correctness of CountIC. It is easy to see that the time complexity of CountIC (*i.e.*, Algorithm 2) is linear to size of the input graph g (*i.e.*, size(g)). Note that at Line 13, rather than online counting the degree of a vertex, we maintain the degrees of all vertices at the beginning of Algorithm 2 and also when a vertex is removed from the graph at Line 15. We prove the correctness of Algorithm 2 in the following lemma and theorem.

Lemma 3.5: *After running Algorithm 2,* keys *is the set of keynodes in g.*

Theorem 3.2: *Algorithm 2 correctly computes the number of influential γ-communities in a graph g.*

Proof: This directly follows from Lemmas 3.4 and 3.5. □

3.2.2 Influential γ-community Enumeration

In this subsection, we show that the influential γ-communities can be obtained from the two arrays, keys and cvs, that are computed by CountIC.

From cvs to Communities. We call the vertex sequence in cvs as *c*ommunity-aware *v*ertex *s*equence, since the influential γ-communities can be extracted from it. From Section 3.2.1, we know that each keynode u corresponds to an influential γ-community with influence value $\omega(u)$, denoted by IC(u). It is easy to verify that IC(u) for the k vertices in keys with the largest weights (*i.e.*, the last k vertices) are the top-k influential γ-communities; note that, vertices in keys are in increasing weight order. In the following, we show how to construct IC(u) efficiently from keys and cvs.

Firstly, given keys and cvs, we construct one group for each keynode in keys. Denote the group of keynode u by gp(u), which consists of u and all vertices after u and before the next keynode in cvs; note that all keynodes of keys are in cvs. For example, for the keys and cvs in Figure 6, $\text{gp}(v_5) = \{v_5\}$, $\text{gp}(v_{13}) = \{v_{13}\}$, $\text{gp}(v_7) = \{v_7, v_{16}, v_6, v_1\}$ and $\text{gp}(v_{11}) = \{v_{11}, v_{20}, v_3, v_{12}\}$, where the groups are also shown at the bottom of Figure 6.

Secondly, IC(u) can be obtained from gp(u) recursively by the following lemma.

Lemma 3.6: *IC(u) equals the union of gp(u) and IC(u') for each keynode u' (in* keys*) such that $\omega(u') > \omega(u)$ and there is an edge between a vertex of gp(u) and a vertex of IC(u'); that is,*

$$\text{IC}(u) = \text{gp}(u) \cup \Big(\bigcup_{u' \in \text{keys}, \omega(u') > \omega(u), (\text{gp}(u) \times \text{IC}(u')) \cap E \neq \emptyset} \text{IC}(u') \Big).$$

Algorithm 3: EnumIC

Input: A graph g, a sequence keys of keynodes, a sequence cvs of vertices, and an integer k
Output: Top-k influential γ-communities

```
1  keys ← the last k keynodes in keys;
2  Initialize v2key(v) ← null for each vertex v in g;
3  for each keynode u in keys in reverse order do
4      Initialize Ch(u) ← ∅ and gp(u) ← ∅;
5      for each vertex v in cvs starting from u do
6          if v is a keynode and v ≠ u then break;
7          gp(u) ← gp(u) ∪ {v};
8          v2key(v) ← u;
9      for each vertex v in gp(u) do
10         for each neighbor w of v in g do
11             if v2key(w) ≠ null and Find(w, v2key(·)) ≠ u then
12                 Ch(u) ← Ch(u) ∪ {Find(w, v2key(·))};
13                 Union(w, u);
14     IC(u) ← gp(u) ∪ ( ⋃_{v∈Ch(u)} IC(v));
```

The Algorithm EnumIC. Based on the above discussions, the pseudocode of influential γ-community enumeration algorithm is shown in Algorithm 3. Firstly, we reduce keys to contain only the last k vertices (Line 1), and initialize a *disjoint-set data structure* v2key (Line 2), which maintains for each vertex v the smallest keynode whose corresponding influential γ-community contains v. Then, we process keynodes in keys in decreasing weight order (Lines 3–14). For each keynode u, we firstly obtain the group gp(u) (Line 7) and initialize v2key(v) to be u for each $v \in \text{gp}(u)$ (Line 8), and then process the neighbors of vertices in gp(u) (Lines 9–13). For each neighbor w, we add the current smallest keynode whose corresponding influential γ-community contains w into Ch(u) (Line 12), and then set v2key($\cdot$) to be u for all vertices in this influential γ-community (Line 13). Then, we have $\text{IC}(u) = \text{gp}(u) \cup \big(\bigcup_{v \in \text{Ch}(u)} \text{IC}(v) \big)$.

Example 3.3: Consider the keys and cvs shown in Figure 6. The 4 keynodes in increasing weight order are v_5, v_{13}, v_7, v_{11}. Firstly, we have $\text{gp}(v_{11}) = \{v_{11}, v_{20}, v_3, v_{12}\}$ and $\text{Ch}(v_{11}) = \emptyset$, and $\text{gp}(v_7) = \{v_7, v_{16}, v_6, v_1\}$ and $\text{Ch}(v_7) = \emptyset$; thus, $\text{IC}(v_{11}) = \text{gp}(v_{11})$ and $\text{IC}(v_7) = \text{gp}(v_7)$. Secondly, we have $\text{gp}(v_{13}) = \{v_{13}\}$ and $\text{Ch}(v_{13}) = \{v_{11}\}$, since v_{13} is connected to v_3, v_{12}, v_{20} that are contained in $\text{IC}(v_{11})$; that is, $\text{v2key}(v_3) = \text{v2key}(v_{12}) = \text{v2key}(v_{20}) = v_{11}$. Thus, $\text{IC}(v_{13}) = \text{gp}(v_{13}) \cup \text{IC}(v_{11})$. Similarly, we have $\text{IC}(v_5) = \text{gp}(v_5) \cup \text{IC}(v_7)$. □

Analysis. The correctness of Algorithm 3 follows from Lemma 3.6. The time complexity of Algorithm 3 is $O(\text{size}(g))$ by using the technique in [7], where $\mathsf{Find}(\cdot,\cdot)$ and $\mathsf{Union}(\cdot,\cdot)$ are the two fundamental operations on disjoint-set data structure and can be implemented to run in constant amortized time [13]. It is worth noting that, at Line 14, we only link $\mathtt{IC}(v)$ to $\mathtt{IC}(u)$ without actually copying the content of $\mathtt{IC}(v)$ to $\mathtt{IC}(u)$; otherwise, the time complexity is also linear to the output size which can be larger than $\mathtt{size}(g)$.

3.3 Analysis of LocalSearch

In the following, we analyze the time complexity of our local search algorithm LocalSearch, discuss the setting of an appropriate δ value, and prove the instance-optimality of LocalSearch.

Time Complexity. Let τ^* be the target value as defined in Theorem 3.1, and $G_{\geq\tau_h}$ be the subgraph that LocalSearch (*i.e.*, Algorithm 1) accesses before terminating. We prove the time complexity of LocalSearch by the following lemmas and theorem. Recall that, $\delta > 1$ is a parameter used at Line 4 of Algorithm 1.

LEMMA 3.7: *The time complexity of* LocalSearch *is* $O((1+\frac{1}{\delta-1})\cdot \mathtt{size}(G_{\geq\tau_h}))$.

PROOF: In Algorithm 1, a series of subgraphs (*i.e.*, $G_{\geq\tau_1},\ldots,G_{\geq\tau_h}$) are constructed and used as input to CountIC for counting influential γ-communities, and the last subgraph $G_{\geq\tau_h}$ is utilized as input to EnumIC for computing the top-k influential γ-communities. Note that, each subgraph $G_{\geq\tau}$ can be extracted from G in $O(\mathtt{size}(G_{\geq\tau}))$ time. Thus, the time complexity of LocalSearch is $(\sum_{i=1}^{h} T_1(G_{\geq\tau_i}))+T_2(G_{\geq\tau_h})$, where $T_1(g)$ and $T_2(g)$ represent the time complexities of CountIC and EnumIC, respectively. As $T_1(g)=T_2(g)=O(\mathtt{size}(g))$ and $\mathtt{size}(G_{\geq\tau_i}) \leq \frac{1}{\delta}\mathtt{size}(G_{\geq\tau_{i+1}})$ for $i<h$, the time complexity of LocalSearch is $O(\sum_{i=1}^{h} T_1(G_{\geq\tau_i})+T_2(G_{\geq\tau_h})) = O(\sum_{i=1}^{h}\mathtt{size}(G_{\geq\tau_i})+\mathtt{size}(G_{\geq\tau_h})) = O(\sum_{i=1}^{h}\frac{1}{\delta^{h-i}}\mathtt{size}(G_{\geq\tau_h})) = O((1+\frac{1}{\delta-1})\cdot\mathtt{size}(G_{\geq\tau_h}))$. □

LEMMA 3.8: *We have* $\mathtt{size}(G_{\geq\tau_h}) < 2\delta\cdot\mathtt{size}(G_{\geq\tau^*})$.

PROOF: It is easy to see that $\tau_{h-1} > \tau^* \geq \tau_h$ and $\mathtt{size}(G_{\geq\tau_{h-1}}) < \mathtt{size}(G_{\geq\tau^*})$. Let u be the vertex with the smallest weight in $G_{\geq\tau_h}$ and let $G_{\geq\tau_h}\backslash u$ be the resulting graph of removing u and all its adjacent edges from $G_{\geq\tau_h}$. Then, $\mathtt{size}(G_{\geq\tau_h}\backslash u) < \delta\cdot\mathtt{size}(G_{\geq\tau_{h-1}})$. Moreover, we have $\mathtt{size}(G_{\geq\tau_h}) \leq 2\cdot\mathtt{size}(G_{\geq\tau_h}\backslash u)+1$. Thus, $\mathtt{size}(G_{\geq\tau_h}) < 2\delta\cdot\mathtt{size}(G_{\geq\tau^*})$, and the lemma holds. □

THEOREM 3.3: *The time complexity of* LocalSearch *is* $O(\frac{2\delta^2}{\delta-1}\cdot\mathtt{size}(G_{\geq\tau^*}))$.

PROOF: This directly follows from Lemmas 3.7 and 3.8. □

Setting δ. Following Theorem 3.3, the time complexity of LocalSearch is $O(\mathtt{size}(G_{\geq\tau^*}))$ for any given constant $\delta > 1$. However, the constant factor in the time complexity will be different for different values of δ. In this paper, we set δ as 2, since $\frac{2\delta^2}{\delta-1}$ achieves the smallest value at $\delta=2$, among all δ values larger than 1; note that, $\frac{2\delta^2}{\delta-1} = 2(1+\delta+\frac{1}{\delta-1})$.

Instance-optimality of LocalSearch. Let $\mathcal{A}$ be the class of algorithms that correctly compute top-k influential communities *without indexes* and knowing only the vertex weight vector of the graph G, while all other information (such as degree/neighbors of a vertex) are obtained through accessing edges of the graph; that is, obtaining the degree of any vertex in $G_{\geq\tau}$ takes linear time to its number of neighbors in $G_{\geq\tau}$. Then, LocalSearch is a member of $\mathcal{A}$. We prove that LocalSearch is instance-optimal [14] within the class $\mathcal{A}$ of algorithms, by the following lemma and theorem.

LEMMA 3.9: *Given a graph G, any algorithm in $\mathcal{A}$ needs to access a subgraph of G of size* $\Omega(\mathtt{size}(G_{\geq\tau^*}))$.

PROOF: Let n be the number of vertices of $G_{\geq\tau^*}$, we prove that any algorithm of $\mathcal{A}$ needs to know the degrees (and thus all neighbors) of at least $n-\gamma$ vertices of $G_{\geq\tau^*}$. Let's consider an arbitrary algorithm A that computes the top-k influential communities by accessing the full lists of neighbors of only $n-\gamma-1$ vertices. Let S be the set of $\gamma+1$ vertices whose lists of neighbors are not accessed in full, and τ_S be the minimum vertex weight of S. Then, (1) we have $\tau_S > \tau^*$ according to the definition of τ^* in Theorem 3.1 and the assumption that each vertex has a distinct weight (see Section 2), and (2) the reported top-k influential communities cannot contain any vertex of S since we need to report all the edges of each community. However, S itself may form a clique in $G_{\geq\tau^*}$ such that there is an influential γ-community containing S with influence value τ_S, which is larger than the influence value τ^* of one of the k reported influential γ-communities; we cannot exclude this possibility without accessing S and without indexes. As a result, algorithm A is incorrect and does not belong to $\mathcal{A}$.

Consequently, for any algorithm B in $\mathcal{A}$, the number of edges of $G_{\geq\tau^*}$ that are not accessed by B is at most k^2, which is smaller than $\frac{1}{2}\mathtt{size}(G_{\geq\tau^*})$ since the number of edges in an influential γ-community is at least $k\cdot(k+1)$. Thus, B needs to access a subgraph of G of size $\Omega(\mathtt{size}(G_{\geq\tau^*}))$. □

Note that, Lemma 3.9 is for the case that each vertex has a distinct weight. This lemma also holds if the number of same-weight vertices is bounded by a constant. This is because the proof of Lemma 3.9 essentially implies the number of unvisited vertices in $G_{\geq\tau^*}$ with weight larger than τ^* is bounded by $\gamma+1$.

THEOREM 3.4: LocalSearch *is instance-optimal within the class $\mathcal{A}$ of algorithms.*

PROOF: This follows from Theorem 3.3 and Lemma 3.9. □

Remarks. Note that, the time complexity and the instance-optimality of LocalSearch in above are analyzed based on the assumption that the set $N_G(u)$ of neighbors of each vertex is pre-partitioned into two disjoint sets, $N_G^{\geq}(u)$ and $N_G^{<}(u)$ (see Section 3.1), such that any subgraph $G_{\geq\tau}$ can be extracted in $O(\mathtt{size}(G_{\geq\tau}))$ time. If this assumption does not hold, then we need to revise the definition of $G_{\geq\tau}$ to be consisting of all the adjacent edges in G for every vertex of $V_{\geq\tau}$. Nevertheless, the time complexity and instance-optimality of LocalSearch still hold based on the revised definition of $G_{\geq\tau}$, by using the same arguments as above.

In Algorithm 1, we choose to grow the subgraph $G_{\geq\tau_i}$ exponentially, based on which we prove the instance-optimality of LocalSearch in above. Another natural choice of growing $G_{\geq\tau_i}$ is that $\mathtt{size}(G_{\geq\tau_i}) = i\cdot m$ for a constant m; that is, add an additional total m vertices and edges to the subgraph each time. However, then the time complexity would be $(\sum_{i=1}^{h} T_1(G_{\geq\tau_i}))+T_2(G_{\geq\tau_h}) = h^2\cdot m$ which is super-linear (or even quadratic when $h \gg m$) to the size of the subgraph $G_{\geq\tau_h}$ accessed by the algorithm, as $\mathtt{size}(G_{\geq\tau_h}) = h\cdot m$. This validates our choice of exponentially growing $G_{\geq\tau_i}$.

4. A PROGRESSIVE APPROACH

In Algorithm 1, as well as in existing global search algorithms in [10, 27, 28], the influential γ-communities are only constructed and reported at the end of an algorithm; that is, the results are only available to the user when the algorithm terminates. Thus, there is a long latency delay between issuing a query and seeing any result. In this section, we propose techniques to compute and report the influential γ-communities progressively in decreasing influence value order. As a by-product of our progressive approach, the user no longer needs to specify k in the query, and can terminate the algorithm once having seen enough results.

A Progressive Framework. Recall that, Algorithm 1 firstly invokes CountIC on a series of subgraphs (*i.e.*, $G_{\geq\tau_1}, \ldots, G_{\geq\tau_h}$ with $\tau_1 > \cdots > \tau_h$) to determine the proper subgraph for processing, and then invokes EnumIC on the last subgraph $G_{\geq\tau_h}$ to compute and report the top-k influential γ-communities. From Lemma 3.1 we know that, for any two values $\tau \leq \tau'$, every influential γ-community in $G_{\geq\tau'}$ is also an influential γ-community in $G_{\geq\tau}$. Thus, the influential γ-communities in $G_{\geq\tau_h}$ can actually be partitioned into influential γ-communities in $G_{\geq\tau_1}$, and influential γ-communities in $G_{\geq\tau_i}$ but not in $G_{\geq\tau_{i-1}}$ for every $1 < i \leq h$. As a result, for each $G_{\geq\tau_i}$ with $1 \leq i \leq h$, we can compute and report a set of influential γ-communities.

Algorithm 4: LocalSearch-P

Input: A graph $G = (V, E, \omega)$, and an integer γ
Output: Influential γ-communities in G in decreasing influence value order

1 $\tau_1 \leftarrow$ the largest τ value such that $G_{\geq\tau}$ would contain an influential γ-community;
2 $\tau_0 = \tau_{max}$; /* τ_{max} is the largest vertex weight in G */;
3 $i \leftarrow 1$;
4 **while** true **do**
5   ConstructCVS($G_{\geq\tau_i}, \gamma, \tau_{i-1}$);
6   Output influential γ-communities in EnumIC-P($G_{\geq\tau_i}$, keys, cvs);
7   **if** $G_{\geq\tau_i} = G$ **then break**;
8     $\tau_{i+1} \leftarrow \max\{\{\tau \mid \texttt{size}(G_{\geq\tau}) \geq 2 \cdot \texttt{size}(G_{\geq\tau_i})\} \cup \{\tau_{min}\}\}$; /* τ_{min} is the smallest vertex weight in G */;
9     $i \leftarrow i + 1$;

Based on the above ideas, our progressive framework is shown in Algorithm 4. We initialize τ_1 be the largest τ value such that $G_{\geq\tau}$ would contain an influential γ-community (Line 1), and τ_0 be the largest vertex weight in G (Line 2). Then, we iteratively construct the keys and cvs for $G_{\geq\tau_i}$ (Line 5), compute and report the influential γ-communities in $G_{\geq\tau_i}$ that are not contained in $G_{\geq\tau_{i-1}}$ (Line 6), find the next largest τ_{i+1} such that the size of $G_{\geq\tau_{i+1}}$ is at least twice the size of $G_{\geq\tau_i}$ (Line 8), and increment i by 1 (Line 9). Note that, Algorithm 4 is terminated either when all influential γ-communities in G have been computed (Line 7) or when a user manually terminates it.

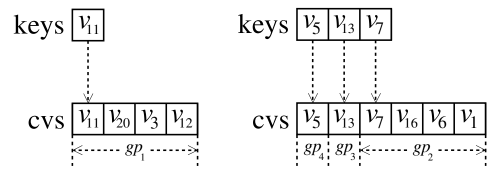
Algorithm 5: ConstructCVS

Input: A graph g, an integer γ, and a threshold τ
Output: keys and cvs

$g \leftarrow$ compute the γ-core of g;
keys $\leftarrow \emptyset$;
cvs $\leftarrow \emptyset$;
while $g \neq \emptyset$ **do**
    $u \leftarrow \arg\min_{v \in g} \omega(v)$;
    if $\omega(u) \geq \tau$ **then break**;
    Append u to the end of keys;
    Remove(u, g, cvs); /* Compute the γ-core of $g \backslash u$ */;

Incrementally Construct cvs**.** From Algorithm 2, it can be verified that the keys and cvs constructed for $G_{\geq\tau_i}$ is a suffix of that constructed for $G_{\geq\tau_{i+1}}$. Moreover, given the influential γ-communities in $G_{\geq\tau_i}$ and to compute the influential γ-communities that are in $G_{\geq\tau_{i+1}}$ but not in $G_{\geq\tau_i}$, we only need the prefixes of keys and cvs that does not contain any keynodes in $G_{\geq\tau_i}$. Thus, we can incrementally construct keys and cvs for $G_{\geq\tau_{i+1}}$ by terminating the construction once the next keynode belongs to $G_{\geq\tau_i}$. The pseudocode of incrementally constructing cvs is shown in Algorithm 5, which is similar to Algorithm 2. But in Algorithm 5, rather than counting the number of influential γ-communities in g, we construct the parts of keys and cvs that correspond to keynodes with weights smaller than a given threshold τ.

Incrementally Enumerate Influential γ-communities. The pseudocode of incrementally enumerating influential γ-communities, denoted by EnumIC-P, is similar to Algorithm 3 with the following differences. Firstly, we retain all keynodes in keys; that is, Line 1 of Algorithm 3 is removed. Secondly, the disjoint-set data structure v2key is a global structure that is shared among different runs of EnumIC-P; moreover, the v2key(v) of v is only lazily initialized for vertices in cvs.



(a) cvs for $G_{\geq\tau_1}$ (b) cvs for $G_{\geq\tau_2}$

Figure 7: Running example of LocalSearch-P

Running Example of LocalSearch-P**.** Consider the graph G in Figure 3 with $\gamma = 3$, and assume the first graph $G_{\geq\tau_1}$ obtained by LocalSearch-P is as shown in Figure 4(b). Firstly, Figure 7(a) shows the keys and cvs computed by ConstructCVS for $G_{\geq\tau_1}$, from which we can obtain the top-1 influential γ-community as $\texttt{IC}(v_{11}) = \{v_{11}, v_{20}, v_3, v_{12}\}$. Secondly, Figure 7(b) shows the keys and cvs computed by ConstructCVS for $G_{\geq\tau_2}$, where v_{11} is not included in keys. From the newly constructed keys and cvs, we can obtain the top-2, top-3, and top-4 influential γ-communities as $\texttt{IC}(v_7)$, $\texttt{IC}(v_{13})$ and $\texttt{IC}(v_5)$. Moreover, we can see that the concatenation of the two keys in Figure 7 is the same as the keys in Figure 6; this also holds for cvs.

Time Complexity of LocalSearch-P**.** For an arbitrary k, let τ_k^* be the largest value such that $G_{\geq\tau_k^*}$ contains k influential γ-communities. Then, the time complexity of LocalSearch-P is $O(\texttt{size}(G_{\geq\tau_k^*}))$ whenever a user terminates the algorithm immediately after reporting k influential γ-communities for an arbitrary k. The reasons are the same as in Section 3.3. Thus, the instance-optimality of LocalSearch carries over to LocalSearch-P.

5. EXTENSIONS

In this section, we extend our framework and techniques to non-containment community search and to other cohesiveness measures.

5.1 Non-containment Community Search

According to the definition of influential γ-community, it is possible that one influential γ-community is a subgraph of another influential γ-community. The problem of computing top-k non-containment influential communities is also studied in the literature [10, 27], based on the definition below.

Definition 5.1: [27] Given a graph G and an integer γ, an influential γ-community g is a ***non-containment influential γ-community*** if it satisfies the ***non-containment*** constraint that none of its subgraph is an influential γ-community.

It is easy to verify that the set of all non-containment influential γ-communities is disjoint.

Computing Top-k Non-containment Influential γ-communities. Our local search framework in Algorithm 1 can be used to compute the k non-containment influential γ-communities with the highest influence values, by slightly modifying CountIC (*i.e.*, Algorithm 2) and EnumIC (*i.e.*, Algorithm 3) as follows. Besides keynode, we also define *non-containment keynode* such that there is a one-to-one correspondence between non-containment keynodes and non-containment influential γ-communities. A keynode u is a non-containment keynode if every vertex that is removed during running the procedure Remove in Algorithm 2 by giving u as input is not connected to any remaining vertex of g obtained after finishing the procedure. Thus, we mark u as a non-containment keynode after Line 7 of Algorithm 2 if this condition holds. Then, the non-containment influential γ-community corresponding to a non-containment keynode u is exactly $\mathsf{gp}(u)$ (see Section 3.2.2 for the definition of $\mathsf{gp}(\cdot)$).

Let τ^* be the largest value such that $G_{\geq\tau^*}$ contains at least k non-containment influential γ-communities. It can be verified that the time complexity of computing top-k non-containment influential γ-communities is also $O(\mathtt{size}(G_{\geq\tau^*}))$. Nevertheless, this subgraph $G_{\geq\tau^*}$ is no smaller than that for computing top-k influential γ-communities, due to the fact that the set of all non-containment influential γ-communities is a subset of all influential γ-communities. Thus, it is expected that computing top-k non-containment influential γ-communities takes longer time than computing top-k influential γ-communities.

5.2 Other Cohesiveness Measures

Our framework in Section 3.1 can also be extended to the general case of top-k influential community search regarding other cohesiveness measures. We start with a general definition of influential γ-cohesive community.

Definition 5.2: Given a vertex-weighted graph $G = (V, E, \omega)$ and a parameter γ, an ***influential γ-cohesive community*** is a subgraph g of G such that the following constraints are satisfied.

- ***Connected:*** g is a connected subgraph;
- ***Cohesive:*** the cohesiveness value of g is at least γ;
- ***Maximal:*** there exists no other subgraph g' of G such that (1) g' is a supergraph of g with $f(g') = f(g)$, and (2) g' is also connected and cohesive.

Note that in the above definition, we do not specify the exact measure of cohesiveness, and it can be any of minimum degree (aka, k-core) [33, 35], average degree (aka, edge density) [9, 18], minimum number of triangles each edge participates in (aka, k-truss) [12, 32], edge connectivity (aka, k-edge connected components) [8, 40], and etc. The influential γ-community defined in Section 2 is influential γ-cohesive community where the cohesiveness of a graph is measured by the minimum degree.

A General Framework for Top-k Influential Community Search. In order for our framework in Algorithm 1 to be applicable to general top-k influential community search regarding other cohesiveness measures, the influential γ-cohesive community should satisfy the following two properties.

Property-I: For any two values $\tau_1 \leq \tau_2$, every influential γ-cohesive community in $G_{\geq\tau_2}$ is also an influential γ-cohesive community in $G_{\geq\tau_1}$ (similar to Lemma 3.1).

Property-II: For any two values $\tau_1 \leq \tau_2$ and an influential γ-cohesive community g in $G_{\geq\tau_1}$, if the influence value of g is no smaller than τ_2, then g is also an influential γ-cohesive community in $G_{\geq\tau_2}$ (similar to Lemma 3.2).

It can be verified that our definition of influential γ-cohesive community with any of minimum degree, average degree, minimum number of triangles each edge participates in, and edge connectivity, satisfies the above two properties. Thus, we can prove a similar theorem to Theorem 3.1, as follows.

Theorem 5.1: *Let τ^* be the largest value such that $G_{\geq\tau^*}$ contains at least k influential γ-cohesive communities. Then, the set of top-k influential γ-cohesive communities in $G_{\geq\tau^*}$ is the set of top-k influential γ-cohesive communities in G.*

Proof: This can be proved in a similar way to Theorem 3.1. □

Algorithm 6: LocalSearch-General

Input: A graph $G = (V, E, \omega)$, and two integers k and γ
Output: Top-k influential γ-cohesive communities in G

```
τ_1 ← the largest τ value such that G_{≥τ} would contain at least k
  influential γ-cohesive communities;
i ← 1;
while CountICC(G_{≥τ_i}, γ) < k and G_{≥τ_i} ≠ G do
    τ_{i+1} ← max {{τ | size(G_{≥τ}) ≥ 2 · size(G_{≥τ_i})} ∪ {τ_min}};  /* τ_min
      is the smallest vertex weight in G */;
    i ← i + 1;
return top-k influential γ-cohesive communities in EnumICC(G_{≥τ_i});
```

Based on Theorem 5.1, we can easily generalize our local search framework in Algorithm 1 to general top-k influential community search regarding other cohesiveness measures as mentioned above. The pseudocode of our general local search framework is shown in Algorithm 6. CountICC and EnumICC are procedures for counting and enumerating the influential γ-cohesive communities in a graph, respectively, and only these two procedures need to be specifically designed for different cohesiveness measures.

Time Complexity. Let $T_{\mathsf{Count}}(g)$ and $T_{\mathsf{Enum}}(g)$ be the time complexities of CountICC and EnumICC, respectively, for an input graph g. The time complexity of Algorithm 6 is as follows.

Theorem 5.2: *If T_{Count} is linear or super-linear, then the time complexity of Algorithm 6 is $O(T_{\mathsf{Count}}(G_{\geq\tau^*}) + T_{\mathsf{Enum}}(G_{\geq\tau^*}))$, where τ^* is as defined in Theorem 5.1.*

Proof: This can be proved in a similar way to the proofs of Lemmas 3.7 and 3.8. □

Given a graph g, a naive approach to CountICC(g) for all these cohesiveness measures is iteratively (1) computing the maximal γ-cohesive subgraph of g and reassigning it as g, and (2) removing the minimum-weight vertex from g and marking it as a keynode. This can be optimized by sharing the computation among different iterations (*e.g.*, Algorithm 2); we illustrate the optimized version of CountICC for influential γ-truss community in the full version [3].

6. EXPERIMENTS

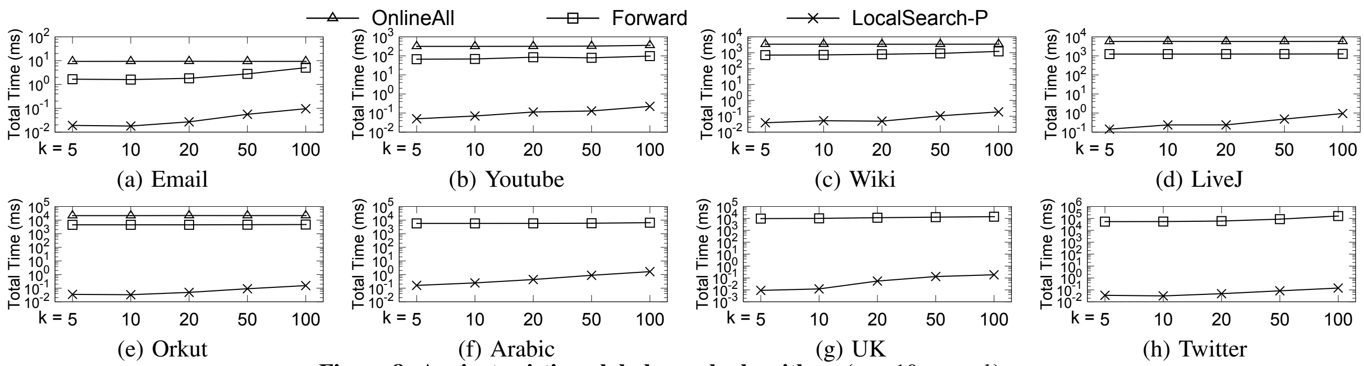
We conduct extensive performance studies to evaluate the efficiency of our local search framework and algorithms. Firstly, regarding main memory algorithms for influential γ-community search, we evaluate the following algorithms.

- OnlineAll: the existing global search algorithm in [27].
- Forward: the state-of-the-art global search algorithm in [10].
- Backward: the existing local search algorithm in [10].
- LocalSearch: our *optimal* local search algorithm (Algorithm 1).
- LocalSearch-OA: our local search algorithm by replacing CountIC with OnlineAll.
- LocalSearch-P: our *optimal* and *progressive* local search algorithm (Algorithm 4).

Secondly, we evaluate the following I/O-efficient algorithms.

- OnlineAll-SE: the semi-external version of OnlineAll [28].
- LocalSearch-SE: our semi-external version of LocalSearch-P, where edges are stored on disk (see Remark in Section 3.1).

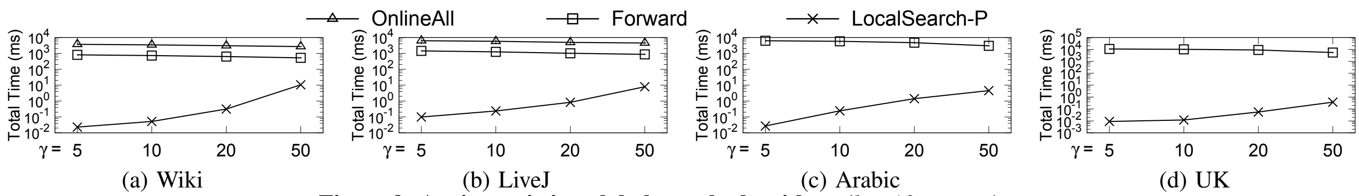
Thirdly, regarding the extension of our framework to influential γ-truss community search, we evaluate the following two algorithms.



(a) Email (b) Youtube (c) Wiki (d) LiveJ

(e) Orkut (f) Arabic (g) UK (h) Twitter

Figure 8: Against existing global search algorithms (γ = 10, vary k)



(a) Wiki (b) LiveJ (c) Arabic (d) UK

Figure 9: Against existing global search algorithms (k = 10, vary γ)

- LocalSearch-Truss: our local search algorithm for computing top-k influential γ-truss communities (Algorithm 6).
- GlobalSearch-Truss: a global search algorithm which first invokes CountICC on the entire graph, and then runs EnumICC for enumerating the top-k influential γ-truss communities.

All algorithms are implemented in C++ and compiled by GNU GCC 4.8.2 with the -O3 flag; the source code of OnlineAll is obtained from the authors of [27] while other algorithms are implemented by us. All experiments are conducted on a machine with an Intel i5 3.20GHz CPU and 16GB main memory.

Table 1: Statistics of real graphs

Graphs	#vertices	#edges	d_{max}	d_{avg}	γ_{max}
Email	36,692	183,831	1,383	10.02	43
Youtube	1,134,890	2,987,624	28,754	5.27	51
Wiki	1,791,489	25,446,040	238,342	28.41	99
LiveJ	3,997,962	34,681,189	14,815	17.35	360
Orkut	3,072,627	117,185,083	33,313	76.28	253
Arabic	22,744,080	553,903,073	575,628	48.71	3,247
UK	39,459,925	783,027,125	1,776,858	39.69	588
Twitter	41,652,230	1,468,365,182	2,997,487	70.51	2,488

Real Graphs. We evaluate the algorithms on eight real graphs: Email, Youtube, Wiki, LiveJ, Orkut, Arabic, UK, and Twitter. The first five graphs are downloaded from the Stanford Network Analysis Platform[2], while the last three are downloaded from the Laboratory of Web Algorithmics[3]. Statistics of the graphs are given in Table 1, where γ_{max} denotes the maximum value such that the graph contains a non-empty γ_{max}-core. The weights of vertices are assigned as their PageRank values with the damping factor being set as 0.85.[4]

Query Parameters. There are two query parameters, k and γ. We choose k from $\{5, 10, 20, 50, 100\}$ and γ from $\{5, 10, 20, 50\}$; $k = 10$ and $\gamma = 10$ by default. Note that, as γ_{max} for Email is 43 as shown in Table 1, the largest γ we tested for Email is 40.

In each testing, for a query with given k and γ, we run an algorithm on a graph three times and report the average CPU time in milliseconds. For main memory algorithms, the graph is assumed to be stored in main memory, while for I/O-efficient algorithms, the reported time also includes the I/O time.

6.1 Experimental Results

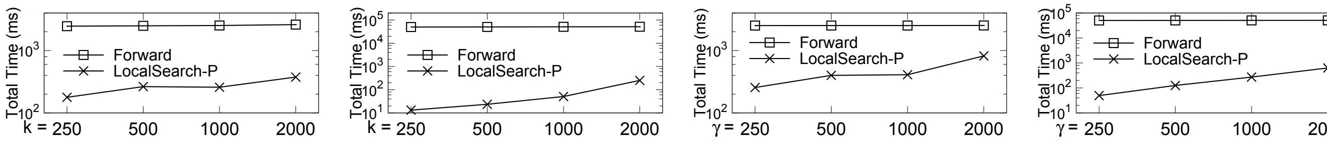
Eval-I: Against Global Search Algorithms by Varying k and γ. In this testing, we evaluate LocalSearch-P against the existing global search algorithms OnlineAll and Forward by varying k and γ. The processing time of the algorithms by varying k is shown in Figure 8, where $\gamma = 10$. We can see that the processing time of OnlineAll and Forward remains almost the same for different k values. This is because, these two algorithms need to process the entire input graph regardless of the value of k. On the other hand, LocalSearch-P runs slower for larger k, due to our local search framework that needs to access a larger subgraph for computing more influential γ-communities. Nevertheless, LocalSearch-P significantly outperforms OnlineAll and Forward across all different k values, and the improvement can be up-to 5 orders of magnitude (*e.g.*, on Orkut). Note that, we omit OnlineAll for Arabic, UK, and Twitter, since it runs out-of-memory for processing these graphs.

The results by varying γ are shown in Figure 9, where $k = 10$. Similar to the results in Figure 8, the processing time of OnlineAll and Forward remains almost the same for different γ values. The processing time of LocalSearch-P increases for larger γ value. This is because, the larger the value of γ, the smaller the influence values of the top-k influential γ-communities. Thus, LocalSearch-P needs to access a larger subgraph for computing top-k influential γ-communities of larger γ. Nevertheless, LocalSearch-P outperforms OnlineAll and Forward regarding all different values of γ.

We also evaluate the algorithms for large values of k and γ on the two graphs Arabic and Twitter that have the largest γ_{max} values (see Table 1). The results are shown in Figure 10, and the trend is similar to that of Figures 8 and 9. Although LocalSearch-P takes more time when k or γ becomes larger, it still outperforms Forward.

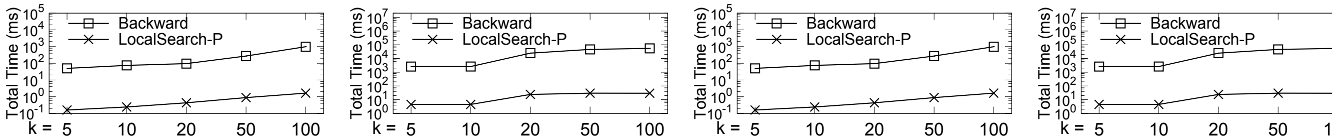
Eval-II: Against Existing Local Search Algorithm Backward. In this testing, we evaluate our local search algorithm LocalSearch-P against the existing algorithm Backward. The results are shown in Figure 11. The processing time of LocalSearch-P and Backward increases for larger k, since both algorithms need to access and process a larger subgraph for computing more communities. Nevertheless, LocalSearch-P consistently outperforms Backward. This is because LocalSearch-P has a linear time complexity regarding

[2] http://snap.stanford.edu/
[3] http://law.di.unimi.it/datasets.php
[4] https://en.wikipedia.org/wiki/PageRank



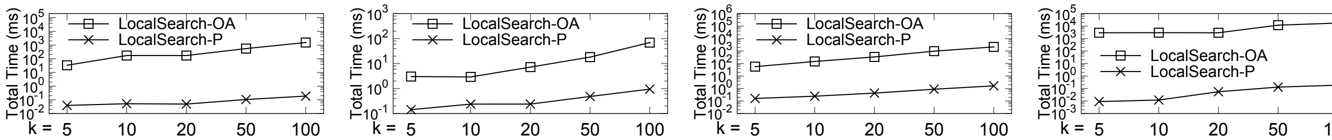
(a) Arabic (γ = 1000, vary k) (b) Twitter (γ = 1000, vary k) (c) Arabic (k = 1000, vary γ) (d) Twitter (k = 1000, vary γ)

Figure 10: Against Forward for large k and γ



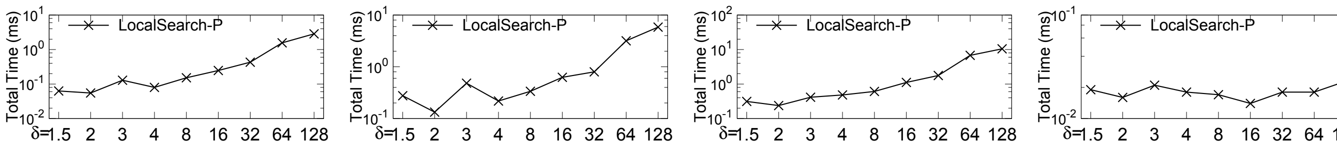
(a) Arabic (γ = 10) (b) Arabic (γ = 50) (c) UK (γ = 10) (d) UK (γ = 50)

Figure 11: Against existing local search algorithm Backward (vary k)



(a) Wiki (b) LiveJ (c) Arabic (d) UK

Figure 12: Evaluate LocalSearch-P against LocalSearch-OA (γ = 10, vary k)



(a) Wiki (b) LiveJ (c) Arabic (d) UK

Figure 13: Evaluate exponential growth ratio δ (k = 10, γ = 10)

the subgraph accessed, while Backward has a quadratic time complexity regarding the subgraph accessed [10]. The improvement of LocalSearch-P over Backward is more evident for larger γ.

Eval-III: Evaluate LocalSearch-P against LocalSearch-OA. Here, we compare LocalSearch-P against its variant that invokes OnlineAll for counting the number of influential γ-communities in a given graph, denoted LocalSearch-OA. The results in Figure 12 show that LocalSearch outperforms LocalSearch-OA. Thus, we propose a new algorithm CountIC for counting the number of influential γ-communities in a graph without enumerating them.

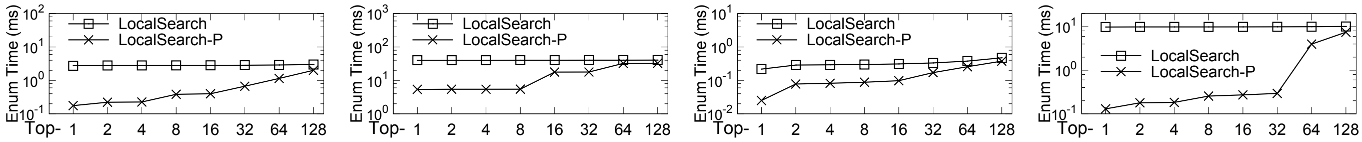
Eval-IV: Evaluate the exponential growth ratio δ. In this testing, we evaluate the performance of LocalSearch-P for different values of the growth ratio δ, chosen from $\{1.5, 2, 3, 4, 8, 16, 32, 64, 128\}$. The results are shown in Figure 13. Recall from Section 3.3 that, given any constant δ, our algorithm LocalSearch-P runs in linear time to $\mathtt{size}(G_{\geq\tau^*})$, and different values of δ will result into different constant in the time complexity. As a result, the running time of LocalSearch-P for similar values of δ are similar. In general, the processing time of LocalSearch-P increases for larger δ, and LocalSearch-P performs the best for δ being around 2.

Eval-V: Evaluate Our Progressive Approach. In this testing, we evaluate our progressive approach LocalSearch-P against our non-progressive approach LocalSearch. The experimental results regarding enumeration time are shown in Figure 14. Here k = 128, and the numeration time is the elapsed time from the start of the algorithm until the top-i community is reported. As LocalSearch reports the communities one-by-one only at the end of the algorithm, the numeration time for different communities is almost the same. In contrast, LocalSearch progressively computes and reports the communities, and thus the enumeration time increases. As a result, based on our progressive approach LocalSearch-P, the communities are reported to a user progressively as early as possible, and the user can terminate the algorithm once having seen enough communities without the need of specifying k in the query.

The results of evaluating the total processing time of LocalSearch and LocalSearch-P by varying k are shown in the full version [3]. LocalSearch-P slightly improves upon LocalSearch, despite that LocalSearch-P has the advantage of progressively reporting the communities. This is because LocalSearch-P shares computations among the processing of different subgraphs.

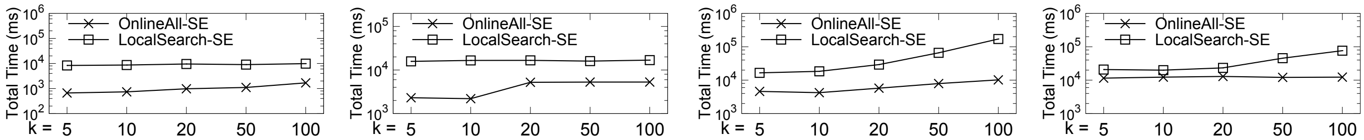
Eval-VI: Evaluate Our I/O-efficient Algorithm LocalSearch-SE. We evaluate our I/O-efficient algorithm LocalSearch-SE against the semi-external version of OnlineAll, OnlineAll-SE [28], on two large graphs Arabic and Twitter. OnlineAll-SE iteratively (1) loads as many edges as possible in decreasing weight order from disk to main memory until the memory is full, (2) conducts computation regarding the subgraph in main memory by invoking OnlineAll, and (3) removes from main memory the edges that are already part of communities and thus not needed for the following computations. In this testing, we assume that the main memory can hold 1GB of edges in addition to the information regarding vertices. The results of the total processing time are shown in Figure 15. We can clearly see that LocalSearch-SE outperforms OnlineAll-SE, which is a result of our optimal local search framework. Moreover, LocalSearch-SE consumes much smaller main memory compared with OnlineAll-SE, as shown in the full version [3].

Eval-VII: Evaluate Non-containment Queries. Here, we evaluate the efficiency of LocalSearch-P for processing non-containment queries, as discussed in Section 5.1; that is, compute the top-k influential γ-communities such that none of its subgraph is an influential γ-community [10, 27]. The results of comparing LocalSearch-P with Forward for non-containment queries are shown in Figure 16; note that, here Forward refers to its variant in [10] that computes non-containment communities. We can see that LocalSearch-P clearly outperforms Forward.



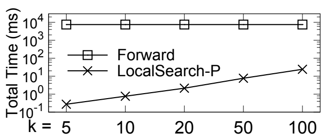
(a) Arabic ($\gamma = 10$) (b) Arabic ($\gamma = 50$) (c) UK ($\gamma = 10$) (d) UK ($\gamma = 50$)

Figure 14: Evaluate our progressive approach regarding enumeration time ($k = 128$)



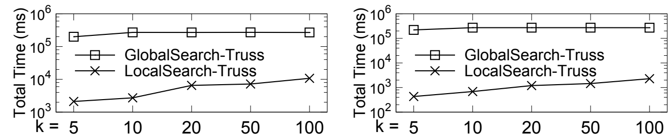
(a) Arabic ($\gamma = 10$) (b) Arabic ($\gamma = 50$) (c) Twitter ($\gamma = 10$) (d) Twitter ($\gamma = 50$)

Figure 15: Evaluate our I/O-efficient algorithm LocalSearch-SE regarding total processing time (vary k)



(a) Arabic (b) UK

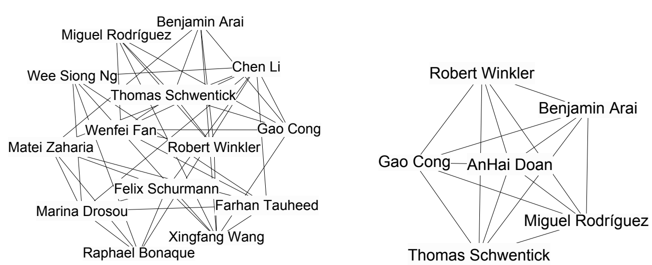
Figure 16: Evaluate non-containment queries (vary k)



(a) Wiki (b) LiveJ

Figure 17: Evaluate γ-truss community search queries (vary k)

Eval-VIII: Evaluate Influential γ-truss Community Search. In this testing, we evaluate the efficiency of our local search approach LocalSearch-Truss for processing influential γ-truss community search queries. To do so, we compare it with a global search approach GlobalSearch-Truss that traverses the entire graph. The results are shown in Figure 17, where $\gamma = 10$. We can see that LocalSearch-Truss significantly outperforms GlobalSearch-Truss. This demonstrates the superiority of our local search framework for general top-k influential community search regarding other cohesiveness measures. By comparing Figure 17 with Figure 8, we can see that computing top-k influential γ-truss communities generally takes more time than computing top-k influential γ-communities. This is because computing γ-truss communities has a higher time complexity and also processes a larger subgraph of G, than computing γ-communities.



(a) Top-1 γ-community (b) Top-1 γ-truss community

Figure 18: Case study on DBLP

Eval-IX: Case Study on DBLP. Here, we conduct a case study for the influential γ-community and γ-truss community on a co-author network, DBLP. We extract a co-author graph from DBLP (`http://dblp.unitrier.de/xml/`) by focusing on the research areas of *Artificial Intelligence*, *Computer Vision*, *Information Retrieval*, *Data Mining*, *Database*, *Machine Learning* and *Natural Language*. Each vertex corresponds to a researcher that has published at least 10 papers in these research areas, and there is an edge between two researchers if they have co-authored at least 3 papers. Weights of vertices are computed as their PageRank values. The top-1 influential 5-community and 6-truss community are shown in Figures 18(a) and 18(b), respectively; note that, the 5-core community of the vertices in Figure 18(a) consists of 1,148 vertices, as shown in the full version [3]. Researchers in these influential communities are good candidates to be invited to co-organize an interdisciplinary workshop on these research areas.

The minimum weight vertex in Figure 18(a) is "Xingfang Wang" which ranks 215 out of 1743 vertices, and the minimum weight vertex in Figure 18(b) is "AnHai Doan" which ranks 339; note that, the higher the weight of a vertex the smaller its rank. Thus, although influential γ-truss community search can find smaller and denser communities, γ-truss communities usually have smaller influence values than γ-communities since the γ-truss constraint is harder to be satisfied than the γ-core constraint. Note that, for any influential γ-truss community g with influence value τ, there is a corresponding $(\gamma - 1)$-community with influence value τ that contains g.

7. CONCLUSION

In this paper, we developed a local search framework for the problem of top-k influential community search. We proved that our LocalSearch algorithm for top-k influential γ-community search is instance-optimal, in the sense that its time complexity is linearly proportional to the size of the smallest subgraph that a correct algorithm needs to access without indexes. We further proposed techniques to make LocalSearch progressively compute and report the influential γ-communities. We also extended our local search framework to the general case of top-k influential community search regarding other cohesiveness measures. Extensive empirical studies on real graphs demonstrated the superiority of our local search approach over the existing online search algorithms. One direction of future work is to integrate our techniques to the WebGraph framework [4] to process larger graphs in main memory. Another possible direction is extending our techniques to the case that the vertex weight vector is computed online based on the query; for example, the weight of a vertex is computed as the reciprocal of its shortest distance to the query vertices as studied in [24]. It will also be an interesting future work to extend our techniques to other community definitions (*e.g.*, that surveyed in [16]), besides cohesive communities that we investigated in this paper.

Acknowledgements. Lijun Chang is supported by ARC DE150100563 and DP160101513. Xuemin Lin is supported by NSFC61672235, DP170101628, DP180103096. Wenjie Zhang is supported by ARC DP150103071 and DP150102728.

8. REFERENCES

[1] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data Min. Knowl. Discov.*, 29(5):1406–1433, 2015.

[2] E. R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):541–550, 1982.

[3] F. Bi, L. Chang, X. Lin, and W. Zhang. An optimal and progressive approach to online search of top-k influential communities. *CoRR'17*, abs/1711.05857, 2017.

[4] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. of WWW'04*, pages 595–601, 2004.

[5] S. Brohée and J. van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7:488, 2006.

[6] L. Chang, W. Li, L. Qin, W. Zhang, and S. Yang. pscan: Fast and exact structural graph clustering. *IEEE Trans. Knowl. Data Eng.*, 29(2):387–401, 2017.

[7] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proc. of SIGMOD'15*, pages 459–474, 2015.

[8] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *Proc. SIGMOD'13*, pages 205–216, 2013.

[9] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proc. of APPROX'00*, pages 84–95, 2000.

[10] S. Chen, R. Wei, D. Popova, and A. Thomo. Efficient computation of importance based communities in web-scale networks using a single machine. In *Proc. of CIKM'16*, pages 1553–1562, 2016.

[11] J. J. Cho, Y. Chen, and Y. Ding. On the (co)girth of a connected matroid. *Discrete Applied Mathematics*, 155(18):2456–2470, 2007.

[12] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, page 16, 2008.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[15] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[16] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75 – 174, 2010.

[17] A. Gajewar and A. D. Sarma. Multi-skill collaborative teams based on densest subgraphs. In *Proc. of ICDM'12*, pages 165–176, 2012.

[18] A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984.

[19] T. Hastie, J. Friedman, and R. Tibshirani. Additive models, trees, and related methods. In *The Elements of Statistical Learning*, pages 257–298. Springer, 2001.

[20] A. Hlaoui and S. Wang. A direct approach to graph clustering. *Neural Networks and Computational Intelligence*, 4(8):158–163, 2004.

[21] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. On minimal steiner maximum-connected subgraph queries. *IEEE Trans. Knowl. Data Eng.*, 29(11):2455–2469, 2017.

[22] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *Proc. of SIGMOD'14*, pages 1311–1322, 2014.

[23] X. Huang, L. V. S. Lakshmanan, and J. Xu. Community search over big graphs: Models, algorithms, and opportunities. *ICDE Tutorial*, 2017.

[24] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.

[25] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[26] J. Li, X. Wang, K. Deng, X. Yang, T. Sellis, and J. X. Yu. Most influential community search over large social networks. In *Proc. of ICDE'17*, pages 871–882, 2017.

[27] R. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5):509–520, 2015.

[28] R. Li, L. Qin, J. X. Yu, and R. Mao. Finding influential communities in massive networks. *VLDB J.*, 26(6):751–776, 2017.

[29] M. E. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

[30] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proc. of NIPS'01*, pages 849–856, 2001.

[31] M. J. Rattigan, M. E. Maier, and D. D. Jensen. Graph clustering with network structure indices. In *Proc. of ICML'07*, pages 783–790, 2007.

[32] K. Saito and T. Yamada. Extracting communities from complex networks by the k-dense method. In *Proc. of ICDMw'06*, pages 300–304, 2006.

[33] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *PVLDB*, 10(3):97–108, 2016.

[34] A. Schenker, M. Last, H. Bunke, and A. Kandel. Graph representations for web document clustering. In *Proc. of IbPRIA'03*, pages 935–942, 2003.

[35] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269 – 287, 1983.

[36] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proc. of CVPR'97*, pages 731–737, 1997.

[37] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *Proc. of SIGKDD'10*, pages 939–948, 2010.

[38] P. R. Suaris and G. Kedem. An algorithm for quadrisection and its application to standard cell placement. *IEEE Transactions on Circuits and Systems*, 35(3):294–303, 1988.

[39] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[40] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li. Finding maximal k-edge-connected subgraphs from a large graph. In *Proc. of EDBT'12*, pages 480–491, 2012.