Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

# Fast Algorithm for Attributed Community Search

Shohei Matsugu[1,a)] Hiroaki Shiokawa[1,b)] Hiroyuki Kitagawa[1,c)]

Received: June 9, 2020, Accepted: September 24, 2020

**Abstract:** Searching communities on attributed graphs has attracted much attention in recent years. The community search algorithm is currently an essential graph data management tool to find a community suited to a user-specified query node. Although community search algorithms are useful in various web-based applications and services, they have trouble handling attributed graphs due to the strict topological constraints of traditional algorithms. In this paper, we propose an accurate community search algorithm for attributed graphs. To relax the topological constraints, we proposed a new model of the community. And we defined the problem of finding them in an attributed graph class called the Flexible Attributed Truss Community (F-ATC). The F-ATC problem has the advantage of being applicable in many situations because it can explore diverse communities. Consequently, the community search accuracy is enhanced compared to traditional community search algorithms. Additionally, we present a novel heuristic algorithm to solve the F-ATC problem. This effective algorithm detects more accurate communities from attributed graphs than the traditional algorithms. For further optimization, we pre-processed the query response to make it faster. Finally, we conducted extensive experiments with real-world attributed graphs to demonstrate that our approach outperforms state-of-the-art methods.

**Keywords:** graphs, community search, clustering

## 1. Introduction

Given an attributed graph, how can we efficiently find the most suitable (or relevant) community to a user-specified query node among all possible communities? Recent advances in information and social sciences have shown that attributed graphs are becoming increasingly important as they represent complicated and schema-less data. For example, in the case of a friendship network (e.g., Facebook), each user node has several attributes such as affiliation, residential area, and topics of interests.

To understand such complicated graphs, community search algorithms [2], [4], [18] play an important role in various applications. Once they receive a query node from a user, the community search algorithms explore a single community (cluster) that has dense inner-community connections with the largest relevance to the query node. Unlike traditional community detection algorithms (e.g., modularity-based methods [1], [14] and density-based methods [16], [17]), community search algorithms can return a search result within a short computation time since they do not need to compute the entire graph. Due to their efficiency, such algorithms have been applied to various applications, including social analysis and protein analysis.

Although community search algorithms are useful in various applications, they have a serious weakness when handling real-world attributed graphs. Traditional community search algorithms [2], [4], [18] cannot find accurate communities on attributed graphs. Many real-world graphs consist of relationships among various attributes [13], [22]. However, traditional algorithms attempt to detect a dense subgraph such as $k$-core [18] or $k$-truss [2], [4], which is the most relevant to the query node without measuring attribute similarities between the query and the community. That is why traditional algorithms fail to capture attributed-driven communities [3].

### 1.1 Existing Approaches and Challenges

To address the above issue, Huang et al. recently proposed LocATC [3], which is a novel community search algorithm for attributed graphs. Once parameters $k$ and $d$ are specified, LocATC searches a $(k, d)$-truss [3] that yields the largest attribute similarity for the query node. In this study, $(k, d)$-truss is a subgraph (1) whose nodes are at least $d$-hop reachable from the query node and (2) whose edges have at least $(k - 2)$ triangles (a.k.a., 3-cliques). Although LocATC successfully handles attributed graphs, its community search accuracy is limited for real-world graphs because LocATC searches $(k, d)$-trusses under the assumption that each community contains a sufficient number of triangles. However, this assumption is not suitable for real-world graphs since they have very diverse topological structures. For example, as Leskovec and Krevl reported in Ref. [6], the average fraction of triangles is only 5.68% in various real-world graphs. In addition, Shiokawa et al. reported that real-world graphs show a wide range of clustering coefficient values [16]; that is, real-world graphs may not contain many triangles. Consequently, LocATC fails to detect precise communities in various real-world graphs, which makes it difficult to efficiently find accurate communities in attributed graphs.

[1] University of Tsukuba, Tsukuba, Ibaraki, 305–8577, Japan
[a)] matsugu@kde.cs.tsukuba.ac.jp
[b)] shiokawa@cs.tsukuba.ac.jp
[c)] kitagawa@cs.tsukuba.ac.jp

### 1.2 Our Approaches and Contributions

Our goal is to achieve fast and accurate community searches on large-scale attributed graphs. In this paper, we define a novel community search problem called the *flexible attributed truss community* (*F-ATC*) *problem* and present novel heuristic community search algorithms to efficiently solve it. To overcome the aforementioned limitations, the F-ATC problem finds a $(k, d)$-truss that maximizes the attribute similarity under all possible $k$ settings, whereas LocATC explores communities only for a specific $k$ value. Although such a relaxation increases the computational cost compared to LocATC, the F-ATC problem allows community search algorithms to explore diverse subgraphs regardless of the actual topological structures included in real-world graphs. To moderate the computational costs incurred by the F-ATC problem, herein we propose two heuristic community search algorithms based on the well-known beam-search algorithm [12].

In our previous method [9], [10], we presented an accurate algorithm design compared to LocATC, however it consumes large computation time. In this paper, we extend [9], [10] to propose more efficient search methods based on [9], [10]. We also propose a preprocessing method for faster query response. In addition, we conducted further extensive experiments to assess the effectiveness of the proposed approaches.

Consequently, our proposed methods achieves the following attractive characteristics:

- **Accurate:** Our proposed method can identify more accurate communities than those obtained by the state-of-the-art method LocATC because parameter $k$ of $(k, d)$-truss is relaxed (Section 4.2).
- **Fast:** Compared with the state-of-the-art method LocATC, our proposal achieves high-speed community searches on attributed graphs (Section 4.3). That is, our proposed method can find accurate communities without sacrificing the community search efficiency (Section 4.5).
- **Easy to deploy:** Our proposed method does not require parameter $k$, which determines the number of triangles included in each community (Algorithm 1). Therefore, our proposal provides a simple solution for diverse applications.

Our extensive experiments showed that our proposed algorithms run up to 50 times faster than LocATC without sacrificing the community search accuracy. For example, our algorithm can compute an attributed graph with 1.1 million nodes and 3 million edges in 0.1 seconds. Although previous community search algorithms have effectively enhanced application quality, they are difficult to apply to large-scale attributed graphs due to their accuracy and efficiency limitations. On the other hand, our propose method should improve the effectiveness of a wide range of applications and realize a fast and accurate approach appropriate to real-world graphs.

**Organization:** This paper is organized as follows. Section 2 introduces basic notations and definitions of this work. Section 3 defines the F-ATC problem, and presents two greedy algorithms; baseline algorithm and fast enumeration algorithm. Section 4 describes the experiments to verify the effectiveness of our approaches. Related works are briefly reviewed in Section 5. Finally, Section 6 concludes this paper.

## 2. Basic Notations and Definitions

Here, we formally define basic notations and definitions used in this paper. Let $G = (V, E, A)$ be a connected attributed graph, where $V$, $E$, and $A$ are sets of nodes, undirected edges, and attributes, respectively. Each node $u \in V$ has a set of attributes denoted by $attr(u) \subseteq A$. To simplify the representaions, each node is assumed to have one or two attributes (i.e., $1 \leq |attr(u)| \leq 2$). Without loss of generality, other types of attributed graphs can be handled even if each node has more than two attributes. For convenience, $V(H)$ and $E(H)$ are denoted as sets of nodes and edges included in subgraph $H$, respectively. Furthermore, given a set of nodes $S \subseteq V$, $G[S] = (V', E', A)$ is an induced subgraph by $S$. Here, $V' = S$ and $E'$ is all the edges in $E$ that have both nodes are in $S$. In addition, $a \in A$, $V_a(H)$ is a set of nodes with attribute $a$ in subgraph $H$. Similarly, we define a user-specified query as $q = (v_q, A_q)$, where $v_q$ is a query node included in $V(G)$, and $A_q$ is a set of query attributes such that $A_q \subseteq A$. **Table 1** summarizes symbols and their corresponding definitions used in this paper. The following basic definitions are necessary to discuss the new community search algorithms in the next section:

**Definition 1 (Query distance)** Let $dist(u, v)$ be the shortest path distance between nodes $u$ and $v$ on graph $G$. Given subgraph $H \subseteq G$ and query node $v_q$, the query distance between query node $v_q$ and subgraph $H$ is defined as $dist(v_q, H) = \max_{v \in V(H)} dist(v_q, v)$.

**Definition 2 ($(k, d)$-truss)** Let $sup(e)$ be the number of triangles containing the edge $e \in E$. Given query node $v_q$ and parameters $k$ and $d$, a set of $(k, d)$-trusses is defined as

$$\Delta_{k,d} = \{H \subseteq G | \forall e \in E(H), \min\{sup(e)\} = k-2, dist(v_q, H) \leq d\}.$$

Definition 2 indicates that a $(k, d)$-truss is a subgraph such that (1) the nodes are $d$-hop reachable from the query node $v_q$ and (2) each edge has more than $k - 2$ triangles, (i.e., $sup(e) \geq k - 2$ for each $e \in E(V)$). By controlling the values of $k$ and $d$, we can determine the density and the size of $(k, d)$-trusses. **Figure 1** shows examples of $(k, 1)$-trusses for various $k$ settings. For instance, as shown in Fig. 1, all $(k, 1)$-trusses are 1-hop reachable

**Table 1** Definition of main symbols.

| Symbol | Definition |
|---|---|
| $G$ | Connected attributed graph |
| $V$ | Set of nodes in $G$ |
| $E$ | Set of edges in $G$ |
| $A$ | Set of attributes in $G$ |
| $attr(u)$ | Set of attributes attached on node $u \in V$ |
| $V(H)$ | Set of nodes in a subgraph $H$ |
| $V_a(H)$ | Set of nodes having an attribute $a \in A$ in a subgraph $H$ |
| $E(H)$ | Set of edges in a subgraph $H$ |
| $G[S]$ | An induced subgraph by a set of nodes $S$ |
| $q$ | User-specified query |
| $v_q$ | User-specified query node such that $v_q \in V$ |
| $A_q$ | User-specified query attributes such that $A_q \subseteq A$ |
| $dist(v_q, H)$ | Query distance between a query node $v_q$ and a subgraph $H$ (Definition 1) |
| $sup(e)$ | Number of triangles (3-cliques) with an edge $e \in E$ in $G$ |
| $\Delta_{k,d}$ | Set of $(k, d)$-trusses in $G$ (Definition 2) |
| $f(H, A_q)$ | Attribute score function (Definition 3) |
| $N(H)$ | Set of 1-hop neighbor nodes of a subgraph $H$ (Definition 5) |
| $C(H)$ | Set of candidate communities obtained from a subgraph $H$ (Definition 6) |

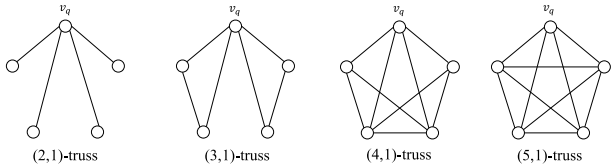



**Fig. 1** Examples of $(k, 1)$-truss.

from the query node $v_q$. If $k = 2$, each edge in $(2, 1)$-truss does not need to have any triangles. By contrast, in the case of $k = 5$, $(5, 1)$-truss should contain at least three triangles for each edge. Finally, we introduce an attribute score function [3] that evaluates the attribute similarity between the query and a community.

**Definition 3 (Attribute score function [3])** Given subgraph $H$ and set of query attributes $A_q$, attribute score function $f(H, A_q)$ is defined as

$$f(H, A_q) = \sum_{a \in A_q} \frac{|V_a(H)|^2}{|V(H)|}.$$

Definition 3 implies that attribute score function $f(H, A_q)$ increases as the subgraph contains more attributes in query attributes $A_q$.

## 3. Proposed Method

Our goal is to efficiently find an accurate community in $G$ that corresponds to the user-specified query. To achieve a highly accurate community search, we first present a novel class of the community search problem, called the *F-ATC problem*, in Section 3.1. In Sections 3.2 and 3.3, we propose two heuristic search algorithms to efficiently solve the F-ATC problem.

### 3.1 The F-ATC Problem

LocATC imposes strict topological constraints such that each community should contain a sufficient number of triangles based on user-specified parameter $k$. However, this assumption is not suitable for real-world graphs, which generally have diverse topological structures and may not contain triangles [6]. Thus, we introduce a new class of the community search problem that relaxes the strict $k$ setting in LocATC.

**Definition 4 (the F-ATC problem)** Given graph $G = (V, E, A)$, query $q = (v_q, A_q)$, and parameter $d$, the F-ATC problem finds subgraph $H \in \bigcup_{k \geq 2} \Delta_{k,d}$ that yields the largest value of $f(H, A_q)$.

Unlike LocATC [3], the F-ATC problem does not require parameter $k$. It attempts to find $(k, d)$-truss maximizing the attribute score function under all possible $k$ settings. For example, if $d = 1$, the F-ATC problem explores all $(k, 1)$-trusses shown in Fig. 1 and returns a single $(k, 1)$-truss that yields the largest score of the attribute score function.

By relaxing user-specified parameter $k$, the F-ATC problem can handle diverse typologies of real-world graphs. However, the F-ATC problem requires exhaustive subgraph searches to obtain a subgraph that maximizes the attribute score function. Let $\overline{k}$ be the maximum $k$ setting for a given graph. We can reduce the F-ATC problem to the $(k, d)$-truss search problem in the polynomial time by performing LocATC [3] for $k = 2$ to $k = \overline{k}$. As discussed in Ref. [3], the $(k, d)$-truss search problem is NP-hard. Therefore, the F-ATC problem is also NP-hard. Below, we present two heuristic search algorithms to efficiently solve the F-ATC problem.

### 3.2 Baseline Algorithm

We refine [9] as our baseline algorithm, which is an algorithm to improve the accuracy of LocATC [3]. The baseline algorithm is based on the well-known beam search technique [12]. By letting $\beta$ represent the beam width that controls a number of search results, a beam search explores graphs maintaining top-$\beta$ search results under an objective function. Based on this search strategy, the baseline algorithm greedily explores top-$\beta$ $(k, d)$-trusses by the attribute score function.

Before providing detailed descriptions of the baseline algorithm, we introduce the following definitions:

**Definition 5 (1-hop neighbor nodes)** Given subgraph $H \subseteq G$, $N(H)$ is 1-hop neighbor nodes of $H$ defined as

$$N(H) = \{v \in V(G) | (u, v) \in E(G) \textit{ for } u \in V(H) \textit{ and } v \notin V(H)\}.$$

**Definition 6 (Candidate communities)** Given subgraph $H \subseteq G$ and beam width $\beta$, we denote a set of candidate communities as $C(H)$, which is defined as

$$C(H) = \{C_1(H), C_2(H), \ldots, C_\beta(H)\} \subseteq 2^{|V(H) \cup N(H)|},$$

where $C_i(H)$ is a $(k, d)$-truss composed of nodes in $V(H) \cup N(H)$ such that $f(C_1(H), A_q) \geq f(C_2(H), A_q) \geq \cdots \geq f(C_\beta(H), A_q) \geq f(C_{\beta+j}(H), A_q)$ for all $j \in \mathbb{N}$.

Definition 6 indicates that (1) $C(H)$ expands subgraph $H$ as $V(H) \cup N(H)$ and (2) $C(H)$ lists the top-$\beta$ $(k, d)$-trusses from $V(H) \cup N(H)$ so that $C(H)$ maximizes the attribute score function among all possible $(k, d)$-trusses.

#### 3.2.1 Algorithm

Based on the above definitions, we present the baseline algorithm to solve the F-ATC problem. Given graph $G = (V, E, A)$, query $q = (v_q, A_q)$, parameter $d$, and beam width $\beta$, we initially set a subgraph as $H = \{v_q\}$. Afterwards, the baseline algorithm performs the following three steps:

**(Step 1)** Obtain $N(H)$ from subgraph $H$ by Definition 5.
**(Step 2)** Construct $C(H)$ from $V(H) \cup N(H)$ by Definition 6.
**(Step 3)** Select a $(k, d)$-truss $C_i(H)$ from $C(H)$, and set $H = C_i(H)$.

The baseline algorithm iterates the above steps until all the $d$-hop reachable nodes of $v_q$ are computed. After the terminating, it returns a $(k, d)$-truss that yields the largest score of the attribute score function in $C(H)$.

By iteratively enumerating $C(H)$ in (Step 2), the baseline algorithm explores $(k, d)$-trusses for various $k$ settings so that the trusses increase the attribute score function. However, (Step 2) requires $\Omega(2^{|V(H) \cup N(H)|} |E(G[N(H)])|^{1.5})$ time to find top-$\beta$ candidate communities from Definition 6. This is because (1) $(k, d)$-trusses need to be explored from all possible subgraph in $|V(H) \cup N(H)|$ and (2) $(k, d)$-truss detection requires $\Omega(|E(G[N(H)])|^{1.5})$ time [7]. If a given graph is large, the size of $V(H) \cup N(H)$ clearly increases. To improve the efficiency, it is important to reduce the computational cost of (Step 2).

### 3.3 Fast Enumeration Algorithm

To solve the F-ATC problem on large attributed graphs, we present a fast enumeration algorithm to search the candidate communities in the baseline algorithm. Instead of enumerating all possible candidates, the fast enumeration algorithm directly lists the top-$\beta$ candidates using attribute-aware candidate selection techniques.

For simplicity, we denote the query attributes as $A_q = \{a_1, a_2\}$ without loss of generality. To achieve fast top-$\beta$ candidate enumeration, we have the following properties from Definition 3.

**Lemma 1** Given subgraph $H$ and node $v \in N(H)$, $f(H \cup \{v\}, A_q) > f(H, A_q)$ holds, if $a_1, a_2 \in attr(v)$ holds.

**Proof** Since $a_1, a_2 \in attr(v)$, the following equation is derived from Definition 3,

$$f(H \cup \{v\}, A_q) - f(H, A_q) = \sum_{a \in A_q} \left\{ \frac{(|V_a(H)| + 1)^2}{|V(H)| + 1} - \frac{|V_a(H)|^2}{|V(H)|} \right\}$$

$$= \frac{\sum_{a \in A_q} \left\{ |V(H)|(|V_a(H)| + 1)^2 - (|V(H)| + 1)|V_a(H)|^2 \right\}}{|V(H)|(|V(H)| + 1)}.$$

Clearly, $|V(H)| \geq |V_a(H)|$ for all attribute $a \in A_q$. Hence,

$$|V(H)|(|V_a(H)| + 1)^2 - (|V(H)| + 1)|V_a(H)|^2 > 0.$$

Therefore, we have $f(H \cup \{v\}, A_q) - f(H, A_q) > 0$, which completes the proof. □

**Lemma 2** Given subgraph $H$ and node $v \in N(H)$, $f(H \cup \{v\}, A_q) < f(H, A_q)$ holds, if $a_1, a_2 \notin attr(v)$ holds.

**Proof** Due to $a_1, a_2 \notin attr(v)$, $|V_a(H \cup \{v\})| = |V_a(H)|$ clearly holds for all attribute $a \in A_q$. That is, from Definition 3,

$$f(H \cup \{v\}, A_q) - f(H, A_q) = \sum_{a \in A_q} \left\{ \frac{|V_a(H \cup \{v\})|^2}{|V(H)| + 1} - \frac{|V_a(H)|^2}{|V(H)|} \right\}$$

$$= \sum_{a \in A_q} \left\{ \frac{|V_a(H)|^2}{|V(H)| + 1} - \frac{|V_a(H)|^2}{|V(H)|} \right\} < 0,$$

which completes the proof of Lemma 2. □

For a given subgraph $H$ and its 1-hop neighbor node $v$, Lemma 1 and Lemma 2 imply that (1) if node $v$ has all query attributes in $A_q$, $(k, d)$-trusses composed of $H \cup \{v\}$ always increase the attribute score function, and (2) if node $v$ has no query attributes in $A_q$, the $(k, d)$-trusses decreases the function. We also identify the following properties, which play essential roles in our fast enumeration algorithm.

**Lemma 3** Given subgraph $H$ and node $v \in N(H)$ such that $a_1 \in attr(v)$ and $a_2 \notin attr(v)$, $f(H \cup \{v\}, A_q) > f(H, A_q)$ if and only if $|V(H)|(2|V_{a_1}(H)| + 1) > |V_{a_1}(H)|^2 + |V_{a_2}(H)|^2$ holds.

**Proof** We first prove the sufficient condition. From Definition 3,

$$f(H \cup \{v\}, A_q) - f(H, A_q) = \frac{2|V_{a_1}(H)| + 1}{|V_{a_1}(H)|^2 + |V_{a_2}(H)|^2} - \frac{1}{|V(H)|}.$$

Since we clearly have $f(H \cup \{v\}, A_q) - f(H, A_q) > 0$, we can derive $|V(H)|(2|V_{a_1}(H)| + 1) > |V_{a_1}(H)|^2 + |V_{a_2}(H)|^2$ from the above equation, which completes the proof.

Next, we prove the necessary condition. From $|V(H)|(2|V_{a_1}(H)| + 1) > |V_{a_1}(H)|^2 + |V_{a_2}(H)|^2$, the following condition is derived;

**Algorithm 1** Fast enumeration

**Require:** A subgraph $H$, a parameter $\beta$, and a parameter $d$
**Ensure:** A set of candidate communities $C(H)$

```
1: Obtain a subgraph H' from V(H) ∪ N(H) by Theorem 1;
2: while |C(H)| < β do
3:     k̄ ← max_{e∈E(H')} sup(e);
4:     for k = k̄ to 2 do
5:         Add all (k, d)-trusses in H' into C(H);
6:         if |C(H)| ≥ β then
7:             break;
8:         end if
9:     end for
10:    Obtain a node v ∈ V(H')∩N(H) decreasing f(H', A_q) by Corollary 1;
11:    H' ← H'\{v};
12: end while
```

$$0 < \frac{2|V_{a_1}(H)| + 1}{|V_{a_1}(H)|^2 + |V_{a_2}(H)|^2} - \frac{1}{|V(H)|} = f(H \cup \{v\}, A_q) - f(H, A_q).$$

Thus, $f(H \cup \{v\}, A_q) > f(H, A_q)$ holds. □

Lemma 3 leads the following corollary for given subgraph $H$ and node $v \in N(H)$ such that $a_1 \in attr(v)$ and $a_2 \notin attr(v)$.

**Corollary 1.** *$f(H \cup \{v\}, A_q) \leq f(H, A_q)$ holds if and only if $|V(H)|(2|V_{a_1}(H)| + 1) \leq |V_{a_1}(H)|^2 + |V_{a_2}(H)|^2$ holds.*

**Proof** We can clearly prove Corollary 1 from Lemma 3, □

From Lemma 3 and Corollary 1, several nodes in $N(H)$ can increase the attribute score function, even if the nodes have only a subset of $A_q$. Consequently, from Lemmas 1, 2, and 3, Theorem 1 can be theoretically derived.

**Theorem 1** Given subgraph $H$ and its 1-hop neighbor node set $N(H)$, $f(H, A_q)$ shows the largest score among all possible communities in $H \cup N(H)$ if the subgraph $H$ is marged with all nodes satisfying Lemmas 1 and 3 in $N(H)$.

**Proof** We can clearly prove Theorem 1 from Lemmas 1, 2, and 3 and Corollary 1. □

Theorem 1 implies that a set of nodes that maximizes the attribute score function can be directly found from a given $H \cup N(H)$.

#### 3.3.1 Algorithm

Based on the above properties, we design a fast enumeration algorithm for (Step 2) in Section 3.2. Algorithm 1 shows details of our algorithm. First, the fast enumeration algorithm obtains subgraph $H'$ from $V(H) \cup N(H)$ so that $f(H', A_q)$ is maximized (line 1). As we proved in Theorem 1, such subgraph $H'$ can be obtained by adding nodes in $N(H)$ into $H$ if the nodes satisfy Lemma 1 or Lemma 3. Then the algorithm adds all $(k, d)$-trusses composed of $H'$ into $C(H)$ (lines 4–9). Afterwards that the algorithm removes node $v \in V(H') \cap N(H)$ from $H'$ so that removing node $v$ decreases $f(H', A_q)$ based on Corollary 1 (lines 10–11). Finally, the algorithm terminates if $|C(H)|$ reaches $\beta$ (lines 2 and 6–8).

#### 3.3.2 Theoretical analysis

Finally, we theoretically assess the time complexity of the fast enumeration algorithm.

**Theorem 2** Given $H$ and $\beta$, the enumeration algorithm requires $\Omega(|N(H)| + \beta|E(G[N(H)])|^{1.5})$ time to find the top-$\beta$ candidates from $V(H) \cup N(H)$.

**Proof** As shown in Algorithm 1 (line 1), the algorithm obtains subgraph $H'$ by Theorem 1 before starting the while loop. This

**Table 2** Statistics of real-world datasets.

| Name | $\|V\|$ | $\|E\|$ | $\|A\|$ | Number of triangles | Fraction of triangles | $\|attr(V)\|$ |
|---|---|---|---|---|---|---|
| Cornell | 195 | 304 | 1,588 | 59 | 0.04 | 18,496 |
| Texas | 187 | 328 | 1,501 | 67 | 0.03 | 15,437 |
| Amazon | 335 K | 926 K | 157 | 667 K | 0.08 | 1,804,419 |
| YouTube | 1.10 M | 3.00 M | 5,327 | 3.0 K | 0.002 | 2,163,290 |

procedure requires $\Omega(|N(H)|)$ time since all nodes in $V(H)$ must be checked using Theorem 1. Afterwards, that our algorithm explores $(k, d)$-trusses that yield large scores of the attribute score function. In the worst case, the algorithm adds only a single $(k, d)$-truss to $C(H)$ in each while loop. That is, the while loop (lines 2–12) must be iterated $\Omega(\beta)$ time. In each while loop, the algorithm can find a $(k, d)$-truss, which incurs $\Omega(|N(H)|^{1.5})$ time [7], and it removes node $v$ from $H'$ in $O(1)$ time. Hence, the algorithm requires $\Omega(\beta|N(H)|)$ time. Therefore, Algorithm 1 incurs $\Omega(|N(H)| + \beta|E(G[N(H)])|^{1.5})$ time. □

Recall that the baseline algorithm requires $\Omega(2^{|V(H)\cup N(H)|}|E(G[N(H)])|^{1.5})$ time for each (Step 2). By contrast, our enumeration algorithm consumes $\Omega(|N(H)| + \beta|E(G[N(H)])|^{1.5})$ time which is clearly a smaller cost than the baseline. Thus, our enumeration algorithm can reduce the computational cost for the F-ATC problem.

### 3.4 Optimization by Preprocessing Phase

Here, we discuss a preprocessing optimization for further speeding up the query time of the proposed method. As shown in Algorithm 1, our fast enumeration method dynamically explores all $(*, d)$-trusses in the subgraph. Specifically, the $(k, d)$-truss search requires finding all the $k$-trusses in a subgraph and checking whether they are $(k, d)$-trusses or not. However, this is not efficient to enumerate the $k$-trusses located around the query node for each query. To avoid this expensive costs, we employ a simple preprocessing phase that enumerates all of the $k$-trusses included in the graph. By using state-of-the-art methods [20], we can perform all $k$-trusses enumeration in $O(|V|^{1.5})$ on a graph $G(V, E)$ even in $k \geq 3$. Once we perform this preprocessing phase, we can check whether a subgraph $H$ is $k$-truss in $O(1)$ for any queries.

## 4. Experimental Analysis

In this section, we experimentally discuss the effectiveness of our proposed algorithms. We designed our experiments to demonstrate that:

- **High accuracy:** Our proposed algorithms achieve higher community search accuracy than those of the state-of-the-art algorithm (LocATC) on real-world graphs.
- **High efficiency:** Although our fast enumeration algorithm outputs more accurate communities than LocATC, it outperforms LocATC and the baseline algorithm in terms of community search time on real-world graphs.

### 4.1 Experimental Setup

#### 4.1.1 Methods:

We compared our proposed algorithms (the baseline algorithm, the fast enumeration algorithm, and the fast enumeration algorithm with preprocessing) with the state-of-the-art algorithm LocATC [3]. As we described in Section 1, LocATC is the state-of-the-art community search method for attributed graphs. Given a user-specified query and parameters $k$ and $d$, LocATC finds a single $(k, d)$-truss that maximizes the attribute score function shown in Definition 3. In our experimental analysis, we used the $k = 4$, which is the default parameter in the original paper [3].

All algorithms were implemented in C++ and compiled with gcc-8.2.0 using the `-O3` option. All experiments were conducted on a server with an Intel Xeon CPU (3.50 GHz) and 128 GiB RAM. Here we report the average results of 100 queries.

#### 4.1.2 Datasets:

We used four real-world graphs, which were published in a previous study [3] and the SNAP repository [6]. **Table 2** shows their statistics. Since all datasets have ground-truth communities, they were used to evaluate the community search accuracy. Each node in Cornell and Texas has at least two node attributes. Because Amazon and YouTube do not provide node attributes to their nodes, we assigned synthetic attributes for each node by following the same method as the previous study [3]. Specifically, we assigned the synthetic attributes as follows:

- For each graph, we generated $|A| = 0.005|V|$ synthetic attributes.
- For each ground-truth community, we randomly selected three attributes in $A$, and assigned each one to 80% nodes in the community.
- To model noise attributes, we assigned randomly selected attributes to each node.

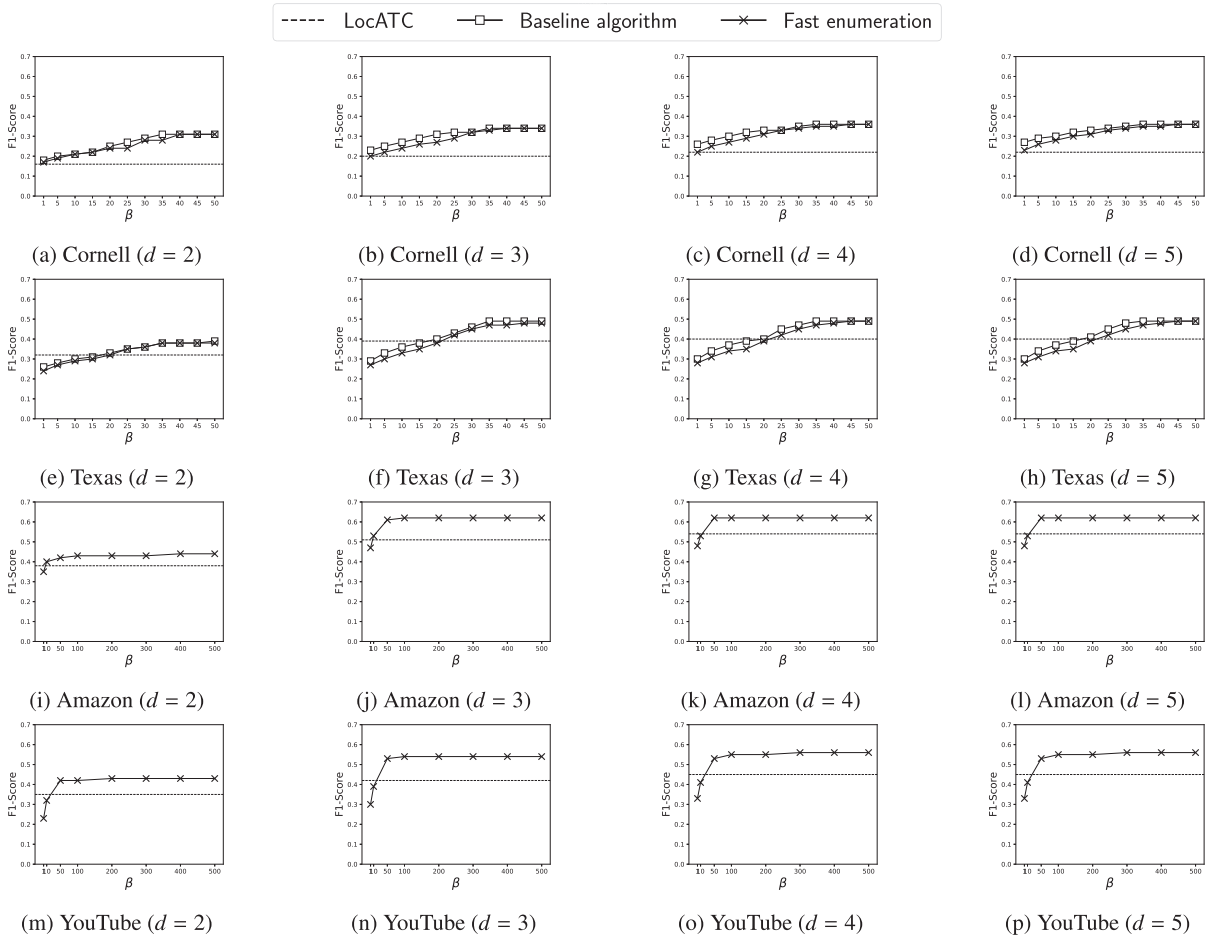
#### 4.1.3 Queries:

We generated 100 queries for each dataset by following the settings in the previous work [3]. Specifically, we randomly selected 100 query nodes from each graph. For each query node $v_q$, we set two attributes as query attributes $A_q$ using the most frequent attributes in the ground-truth community, including node $v_q$.

### 4.2 Accuracy

To assess whether the F-ATC problem achieves higher accuracy than LocATC, we evaluated the community search accuracy on real-world graphs. We compared the community search results with the ground-truth through F1-measure [8]. **Figure 2** shows the community search accuracy of each algorithm by varying the size of $d$ from 2 to 5. We also varied $\beta$ for our proposed algorithms since they require the beam width size $\beta$ for the beam searches. Note that the results of our baseline algorithm are omitted from Fig. 2 because it did not return any results within one hour on Amazon or YouTube.

Figure 2 shows that our proposed algorithms outperform LocATC in terms of the F1-score if the beam width sizes are large. Moreover, our proposed algorithms show higher accuracies than LocATC, except for Texas, even if the $\beta$ values are small. As we described in Section 1, LocATC assumes that real-world graphs contain a sufficient number of triangles although they can con-




(a) Cornell ($d = 2$) (b) Cornell ($d = 3$) (c) Cornell ($d = 4$) (d) Cornell ($d = 5$)

(e) Texas ($d = 2$) (f) Texas ($d = 3$) (g) Texas ($d = 4$) (h) Texas ($d = 5$)

(i) Amazon ($d = 2$) (j) Amazon ($d = 3$) (k) Amazon ($d = 4$) (l) Amazon ($d = 5$)

(m) YouTube ($d = 2$) (n) YouTube ($d = 3$) (o) YouTube ($d = 4$) (p) YouTube ($d = 5$)

**Fig. 2** F1-scores by varying $\beta$.

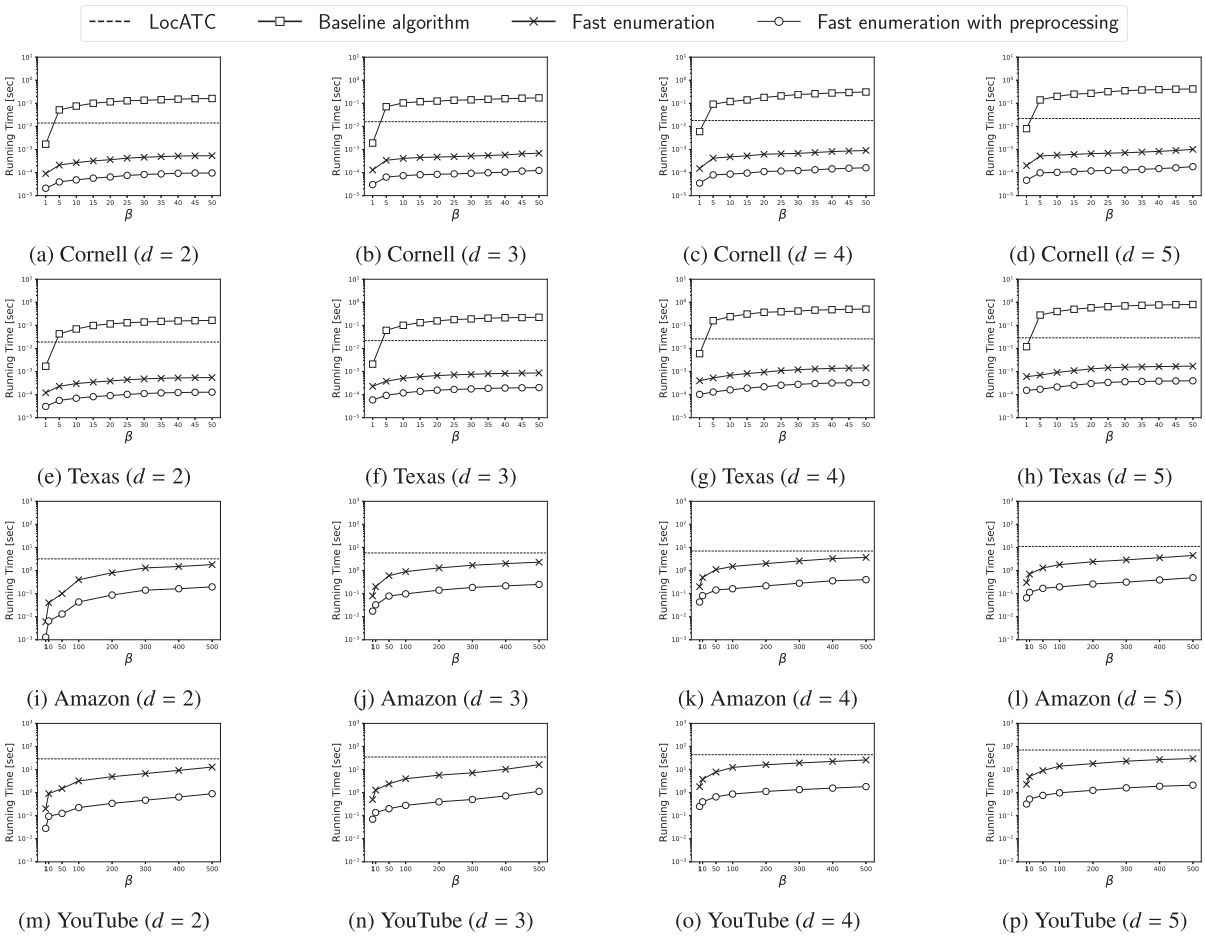
tain very diverse topological structures. Hence, LocATC fails to capture the ground-truth communities if those communities have a small number of triangles. By contrast, the F-ATC problem allows more diverse typologies along with the maximization of the attribute score function to be explored. Hence, our proposed method can achieve higher accuracy on a wider range of real-world graphs than LocATC. Specifically, our proposed method maintains higher accuracy than that of LocATC on YouTube even though the datasets have relatively smaller fractions of triangles than the others. These results imply that the F-ATC problem successfully captures diverse typologies in real-world graphs.

Figure 2 also indicates that the fast enumeration algorithm does not sacrifice the community search accuracy compared with the baseline algorithm. As theoretically discussed in Section 3.3, the fast enumeration algorithm can directly find a subgraph that maximizes the attribute score function (Theorem 1). Therefore, the fast enumeration algorithm does not degrade the community search accuracy compared with the baseline algorithm, which performs exhaustive searches.

### 4.3 Efficiency

We evaluated the community search time of each algorithm on four real-world datasets. Similar to the previous section, we varied beam width $\beta$ of our proposed methods for each $d$ setting. In this experiment, we also evaluated the effect of preprocessing (Section 3.4). **Figure 3** shows the community search time on the real-world datasets. The results for the baseline algorithm are omitted since it did not finish the community search on Amazon or YouTube within one hour.

The fast enumeration algorithm outperforms LocATC and the baseline algorithm under all examined conditions (Fig. 3). Although the baseline algorithm is 10 times slower than the query processing time of LocATC, the fast enumeration algorithm successfully mitigates the expensive enumeration cost in the baseline algorithm. In our experimental results, the fast enumeration algorithm has an improved speed up to three orders of magnitude higher than the baseline algorithm. Furthermore, the fast enumeration algorithm has up to 50 times faster query processing time than the state-of-the-art method LocATC. By comparing the running time among different parameter settings, the fast enumeration algorithm gradually increases the running time as the sizes of $\beta$ and $d$ increase. This is because the F-ATC problem requires a large number of community candidates to be searched if those parameters are large. However, the community search accuracy reaches a plateau on the real-world graphs, even if $\beta$ and $d$ are small (Fig. 2). For instance, our proposed method shows an almost constant accuracy on Amazon and YouTube at $\beta = 50$ and $d = 3$. Hence, the fast enumeration algorithm can reduce the running time while keeping its highly accurate community search results. In addition, the proposed method using preprocessing is up to 14 times faster than the original method. This is because fast enumeration with preprocessing does not need to compute




(a) Cornell ($d = 2$) (b) Cornell ($d = 3$) (c) Cornell ($d = 4$) (d) Cornell ($d = 5$)

(e) Texas ($d = 2$) (f) Texas ($d = 3$) (g) Texas ($d = 4$) (h) Texas ($d = 5$)

(i) Amazon ($d = 2$) (j) Amazon ($d = 3$) (k) Amazon ($d = 4$) (l) Amazon ($d = 5$)

(m) YouTube ($d = 2$) (n) YouTube ($d = 3$) (o) YouTube ($d = 4$) (p) YouTube ($d = 5$)

**Fig. 3** Query processing time by varying $\beta$.

**Table 3** Performance of preprocessing.

| Name | Preprocessing time [s] | Query time ratio |
|---|---|---|
| Cornell | 0.1 | x4.3 |
| Texas | 0.1 | x5.6 |
| Amazon | 28 | x9.2 |
| YouTube | 101 | x14.3 |

each subgraph dynamically is $(k, d)$-truss.

### 4.4 Performance of preprocessing

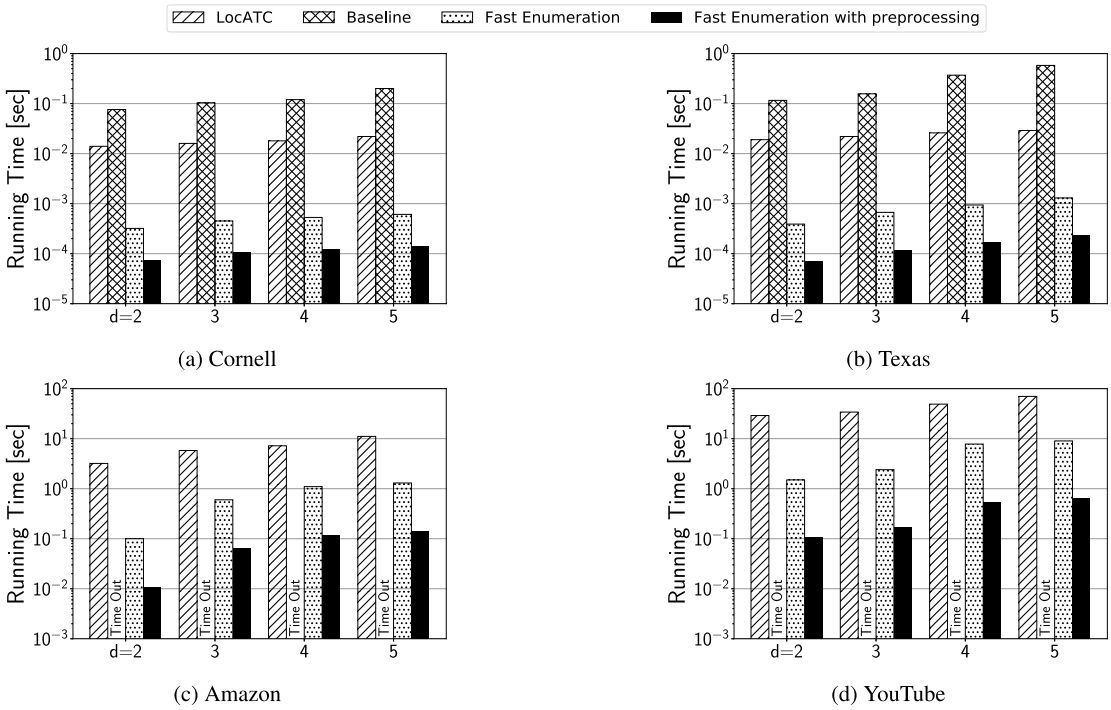
As discussed in Section 3.4, we focused on speeding up the query response and extended the proposed method by preprocessing. **Table 3** shows preprocessing time for enumerating $k$-truss. Query time ratio in Table 3 shows how many times faster the speed is compared to the fast enumeration algorithm without preprocessing. As we can see from the Table 3, the computation time required for preprocessing increases with graph size, but the benefit to query time also increases.

### 4.5 Peak Performance Analysis

The fast enumeration algorithm shows a trade-off between the community search accuracy and processing time. Thus, we discuss the peak query processing performance. In this evaluation, we compared the running time of our proposed algorithms with the best $\beta$ value, which returns the highest F1-score among all possible $\beta$ settings. **Figure 4** shows that our fast enumeration algorithm outperforms the community search time of LocATC for all settings. Specifically, our proposed method provides a community search that is up to 50 times faster than LocATC. Furthermore, by using preprocessing, our proposed method is up to 270 times faster than LocATC. Additionally, the fast enumeration algorithm outputs more accurate communities than LocATC in the case of the best $\beta$ settings. That is, these results imply that our proposed method achieves higher peak performances than LocATC on real-world graphs.

### 4.6 Comparison with the best $k$ in LocATC

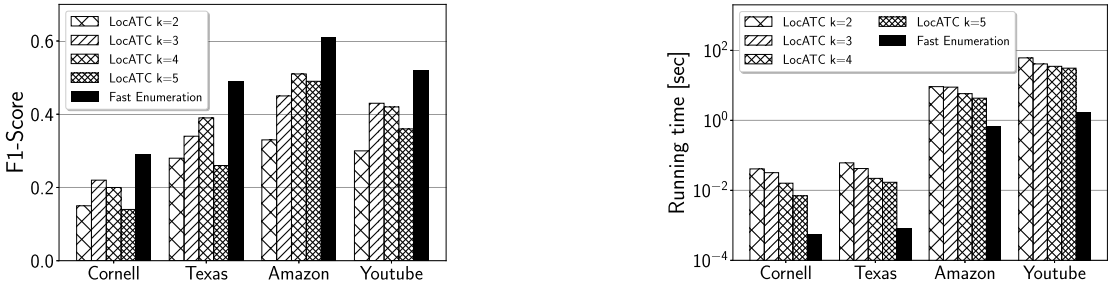
In this evaluation, we discuss accuracy and efficiency by varying $k$ in LocATC. We compared the performance of proposed method with LocATC ($k = 2, 3, 4$, and 5). Because the number of $(k, d)$-trusses in the dataset is significantlly small, we omitted the results of $k > 5$ from **Fig. 5**. We set $d = 3$, and $\beta$ is equal to the value at the peak size shown in Section 4.5. Figure 5 (a) demonstrates that our fast enumeration algorithm outperforms the LocATC in all $k$ settings. LocATC reaches the best accuracy if $k = 3$ or 4. However, as shown in Fig. 5 (b), it requires the largest running time. By contrast, our algorithm achieves faster search while keeping higher accuracy than LocATC ($k = 3, 4$). Specifically, our fast enumeration algorithm results at most 100 times faster than the most accurate setting of LocATC, which is about 10 points higher than the most accurate setting of LocATC.




(a) Cornell (b) Texas

(c) Amazon (d) YouTube

**Fig. 4** Query processing time at best beam width size $\beta$.




(a) Accuracy (b) Efficiency

**Fig. 5** Comparison of the performance of LocATC for all $k$ and the proposed method.

## 5. Related Work

Community search algorithms are fundamental tools to analyze complex data structures obtained from various applications [5], [18], [19]. Unlike traditional community detection algorithms [11], [15], [21], community search algorithms do not compute the entire given graph. Consequently, they efficiently find a community for the user-specified query. Here, we briefly review some of the more successful community algorithms.

Traditionally, community search algorithms are considered as a problem to detect cohesive communities that contain user-specified query nodes on non-attributed graphs. For example, Sozio and Gionis [18] designed a community search problem to find $k$-core that includes query nodes. Similarly, Huang et al. proposed the $k$-truss search algorithm to reveal the most relevant communities against a given query. Because these algorithms assume that the community has dense and robust inner-community connections, they perform local search methods to retrieve dense subgraphs (i.e., $k$-core and $k$-truss). However, these methods are designed for non-attributed graphs. Hence, they are unsuited to extract attribute-driven communities.

To overcome the above issue, Huang et al. recently proposed another class of the community search problem, namely the ATC problem [3]. The ATC problem is designed to find the $(k, d)$-truss, which is shown in Definition 2 that yields the largest attribute similarity with the query. Since the ATC problem is NP-hard, Huang et al. proposed the LocATC algorithm. This is the state-of-the-art algorithm to solve such a problem within a short running time. By introducing the ATC problem, LocATC can efficiently extract communities while ensuring a high cohesiveness and a high attribute similarity. However, LocATC assumes that each community has a sufficient number of triangles. This is unrealistic because real-world graphs have very diverse topological structures [6]. In this paper, we experimentally confirm that the accuracy of LocATC reaches a plateau if a given graph is sparse and has a small fraction of triangles. By contrast, our proposed algorithms overcome these performance limitations by relaxing the topological constraints. Consequently, our proposed method achieves faster community searches and higher accuracies than LocATC.

## 6. Conclusion

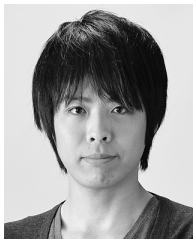
Herein we propose a novel community search problem called the F-ATC problem for attributed graphs. By relaxing the topological constraints of the community search, the F-ATC problem can explore divergent community structures included in real-world graphs. Because the F-ATC problem is NP-hard, we also present two heuristic algorithms based on the beam search to solve the F-ATC problem efficiently. Our experiments on real-world graphs demonstrate the advantages of our proposed algorithms compared to the state-of-the-art method.

**Acknowledgments** This work was supported by JSPS KAKENHI Early-Carrer Scientists Grant Number JP18K18057, and JST ACT-I.

### References

[1] Blondel, V., Guillaume, J., Lambiotte, R. and Mech, E.: Fast Unfolding of Communities in Large Networks, *Journal of Statistical Mechanics: Theory and Experiment*, Vol.2008, No.10, p.P10008 (2008).
[2] Huang, X., Cheng, H., Qin, L., Tian, W. and Yu, J.X.: Querying K-truss Community in Large and Dynamic Graphs, *Proc. SIGMOD 2014*, pp.1311–1322 (2014).
[3] Huang, X. and Lakshmanan, L.: Attribute-Driven Community Search, *PVLDB*, Vol.10, No.9, pp.949–960 (2017).
[4] Huang, X., Lakshmanan, L., Yu, J.X. and Cheng, H.: Approximate Closest Community Search in Networks, *PVLDB*, Vol.9, No.4, pp.276–287 (2015).
[5] King, A.D., Pržulj, N. and Jurisica, I.: Protein complex prediction via cost-based clustering, *Bioinformatics*, Vol.20, No.17, pp.3013–3020 (2004).
[6] Leskovec, J. and Krevl, A.: SNAP Datasets: Stanford Large Network Dataset Collection (2014), available from ⟨http://snap.stanford.edu/data⟩.
[7] Li, Z., Lu, Y., Zhang, W., Li, R., Guo, J., Huang, X. and Mao, R.: Discovering Hierarchical Subgraphs of K-Core-Truss, *Data Science and Engineering*, Vol.3, No.2, pp.136–149 (2018).
[8] Manning, C.D., Raghavan, P. and Schütze, H.: *Introduction to Information Retrieval*, Cambridge University Press (2008).
[9] Matsugu, S., Shiokawa, H. and Kitagawa, H.: Flexible Community Search Algorithm on Attributed Graphs, *Proc. 21st International Conference on Information Integration and Web-based Applications & Services, iiWAS 2019*, pp.103–109, ACM (2019).
[10] Matsugu, S., Shiokawa, H. and Kitagawa, H.: Fast and Accurate Community Search Algorithm for Attributed Graphs, *International Conference on Database and Expert Systems Applications, DEXA 2020*, pp.233–249, Springer (2020).
[11] Onizuka, M., Fujimori, T. and Shiokawa, H.: Graph Partitioning for Distributed Graph Processing, *Data Science and Engineering*, Vol.2, No.1, pp.94–105 (2017).
[12] Reddy, D.R.: Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort, Department of Computer Science, Technical Report, Carnegie-Mellon University (1977).
[13] Sato, T., Shiokawa, H., Yamaguchi, Y. and Kitagawa, H.: FORank: Fast ObjectRank for Large Heterogeneous Graphs, *Companion Proc. Web Conference 2018*, *WWW '18*, Republic and Canton of Geneva, CHE, International World Wide Web Conferences Steering Committee, pp.103–104 (2018).
[14] Shiokawa, H., Amagasa, T. and Kitagawa, H.: Scaling Fine-grained Modularity Clustering for Massive Graphs, *Proc. 28th International Joint Conference on Artificial Intelligence, IJCAI-19*, pp.4597–4604 (2019).
[15] Shiokawa, H., Fujiwara, Y. and Onizuka, M.: Fast Algorithm for Modularity-Based Graph Clustering, *Proc. 27th AAAI Conference on Artificial Intelligence* (*AAAI 2013*), pp.1170–1176 (2013).
[16] Shiokawa, H., Fujiwara, Y. and Onizuka, M.: SCAN++: Efficient Algorithm for Finding Clusters, Hubs and Outliers on Large-scale Graphs, *PVLDB*, Vol.8, No.11, pp.1178–1189 (2015).
[17] Shiokawa, H., Takahashi, T. and Kitagawa, H.: ScaleSCAN: Scalable Density-based Graph Clustering, *Proc. 29th International Conference on Database and Expert Systems Applications*, pp.18–34, DEXA (2018).
[18] Sozio, M. and Gionis, A.: The Community-Search Problem and How to Plan a Successful Cocktail Party, *Proc. KDD 2010*, pp.939–948 (2010).
[19] Takahashi, T., Shiokawa, H. and Kitagawa, H.: SCAN-XP: Parallel Structural Graph Clustering Algorithm on Intel Xeon Phi Coprocessors, *Proc. 2nd International Workshop on Network Data Analytics* (*NDA*), pp.6:1–6:7 (2017).
[20] Wang, J. and Cheng, J.: Truss Decomposition in Massive Networks, *Proc. VLDB Endowment*, Vol.5 (2012).
[21] Zhang, X. and Newman, M.E.J.: Multiway Spectral Community Detection in Networks, *Physical Review E*, Vol.92, p.052808 (2015).
[22] Zhou, Y., Cheng, H. and Yu, J.X.: Graph Clustering Based on Structural/Attribute Similarities, *PVLDB*, Vol.2, No.1, pp.718–729 (2009).

**Shohei Matsugu** received B.S. in engineering from University of Tsukuba in 2019. He is currently a Master student in University of Tsukuba. His research interests include large-scale graph data analysis.



**Hiroaki Shiokawa** is an Associate Professor at University of Tsukuba. He received B.S., M.E., and Ph.D. in engineering from University of Tsukuba in 2009, 2011 and 2015, respectively. From 2011 to 2015, he was a research scientist at Nippon Telegraph and Telephone Corporation, and he joined Center for Computational Sciences at University of Tsukuba in Nov. 2015. His current research interests include database systems, data engineering, data mining, and graph data management.

**Hiroyuki Kitagawa** received his B.Sc. degree in physics and his M.Sc. and Dr.Sc. degrees in computer science, all from the University of Tokyo. He is currently a full professor at Center for Computational Sciences and Center for Artificial Intelligence Research, University of Tsukuba. His research interests include databases, data integration, data mining, information retrieval, and data engineering applications. He served as President of the Database Society of Japan from 2014 to 2016. He is an IEICE Fellow, an IPSJ Fellow, an Associate Member of the Science Council of Japan, and a member of ACM, IEEE, JSST.

(Editor in Charge: *Minoru Sasaki*)