Query-centered Fairness-aware Maximum Clique Search in Dynamic Attributed Graphs

Jingwen Chen[1], Jianuo Xu[1], Xinrui Wang[1]✉, Hong Gao[2], and Dongxiao Yu[1]

[1] Shandong University, Qingdao 266237,China
jwchen_sdu@126.com,jnxu@mail.sdu.edu.cn,{xrwang,dxyu}@sdu.edu.cn
[2] Zhejiang Normal University, Jinhua 321004, China
honggao@zjnu.edu.cn

Abstract. On attributed graphs, fairness-aware maximal clique models which consider the fairness of members' attributes in a clique, have a wide range of practical applications. Existing work has proposed efficient algorithms to enumerate all maximal fair cliques in static attributed graphs. However, the real-world graphs are often dynamic, including the insertion and deletion of vertices and edges over time. But existing work ignored how to efficiently search fair cliques in dynamic attributed graphs. In this paper, we focus on how to efficiently search maximum fair cliques which contain the given query vertex in a dynamic attributed graph. Firstly, we formalize the problem of query-centered maximum fair clique search in a dynamic attributed graph which is NP-hard. Then we give a basic algorithm to recalculate maximum fair cliques in the pruned search space each time inserting or deleting one edge. Furthermore, we develop two incremental algorithms with pruning and early termination strategies, both can avoid recalculating all results from scratch, and greatly reduce the time of searching maximum fair cliques each time inserting or deleting one edge. Extensive experiments on 6 real-world graphs show extremely higher efficiency and better scalability of our incremental algorithms compared to the basic algorithm.

Keywords: Dynamic Attributed Graphs · Fairness · Maximum Clique Search · Incremental Algorithms.

1 Introduction

Community search, which aims at finding cohesive subgraphs containing the given query vertices, has been extensively studied in graph mining[10,26]. In decades, various kinds of models have been proposed to describe cohesive subgraphs, like k-core[2], k-truss[6], and cliques[5]. Among them, cliques, as an elementary model, have a wide range of applications in reality, such as describing Co-Expression Groups in gene co-expression networks[25], abnormal transactions groups in financial networks[3], and geographic regions containing houses in close proximity in transportation networks[12].

In recent years, many researchers have concerned the issue of fairness in machine learning [15,18], in order to eliminate the unfair factors (e.g. gender and age) caused by group difference[18], opportunity distribution[15] and so on. Moreover, fairness has attracted much attention in graph mining, like node embeddings[4], pagerank[23], and graph clustering[14].

Recently, Pan et al.[17] combined fairness with the clique model, proposing two kinds of maximal fair cliques. Specifically, given an attributed graph (where each vertex has an attribute value), and a fairness threshold k, **a weak fair clique** is a maximal clique that the number of its vertices for each attribute value is no less than k (e.g. in Fig.1(a), $\{q, v1, v2, v3, v4\}$ is a weak fair clique with $k = 2$). Furthermore, besides the conditions of the weak fair clique, **a strong fair clique** also needs to satisfy that the number of its vertices for each attribute value is exactly the same (e.g. in Fig.1(a), $\{q, v1, v2, v4\}$ is a strong fair clique with $k = 2$). Mining fair cliques has many applications. E.g., in social networks like Facebook which contain users and their friendship (users' attribute is their gender), a fair clique is a well-connected group of users with no gender discrimination. In bibliographic networks like DBLP which contain authors and their co-authorship relations (authors' attribute is their research areas), a fair clique is a group of experts with close cooperation and diverse research areas.

However, the real-life graphs are usually dynamically changing with edges and vertices insertion or deletion[7,24]. But Pan et al.[17] do not consider how to efficiently find fair cliques in dynamic attributed graphs. Moreover, the fair cliques which contain the given query vertices and have bigger size often play important roles in the real world. Thus, in this paper, as far as we know, we are the first to focus on how to efficiently search maximum fair cliques which contain the given query vertex in a dynamic attributed graph. Specifically, we formalize the problem of **query-centered maximum fair cliques search (MFCS) in a dynamic attributed graph** (i.e., MFCS-OneEdgeInsertion and MFCS-OneEdgeDeletion), considering the scenario of inserting or deleting one edge each time the graph G_{i-1} turns to G_i respectively.

Challenges and Contributions. The MFCS-OneEdgeInsertion and MFCS-OneEdgeDeletion problems are NP-hard. We first present a basic algorithm to recalculate maximum fair cliques each time G_{i-1} turns to G_i. Then we propose two incremental algorithms to avoid recalculating all results from scratch when G_{i-1} turns to G_i. In sum, the main contributions of this paper are as follows.

(1) We formalize the MFCS-OneEdgeInsertion and MFCS-OneEdgeDeletion problems and show they are NP-hard.

(2) We propose a **Basic** algorithm to recalculate the maximum fair cliques each time G_{i-1} turns to G_i by inserting or deleting one edge. We also present two pruning strategies to reduce the time of searching maximal fair cliques.

(3) For the MFCS-OneEdgeInsertion problem, we propose the incremental algorithm **MFCSEI** which only needs to search new maximal fair cliques generated after inserting an edge, rather than recalculate all results in G_i from scratch. For the MFCS-OneEdgeDeletion, we present the incremental algorithm **MFCSED**, using early termination strategies to obtain G_i's results directly from G_{i-1}'s results so as to avoid searching maximal fair cliques. When search-

ing maximal fair cliques can not be avoided, we also develop pruning strategies to reduce the search time for strong fair cliques.

(4) We conduct extensive experiments on 4 real-world graphs to evaluate the efficiency and effectiveness of our algorithms. The results indicate that our MFCSEI and MFCSED algorithms significantly reduce the running time and have better scalability compared to the Basic algorithm.

2 Related Work

Clique Computation. Clique computation (e.g., identifying maximal/maximum cliques), as a foundational NP-hard problem in graph theory, has been extensively studied in decades[5,24,27]. Early classic algorithms are based on backtracking methods with pruning strategies[5,9,22]. Besides, to handle large sparse graphs, researchers have developed parallelized implementations and near-optimal algorithms for clique computation[8,20]. Moreover, since the real-life graphs are usually dynamically changing, researchers have proposed incremental algorithms and heuristic methods for clique computation in dynamic graphs[7,21,24]. However, none of the above works have simultaneously considered the dynamic changes of the graphs and the fairness constraints of the cliques.

Fairness-aware Data Mining. Concerning fairness (i.e., eliminating unfair factors) in machine learning has attracted much attention in recent years[15,18,19]. Various definitions of fairness (e.g., group fairness and individual fairness) have been proposed in many practical applications[15,18]. For example, in recommendation systems, researchers focus on fair ranking to ensure equal item exposure and balanced provider representation[15,16]. In classification tasks, many fairness metrics and algorithms have been developed to achieve equal classification probabilities between protected and unprotected groups[18,19]. Recently, numerous work has considered fairness in graph mining, like node embeddings[4], pagerank[23], and graph clustering[14]. The work most related to our work is fair clique enumeration on attributed graphs[17], which considers the fairness of members' attributes in a clique. However, their algorithms[17] can not efficiently handle the case of dynamic attributed graphs whose edges and vertices constantly inserting or deleting over time.

3 Preliminaries

A *static attributed graph* $G = (V, E, A, \phi)$ is a simple undirected graph, where V (E resp.) is the vertex (edge resp.) set, A is the attribute value set, and $\phi : V \to A$ is a mapping function so that each vertex $v \in V$ has a particular attribute value $a_r \in A$ (i.e., $\phi(v) = a_r, 1 \leq r \leq |A|$). The neighbor set of a vertex $u \in V$ in G is denoted as $Nei(u, G)$. And the degree of u is denoted as $Deg(u, G) = |Nei(u, G)|$. In G, a subgraph induced by a vertex subset $V_H \subseteq V$ is defined as $H = (V_H, E_H, A, \phi)$ where $E_H = \{(u, v)|u, v \in V_H, (u, v) \in E\}$. Let $Ego(q) = (V', E', A, \phi)$ be the ego-network of a vertex q in G, where $V' = \{q \cup Nei(q, G)\}$, $E' = \{(u, v)|u, v \in V', (u, v) \in E\}$.

In a graph G, a *clique* is a complete subgraph where there is an edge between each pair of vertices in the clique. In an attributed graph, the maximal weak/strong fair clique is defined as follows.

Definition 1. *(Maximal Weak/Strong Fair Clique[17]) Given an attributed graph G and the fairness threshold k, a maximal weak fair clique C in G satisfies that: (1) for each attribute value $a_r \in A$, the number of vertices whose attribute value is a_r is no less than k; (2) there is no clique $C' \supset C$ such that C' satisfies (1). And if in condition (1), C also satisfies that the number of vertices for each attribute value a_r is exactly the same, C is a maximal strong fair clique.*

In an attributed graph G, a *maximum weak/strong fair clique* is a maximal weak/strong fair clique which has the largest size among all weak/strong maximal fair cliques in G.

In reality, attributed graphs are constantly changing over time (called *dynamic attributed graphs*), including the insertion and deletion of vertices and edges. We define a dynamic attributed graph as $\mathcal{G} = (G_0, \varDelta)$, where $G_0 = G$ is the initial graph, and $\varDelta = \{(t_1, o_1), (t_2, o_2), \dots, (t_i, o_i), \dots\}$ is a sequence of update operations to G_0. In specific, (t_i, o_i) represents that at the timestamp t_i, G_{i-1} turns into G_i by the update operation o_i (including the insertion and deletion of the vertices and edges).

In the following, we formalize the query-centered maximum fairness clique search problems in a dynamic attributed graph (i.e., MFCS-OneEdgeInsertion and MFCS-OneEdgeDeletion), considering the scenario of inserting or deleting one edge each time respectively. Our algorithms proposed in this paper can directly apply to the scenario of inserting or deleting multiple edges each time (i.e., invoking the algorithms multiple times). And inserting or deleting one vertex can be directly seen as inserting or deleting multiple edges and an isolated vertex[1][11]. We leave the study of advanced algorithms for handling the scenario of inserting or deleting one or multiple vertices in our future work.

Problem Statement (MFCS-OneEdgeInsertion/Deletion). Given a dynamic graph $\mathcal{G}$, the query vertex q, and the fairness threshold k, each time when G_{i-1} turns into G_i by inserting/deleting one edge (u_i, v_i) (i.e., $o_i = (u_i, v_i)^+$ or $o_i = (u_i, v_i)^-$), find the maximum weak (strong resp.) fair clique C_i^w (C_i^s resp.) containing q in G_i.

In reality, there might exist more than one maximum weak/strong fair cliques containing q which all have the same size in G_i. In such cases, the set of all those maximum weak/strong fair cliques (denoted as $\mathcal{C}_i^w$ or $\mathcal{C}_i^s$) should be outputted.

Problem Hardness. Finding maximal weak/strong fair cliques in static attributed graph is NP-hard[17], thus, the problems of MFCS-OneEdgeInsertion and MFCS-OneEdgeDeletion studied in this paper are also NP-hard.

We also give several important definitions and lemmas which will be used in the rest of this paper as follows.

Lemma 1. *Given an attributed graph $G = (V, E, A, \phi)$, a fairness threshold k, and a query vertex q, a fair clique containing q must be in the ego-network of q.*

Definition 2. *(Vertex Coloring[13]) Given a simple graph $G = (V, E)$, vertex coloring is to color the vertices in G such that no two adjacent vertices have the same color, and the color of a vertex $u \in V$ is denoted as $color(u)$.*

Definition 3. *(Colorful Degree[17]) Given an attributed graph $G = (V, E, A, \phi)$ and an attribute value $a_r \in A$, the colorful degree of a vertex $u \in V$ based on a_r, denoted as $D_{a_r}(u, G)$, is the number of colors of u's neighbors whose attribute values are all a_r, i.e. $D_{a_r}(u, G) = |\{color(v)|v \in Nei(u), \phi(v) = a_r\}|$. Let $D_{min}(u, G)$ be the minimum colorful degree of a vertex u, i.e., $D_{min}(u, G) = min\{D_{a_r}(u, G)|a_r \in A\}$.*

For example, in Fig.1 (a), $D_a(v2, G) = 3$, $D_b(v2, G) = 1$, $D_{min}(v2, G) = 1$.

Definition 4. *(Colorful k-core[17]) Given an attributed graph $G = (V, E, A, \phi)$ and a fairness threshold k, a subgraph $H = (V_H, E_H, A, \phi)$ is a colorful k-core if (1) $D_{min}(u, H) \geq k$ for each $u \in V_H$, (2) there is no $H' \supset H$ satisfying (1).*

Lemma 2. *Given an attributed graph $G = (V, E, A, \phi)$ and a fairness threshold k, a fair clique must be in the colorful (k-1)-core of G.[17]*

4 Basic Algorithm

In this section, we present a **Basic** algorithm to constantly search maximum weak/strong fair cliques containing the query vertex q in a dynamic attributed graph: each time G_{i-1} turns to G_i (inserting or deleting an edge (u_i, v_i)), it recalculates $\mathcal{C}_i^w$ and $\mathcal{C}_i^s$ in G_i from scratch.

Algorithm 1: Basic$(G_{i-1}, (u_i, v_i), k, q)$

```
$\mathcal{C}_i^w \leftarrow \emptyset$, $\mathcal{C}_i^s \leftarrow \emptyset$, $Size_{C_w} \leftarrow 0$, $Size_{C_s} \leftarrow 0$, $size \leftarrow 0$;
$G_i \leftarrow (G_{i-1}, (u_i, v_i))$; Compute $Ego_i(q)$ in $G_i$;
$\widetilde{Ego}_i(q)$ =**ColorfulCore**($Ego_i(q), k$)[17];            //By **Lemma** 2;
**for** *each* $x \in V_{\widetilde{Ego}_i(q)}$ *and* $x \neq q$ **do**
    $P \leftarrow \{q, x\}$;
    $I \leftarrow \{y|y \in Nei(q, \widetilde{Ego}_i(q)) \cap Nei(x, \widetilde{Ego}_i(q))\ , ID(y) > ID(x)\}$;
    $Size_{C_w} \leftarrow 2$, $Size_{C_s} \leftarrow 2$, $size \leftarrow 2$;
    **DeepSearch**($size + 1, I, P$);
**return** $\mathcal{C}_i^w$, $\mathcal{C}_i^s$;

**Procedure** ***DeepSearch***$(sizetemp, I, P)$
    **for** *each* $y \in I$ **do**
        $P' \leftarrow P \cup \{y\}$, $I' \leftarrow \{z|z \in Nei(y, \widetilde{Ego}_i(q)) \cap I, z > y\}$;
        **if** $|P'| + |I'| < k \times |A|$ **then** continue;            //By **Pruning Strategy 1**;
        **if** $|P'| + |I'| < min\{Size_{C_w}, Size_{C_s}\}$ **then** continue;//By **Pruning Strategy 2**;
        **if** $P'$ *is a weak fair clique* **then**
            **if** $sizetemp > Size_{C_w}$ **then** $\mathcal{C}_i^w \leftarrow \{P'\}, Size_{C_w} \leftarrow sizetemp$;
            **if** $sizetemp = Size_{C_w}$ **then** $\mathcal{C}_i^w \leftarrow \mathcal{C}_i^w \cup \{P'\}$;
        **if** $P'$ *is a strong fair clique* **then**
            **if** $sizetemp > Size_{C_s}$ **then** $\mathcal{C}_i^s \leftarrow \{P'\}, Size_{C_s} \leftarrow sizetemp$;
            **if** $sizetemp = Size_{C_s}$ **then** $\mathcal{C}_i^s \leftarrow \mathcal{C}_i^s \cup \{P'\}$;
        **DeepSearch**($sizetemp + 1, I', P'$);
```

The **Basic** algorithm contains two steps to find all maximum weak/strong fair cliques $\mathcal{C}_i^w$ ($\mathcal{C}_i^s$ resp.) in G_i.

Firstly, prune the search space in G_i: compute $Ego_i(q)$ in G_i, and then compute the colorful $(k-1)$-core $\widetilde{Ego_i}(q)$ in $Ego_i(q)$ by invoking the procedure **ColorfulCore** [17]. Based on Lemma 1 and Lemma 2, $\widetilde{Ego_i}(q)$ contains all maximal fair cliques containing q.

Secondly, conduct DFS from q to find all maximal weak/strong fair cliques containing q in the pruned search space $\widetilde{Ego_i}(q)$, and use the set $\mathcal{C}_i^w$ or $\mathcal{C}_i^s$ to maintain the maximum weak/strong fair cliques containing q.

During the DFS, let $Size_{C_w}$ or $Size_{C_s}$ be the size of the current maximum weak/strong fair clique maintained in $\mathcal{C}_i^w$ or $\mathcal{C}_i^s$. The detailed phases of the second step are as follows:

(1) Treat q and each vertex in $V_{\widetilde{Ego_i}(q)}$ as an initial clique $P = \{q, x\}$. Compute the intersection I of the neighbors of all vertices in P. By the clique definition, a candidate vertex which can join P to form a clique larger than P must be the neighbor of all vertices in P. So I contains all those candidate vertices.

(2) In the procedure **DeepSearch**, add each vertex $y \in I$ to P to form a larger clique P'. Compute the intersection I' of the neighbors of all vertices in P'. Check whether to stop expanding P' by the following two pruning strategies: If $|P'|+|I'| < k \times |A|$ (**Pruning Strategy 1**), or $|P'|+|I'| < min\{Size_{C_w}, Size_{C_s}\}$ (**Pruning Strategy 2**), stop expanding P', re-consider a new $y \in I$. If P' can continue to expand, check whether P' is weak/strong fair. If yes, compare the size of P' with the size $Size_{C_w}$ or $Size_{C_s}$ of the current maximum weak/strong fair clique maintained in $\mathcal{C}_i^w$ or $\mathcal{C}_i^s$. If P' has the larger size, P' is the only new result, so $\mathcal{C}_i^w = \{P'\}$ *or* $\mathcal{C}_i^s = \{P'\}$. If P' has the same size, besides the results previously maintained in $\mathcal{C}_i^w$ or $\mathcal{C}_i^s$, P' is a new result which should also be added into $\mathcal{C}_i^w$ or $\mathcal{C}_i^s$. Then update $Size_{C_w}$ or $Size_{C_s}$.

(3) Repeat (1)-(2) until all vertices in $V_{\widetilde{Ego_i}(q)}$ are considered.

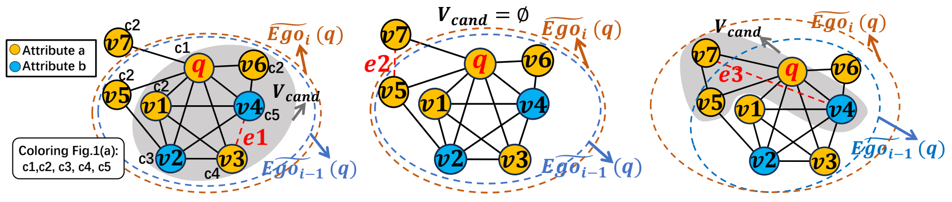
Theorem 1. *Pruning Strategies 1 and 2 are correct.*

Proof. During the DFS, the size of the maximal fair clique expanded from P' can not be larger than $|P'|+|I'|$. For Pruning Strategy 1, a fair clique with a fairness threshold k in G_i contains at least $k \times |A|$ vertices. If $|P'|+|I'| < k \times |A|$, it means that even $P' \cup I'$ is a clique, it will never be fair. Thus, P' can stop expanding. For Pruning Strategy 2, if $|P'| + |I'| < min\{Size_{C_w}, Size_{C_s}\}$, it means that even $P' \cup I'$ is a clique, its size is still smaller than the current maximum clique containing q. Thus, P' can stop expanding. ■

Complexity Analysis. Let $|V'|$ ($|E'|$ resp.) be the number of vertices (edges resp.) in $\widetilde{Ego_i}(q)$, and d be the max recursion depth. The worst time complexity of **Basic** is $O(|V'| + |E'| + |V'|^3 \times d)$.

5 Incremental Algorithm

In this section, we propose two incremental algorithms **MFCSEI** and **MFCSED**, both of which can avoid recalculating all results from scratch when G_{i-1} turns to G_i (inserting or deleting one edge (u_i, v_i)), and also can greatly reduce the



(a) Inserting e1 (b) Inserting e2 (c) Inserting e3

Fig. 1: Illustration Examples in MFCSEI

Algorithm 2: MFCSEI$(G_{i-1}, (u_i, v_i)^+, k, q, \mathcal{C}^w_{i-1}, \mathcal{C}^s_{i-1}, \widetilde{Ego}_{i-1}(q), Ego_{i-1}(q))$

```
1  $Size_{C_w} \leftarrow |C_w|(C_w \in \mathcal{C}^w_{i-1}),\ Size_{C_s} \leftarrow |C_s|(C_s \in \mathcal{C}^s_{i-1}),\ size \leftarrow 0$;
2  $\mathcal{C}^w_i \leftarrow \mathcal{C}^w_{i-1},\ \mathcal{C}^s_i \leftarrow \mathcal{C}^s_{i-1}$; Compute $Ego_i(q)$;
3  if $(u_i, v_i \notin V_{Ego_{i-1}(q)})$ or $(u_i \neq q, v_i \notin V_{Ego_{i-1}(q)})$ or $(v_i \neq q, u_i \notin V_{Ego_{i-1}(q)})$ then
4      return $\mathcal{C}^w_{i-1}, \mathcal{C}^s_{i-1}$;
5  $\widetilde{Ego}_i(q) =$ ColorfulCore$(Ego_i(q), k)$[17];                    //By Lemma 2;
6  if $u_i, v_i \in V_{\widetilde{Ego}_{i-1}(q)}$ then
7      $V_{cand} \leftarrow \{u_i, v_i\}$; $V_{cand} \leftarrow V_{cand} \cup \{Nei(V_{cand}, \widetilde{Ego}_i(q))\}$
8  else
9      $V_{cand} \leftarrow V_{\widetilde{Ego}_i(q)} \setminus V_{\widetilde{Ego}_{i-1}(q)}$; $V_{cand} \leftarrow V_{cand} \cup \{Nei(V_{cand}, \widetilde{Ego}_i(q))\}$;
10 if $V_{cand} = \emptyset$ then return $\mathcal{C}^w_{i-1}, \mathcal{C}^s_{i-1}$;
11 for each $w \in V_{cand}$ and $w \neq q$ do
12     $P \leftarrow \{q, w\}$; $I \leftarrow \{x | x \in Nei(q, \widetilde{Ego}_i(q)) \cap Nei(w, \widetilde{Ego}_i(q)) \cap V_{cand}, ID(x) > ID(w)\}$;
13     $size \leftarrow 2$;
14     DeepSearch$(size + 1, I, P)$;
15 return $\mathcal{C}^w_i, \mathcal{C}^s_i$;
```

time of searching maximal fair cliques in G_i which is the bottleneck of the **Basic** algorithm. **MFCSEI** (**MFCSED** resp.) handles the case of inserting (u_i, v_i) (deleting (u_i, v_i) resp.).

5.1 One Edge Insertion

Although the **Basic** algorithm has pruned the maximal fair clique search space from G_i to $\widetilde{Ego}_i(q)$, it is still quite large so that the second step of **Basic** would spend too much time on searching maximal fair cliques.

To greatly reduce the total time for solving the MFCS-OneEdgeInsertion problem, we propose an incremental algorithm **MFCSEI**: by maintaining the results in G_{i-1}, it only needs to search new maximal fair cliques which are generated after inserting (u_i, v_i) in a search space much smaller than $\widetilde{Ego}_i(q)$, rather than recalculate all results in G_i from scratch.

The **MFCSEI** algorithm contains the following five steps.

(1) Maintain the results in G_{i-1}, including $\mathcal{C}^w_{i-1}$, $\mathcal{C}^s_{i-1}$, and $\widetilde{Ego}_{i-1}(q)$. Note that after inserting (u_i, v_i), in the following cases: a) $u_i, v_i \notin V_{Ego_{i-1}(q)}$,b) $u_i \neq q, v_i \notin V_{Ego_{i-1}(q)}$, or c) $v_i \neq q, u_i \notin V_{Ego_{i-1}(q)}$, it has $Ego_i(q) = Ego_{i-1}(q)$ which

means that the neighbors of q do not change, so that the results in G_i would be the same as those in G_{i-1}. In the above cases, directly output $\mathcal{C}^w_{i-1}$, $\mathcal{C}^s_{i-1}$ as the results in G_i.

(2) Compute the colorful $(k-1)$-core $\widetilde{Ego}_i(q)$ in $Ego_i(q)$.

(3) Compute the set of candidate vertices V_{cand} which might form new maximal fair cliques after inserting (u_i, v_i): If $u_i, v_i \in V_{\widetilde{Ego}_{i-1}(q)}$, only u_i, v_i and their neighbors in $\widetilde{Ego}_i(q)$ might form new maximal fair cliques (e.g., in Fig. 1(a), $V_{cand} = \{q, v1, v2, v3, v4, v6\}$). Otherwise, the vertices in $\widetilde{Ego}_i(q)$ but not in $\widetilde{Ego}_{i-1}(q)$, as well as their neighbors in $\widetilde{Ego}_i(q)$, might form new maximal fair cliques (e.g., in Fig. 1(c),$V_{cand} = \{q, v4, v5, v7\}$).

(4) If V_{cand} is empty (i.e., $\widetilde{Ego}_i(q) = \widetilde{Ego}_{i-1}(q)$) which means that there exist no candidate vertices which could form new maximal fair cliques, then directly output $\mathcal{C}^w_{i-1}$, $\mathcal{C}^s_{i-1}$ as the results in G_i. E.g., in Fig. 1(b), $V_{cand} = \emptyset$.

(5) While if V_{cand} is not empty, treat q and each vertex w in V_{cand} as an initial clique $P = \{q, w\}$, then perform DFS to find new maximal fair cliques in $G_{V_{cand}}$ by **DeepSearch**. If a new maximal fair clique has the same or larger size than the current maximum fair cliques in $\mathcal{C}^w_i$ or $\mathcal{C}^s_i$, update $\mathcal{C}^w_i$ or $\mathcal{C}^s_i$.

Complexity Analysis. The worst time complexity of **MFCSEI** is $O(|V'| + |E'| + |V_{cand}|^3 \times d)$. In reality, $|V_{cand}|$ is usually much smaller than $|V'|$, so **MFCSEI** is quite more efficient than **Basic**.

5.2 One Edge Deletion

In order to avoid recalculating all results from scratch after deleting (u_i, v_i) from G_{i-1}, we propose an incremental algorithm **MFCSED** to solve the MFCS-OneEdgeDeletion problem by reusing G_{i-1}'s results as much as possible.

The basic idea of **MFCSED** is that by analyzing G_{i-1}'s results which would be affected after deleting (u_i, v_i) from G_{i-1}, early termination strategies are given so as to obtain G_i's results directly from G_{i-1}'s results, and simultaneously avoid searching maximal fair cliques which is the bottleneck of the **Basic** algorithm. Moreover, in some cases where early termination strategies can not be satisfied, searching maximal fair cliques can not be avoided, then we develop pruning strategies to reduce the search time for strong fair cliques.

For G_{i-1}'s results $\mathcal{C}^w_{i-1}$ and $\mathcal{C}^s_{i-1}$, let $\mathcal{C}^w_{i-1}[x]$ ($\mathcal{C}^s_{i-1}[x]$ resp.) be the set of maximum weak (strong resp.) fair cliques in $\mathcal{C}^w_{i-1}$ ($\mathcal{C}^s_{i-1}$ resp.) which contain the vertex x. That is, $\mathcal{C}^w_{i-1}[x] = \{C_w \mid C_w \in \mathcal{C}^w_{i-1}, x \in C_w\}$, $\mathcal{C}^s_{i-1}[x] = \{C_s \mid C_s \in \mathcal{C}^s_{i-1}, x \in C_s\}$. In other words, $\mathcal{C}^w_{i-1}[x]$ and $\mathcal{C}^s_{i-1}[x]$ record those G_{i-1}'s results which would be affected after deleting x from G_{i-1}. E.g., in Fig. 2(a), before deleting $e1$, $\mathcal{C}^w_0 = \{C1, C2\}$, then $\mathcal{C}^w_0[q] = \{C1, C2\}$, $\mathcal{C}^w_0[v2] = \{C2\}$.

The **MFCSED** algorithm contains the following three steps.

(1) Compute the intersection $InterW = \mathcal{C}^w_{i-1}[u_i] \cap \mathcal{C}^w_{i-1}[v_i]$, $InterS = \mathcal{C}^s_{i-1}[u_i] \cap \mathcal{C}^s_{i-1}[v_i]$. If both $InterW$ and $InterW$ are empty, which means that no G_{i-1}'s results (i.e., maximum weak or strong fair cliques in $\mathcal{C}^w_{i-1}$ or $\mathcal{C}^s_{i-1}$) would be affected after deleting (u_i, v_i) from G_{i-1}. Thus, directly output $\mathcal{C}^w_{i-1}$, $\mathcal{C}^s_{i-1}$ as the results in G_i. E.g., in Fig. 2(a), after deleting $e1$, $\mathcal{C}^w_1 = \mathcal{C}^w_0 = \{C1, C2\}$.

Algorithm 3: MFCSED$(G_{i-1}, (u_i, v_i)^-, k, q, \mathcal{C}^w_{i-1}, \mathcal{C}^s_{i-1})$

```
1  $Size_{C^w_{i-1}} \leftarrow |C_w|(C_w \in \mathcal{C}^w_{i-1}),\ Size_{C^s_{i-1}} \leftarrow |C_s|(C_s \in \mathcal{C}^s_{i-1}),\ size \leftarrow 0$;
2  $InterW \leftarrow \mathcal{C}^w_{i-1}[u_i] \cap \mathcal{C}^w_{i-1}[v_i]$ , $InterS \leftarrow \mathcal{C}^s_{i-1}[u_i] \cap \mathcal{C}^s_{i-1}[v_i]$;
3  if $InterW = \emptyset$ and $InterS = \emptyset$ then return $\mathcal{C}^w_{i-1}, \mathcal{C}^s_{i-1}$;
4  else if $InterW \neq \emptyset$ then
5      if $|\mathcal{C}^w_{i-1}| \neq |InterW|$ then
6          $\mathcal{C}^w_i \leftarrow \mathcal{C}^w_{i-1} \setminus InterW$;  //By Early Termination Strategy 1 ;
7          Update $\mathcal{C}^s_i$ using lines 19-24 ;
8      else
9          if $Size_{C^w_{i-1}} = k \times |A|$ then
10             $\mathcal{C}^w_i \leftarrow \emptyset$ ;  //By Early Termination Strategy 2 ;
11             Update $\mathcal{C}^s_i$ using lines 19-24 ;
12         Compute $Ego_i(q)$ in $G_i$; $\widetilde{Ego}_i(q)$ =ColorfulCore($Ego_i(q), k$)[17]; ;
13         for each $w \in \widetilde{Ego}_i(q)$ and $w \neq q$ do
14             $P = \{q, w\}$; $I = \{x | x \in Nei(q, \widetilde{Ego}_i(q)) \cap Nei(w, \widetilde{Ego}_i(q)), ID(x) > ID(w)\}$;
15             $Size_{C^w_i} \leftarrow 2,\ Size_{C^s_i} \leftarrow 2,\ size \leftarrow 2$;
16             DeepSearch($size + 1, I', P'$) ;
17 else
18     if $|\mathcal{C}^s_{i-1}| \neq |InterS|$ then
19         $\mathcal{C}^s_i \leftarrow \mathcal{C}^s_{i-1} \setminus InterS$ ; return $\mathcal{C}^w_i, \mathcal{C}^s_i$; //By Early Termination Strategy 1 ;
20     else
21         if $Size_{C^s_{i-1}} = k \times |A|$ then
22             $\mathcal{C}^s_i \leftarrow \emptyset$ ; //By Early Termination Strategy 2 ;
23             return $\mathcal{C}^w_i, \mathcal{C}^s_i$;
24         Compute $Ego_i(q)$ in $G_i$; $\widetilde{Ego}_i(q)$ =ColorfulCore($Ego_i(q), k$)[17] ;
25         MSFCS($\widetilde{Ego}_i(q), k, q, \mathcal{C}^s_{i-1}$) ;
26 return $\mathcal{C}^w_i, \mathcal{C}^s_i$
```

(2) When $InterW$ is not empty, consider the following two cases:

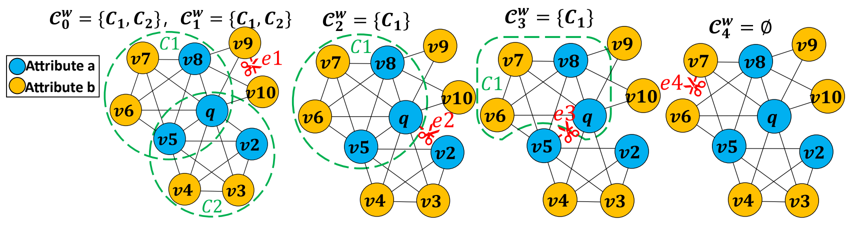
First, if $|\mathcal{C}^w_{i-1}| \neq |InterW|$ which means that part of G_{i-1}' maximum weak fair cliques (i.e., $\mathcal{C}^w_{i-1} \setminus InterW$) are not affected after deleting (u_i, v_i), then just output them as G_i's maximum weak fair cliques (i.e., $\mathcal{C}^w_i = \mathcal{C}^w_{i-1} \setminus InterW$). This is **Early Termination Strategy 1**. E.g., in Fig. 2(b), $\mathcal{C}^w_2 = \mathcal{C}^w_1 \setminus \{C2\} = \{C1\}$.

Second, while if $|\mathcal{C}^w_{i-1}| = |InterW|$, it means that all G_{i-1}' maximum weak fair cliques are affected after deleting (u_i, v_i). Their size are the same and denoted as $Size_{C^w_{i-1}}$. If $Size_{C^w_{i-1}} = k \times |A|$, then there exist no weak fair cliques satisfying the fairness threshold k in G_i so that $\mathcal{C}^w_i$ is empty. This is **Early Termination Strategy 2**. E.g., in Fig. 2(d), $Size_{C^w_3} = 2 \times 2$, so $\mathcal{C}^w_4 = \emptyset$.

After checking Early Termination Strategy 1 or 2 for G_i's maximum weak fair cliques, continue to find G_i's maximum strong fair cliques according to the step (3). Obviously, the above two early termination strategies also apply to G_i's maximum strong fair cliques.

Note that in the second case of the step (2), if $Size_{C^w_{i-1}} \neq k \times |A|$, then it has to invoke **DeepSearch** to re-compute $\mathcal{C}^w_i$ and $\mathcal{C}^s_i$.

(3) When $InterW$ is empty but $InterS$ is not empty, check Early Termination Strategy 1 or 2 for G_i's maximum strong fair cliques. If $|\mathcal{C}^s_{i-1}| = |InterS|$ and $Size_{C^s_{i-1}} \neq k \times |A|$ which means that all G_{i-1}' maximum strong fair cliques


$\mathcal{C}_0^w = \{C_1, C_2\}$, $\mathcal{C}_1^w = \{C_1, C_2\}$
$\mathcal{C}_2^w = \{C_1\}$
$\mathcal{C}_3^w = \{C_1\}$
$\mathcal{C}_4^w = \emptyset$
Attribute a
Attribute b
C1
C2
e1
e2
e3
e4
v2
v3
v4
v5
v6
v7
v8
v9
v10
q

(a) Deleting $e1$ (b) Deleting $e2$ (c) Deleting $e3$ (d) Deleting $e4$

Fig. 2: Illustration Examples for MFCSED.

```
Algorithm 4: MSFCS($\widetilde{Ego}_i(q), k, q, \mathcal{C}^s_{i-1}$)
1  $\mathcal{C}^s_i \leftarrow \emptyset,\ Size_{C^s_i} \leftarrow 0,\ size \leftarrow 0,\ Attr[r] \leftarrow 0 (r \in |A|)$;
2  $\overline{size} \leftarrow Size_{C^s_{i-1}} - |A|, \overline{NumA} \leftarrow \frac{Size_{C^s_{i-1}}}{|A|} - 1$ ;
3  for each $x \in V_{\widetilde{Ego}_i(q)}$ and $x \neq q$ do
4      $P \leftarrow \{q, x\},\ I \leftarrow \{y | y \in Nei(q, \widetilde{Ego}_i(q)) \cap Nei(x, \widetilde{Ego}_i(q))\ , ID(y) > ID(x)\}$;
5      $Attr[\phi(q)] \leftarrow Attr[\phi(q)] + 1, Attr[\phi(x)] \leftarrow Attr[\phi(x)] + 1$ ;
6      $Size_{C^s_i} \leftarrow 2,\ size \leftarrow 2$;
7      StrongDeepSearch($size + 1, I, P$);
8  return $\mathcal{C}^s_i$;

9  Procedure StrongDeepSearch($sizetemp, I, P$)
10     if $sizetemp > \overline{size}$ then return;  //By Pruning Strategy 3;
11     for each $y \in I$ do
12         $Attr[\phi(y)] \leftarrow Attr[\phi(y)] + 1$;
13         $P' = \{P \cup y\},\ I' = \{z | z \in Nei(y, \widetilde{Ego}_i(q)) \cap I, z > y\}$;
14         if $|I'| + |P'| < k * |A|$ then continue;
15         if $|I'| + |P'| < Size_{C^s_i}$ then continue;
16         if $\exists\ r \in A,\ Attr[r] > \overline{NumA}$ then continue; //By Pruning Strategy 4;
17         Update $\mathcal{C}^s_i$ and $Size_{C^s_i}$ using lines 18–20 of Algorithm 1.;
18         StrongDeepSearch($sizetemp + 1, I', P'$);
```

(whose size are $Size_{C^s_{i-1}}$) are affected after deleting (u_i, v_i), then we propose a new algorithm **MSFCS** with two pruning strategies to reduce the time of re-computing $\mathcal{C}^s_i$.

Similar to **DeepSearch**, **MSFCS** also treats q and each vertex in $V_{\widetilde{Ego}_i(q)}$ as an initial clique $P = \{q, x\}$, then performs DFS to search maximal and maximum strong fair cliques expended from P. The main differences between them are the following two new pruning strategies.

After deleting (u_i, v_i), let $\overline{size} = Size_{C^s_{i-1}} - |A|$ be the upper bound of the size which G_i's maximum strong fair cliques could have, and $\overline{NumA} = Size_{C^s_{i-1}}/|A| - 1$ be the upper bound of the number of each attribute value which G_i's maximum strong fair cliques' vertices could have. Moreover, the array $Attr[r]$ records the number of vertices with each attribute value r in the current candidate clique P'.

During the DFS, if the size *sizetemp* of the current candidate clique P' is larger than $\overline{size}$, stop expanding P' (**Pruning Strategy 3**). If there are vertices with any an attribute value r greater than $\overline{NumA}$ in P', stop expanding P' (**Pruning Strategy 4**).

Theorem 2. *Early Termination Strategies 1 and 2 are correct.*

Proof. Obviously, deleting (u_i, v_i) could not generate any cliques in G_i with the size larger than $Size_{C^w_{i-1}}$. For Early Termination Strategy 1, after deleting (u_i, v_i), in $\mathcal{C}^w_{i-1}$, the unaffected cliques' size is still $Size_{C^w_{i-1}}$, while the affected cliques' size is $Size_{C^w_{i-1}} - 1$. Thus, the set of the unaffected cliques (i.e., $\mathcal{C}^w_{i-1} \setminus$ InterW) is just the results in G_i. For Early Termination Strategy 2, $|InterW| = |\mathcal{C}^w_{i-1}|$ means that after deleting (u_i, v_i), all cliques in $\mathcal{C}^w_{i-1}$ are affected, so their size is all smaller than $Size_{C^w_{i-1}}$. When $Size_{C^w_{i-1}} = k \times |A|$, all cliques in $\mathcal{C}^w_{i-1}$ could no longer be weak fair in G_i. Thus, there exist no results in G_i. Similar proofs for strong fair cliques are omitted for space limitation. ■

Theorem 3. *Pruning Strategies 3 and 4 are correct.*

Proof. After deleting (u_i, v_i), $|\mathcal{C}^s_{i-1}| = |InterS|$ means that all maximum strong fair cliques are affected. Their size all reduces to $Size_{C^s_{i-1}} - |A| = \overline{size}$, and the number of vertices with each attribute value in a strong fair clique reduces to $Size_{C^s_{i-1}}/|A| - 1 = \overline{NumA}$. Deleting (u_i, v_i) could not generate any larger strong fair cliques, thus, if a candidate clique searched by **MSFCS** did not satisfy the upper bound $\overline{size}$ or $\overline{NumA}$, it should stop expanding. ■

Complexity Analysis. The worst time complexity of **MFCSED** is $O(Size_{C^w_{i-1}} \times |\mathcal{C}^w_{i-1}| + Size_{C^s_{i-1}} \times |\mathcal{C}^s_{i-1}| + |V'| + |E'| + |V'|^3 \times d)$. When Early Termination Strategies work, the time decreases to $O(Size_{C^w_{i-1}} \times |\mathcal{C}^w_{i-1}| + Size_{C^s_{i-1}} \times |\mathcal{C}^s_{i-1}|)$.

6 Experiments

All experiments are implemented with C++, on a Linux Server with AMD Ryzen Threadripper PRO 5995WX CPU(2.7 GHz) and 256 GB main memory.

Table 1: Summary of Algorithms.

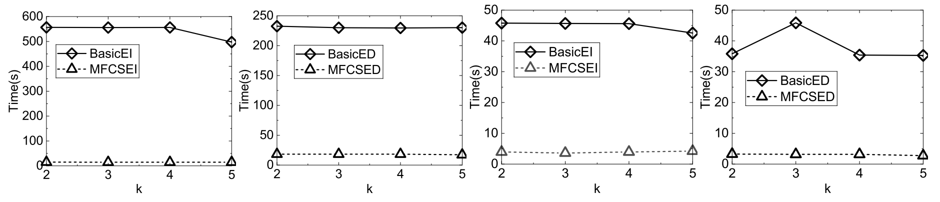
Algorithm	Description
BasicEI/ED	Basic algorithm
MFCSEI	MFCS-Edge Insertion
MFCSED	MFCS-Edge Deletion

Table 2: Statistics of Datasets.

Dataset	Vertices	Edges	Type
Wiki	2277	31421	Unattributed
Facebook	4039	88234	Unattributed
DBLP	317080	1049866	Unattributed
LiveJournal	3997962	34681189	Unattributed
GitHub	37700	289003	Attributed
Twitch	168114	6797557	Attributed

Table 3: Parameter Configuration for Datasets.

Parameter	Networks	Range	Default Value
k	All 6 Datasets	{2,3,4,5}	3
$\|A\|$	All 6 Datasets	{2,3,4,5}	3
ρ_q	Wiki	{[0.2, 0.25), [0.25, 0.3), [0.3, 0.35), [0.35, 0.4)}	[0.25, 0.3)
	DBLP	{[0.4, 0.45), [0.45, 0.5), [0.5, 0.55), [0.55, 0.6)}	[0.45, 0.5)
	Other Datasets	{[0.05, 0.1), [0.1, 0.15), [0.15, 0.2), [0.2, 0.25)}	[0.1, 0.15)



(a) Wiki, Edge Inst. (b) Wiki, Edge Del. (c) Facebook, Edge Inst. (d) Facebook, Edge Del.

Fig. 3: Varying k.

Algorithms. For performance testing, we perform our BasicEI/ED, MFCSEI and MFCSED algorithms listed in Table 1.

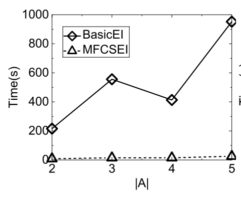
Datasets. We use 6 real-world graphs shown in Table 2. Wiki[3] describes hyperlink relationships between Wikipedia articles. Facebook[4] reflects friendship links among users in social network platforms. DBLP[5] represents co-authorship relations between authors. LiveJournal[6] describes friendship among users in a free on-line blogging community. The above 4 graphs are non-attributed, for performance testing, we randomly generate an attribute value for each vertex on each graph. GitHub[7] reflects the mutual follower relationships among GitHub developers, with the developer type serving as node attributes. Twitch[8] describes the mutual follower relationships among Twitch gamers, with the affiliate status serving as node attributes.

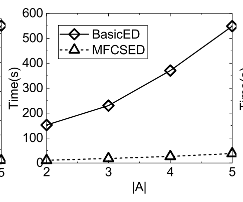
Parameters and Metrics. We vary 3 parameters k, $|A|$, $\rho_q(=\frac{Deg(q)}{deg_{max}}$, where deg_{max} is the maximum vertex degree in a graph) to compare the performance of our Basic, MFCSEI and MFCSED algorithms. The ranges and default values of the parameters are shown in Table 3. Without otherwise stated, when varying a parameter, other parameters are set to their default value. For all experiments, we conduct several queries and report the average CPU running time.

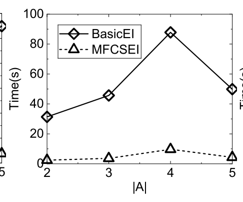
Exp-1 Impact of k. To examine the impact of k, we test the running time of three algorithms by varying k in Wiki and Facebook. As shown in Fig. 3, when k increases, both MFCSEI and MFCSED always cost quite little time, and are significantly superior to Basic. As k grows, the time of Basic slightly decreases, because the pruned search space becomes smaller when k is larger.

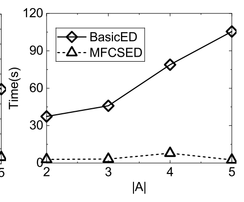
Exp-2 Impact of $|A|$. To examine the impact of $|A|$, we test the running time of three algorithms by varying $|A|$ in Wiki and Facebook. In Fig. 4, as $|A|$ increases, both MFCSEI and MFCSED run much faster than Basic in all cases.

[3] Wiki: `https://snap.stanford.edu/data/wikipedia-article-networks.html`
[4] Facebook: `https://snap.stanford.edu/data/egonets-Facebook.html`
[5] DBLP: `https://dblp.uni-trier.de/xml/`
[6] LiveJournal:`https://snap.stanford.edu/data/com-LiveJournal.html`
[7] GitHub:`https://snap.stanford.edu/data/github-social.html`
[8] Twitch:`https://snap.stanford.edu/data/twitch_gamers.html`



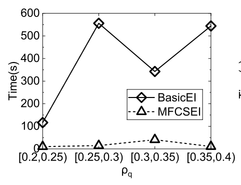


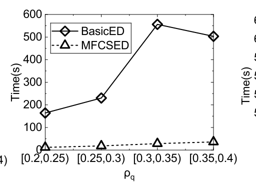


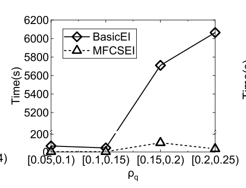


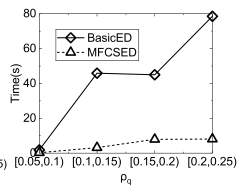
(a) Wiki, Edge Inst. (b) Wiki, Edge Del. (c) Facebook, Edge Inst. (d) Facebook, Edge Del.

Fig. 4: Varying $|A|$.







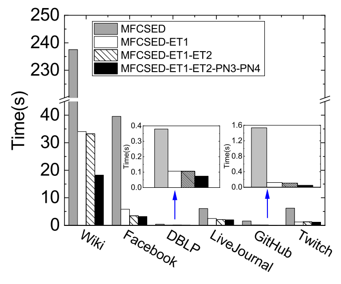


(a) Wiki, Edge Inst. (b) Wiki, Edge Del. (c) Facebook, Edge Inst. (d) Facebook, Edge Del.

Fig. 5: Varying ρ_q.

Dataset	$avg(\lvert V_{\widetilde{Ego_i}(q)}\rvert)$	$avg(\lvert V_{cand}\rvert)$	$\frac{avg(\lvert V_{\widetilde{Ego_i}(q)}\rvert)}{avg(\lvert V_{cand}\rvert)}$
Wiki	105	55	1.91
Facebook	106	51	2.08
DBLP	54	7	7.71
LiveJournal	1220	348	3.51
GitHub	56	6	9.33
Twitch	2016	5	403.20

(a) MFCSEI



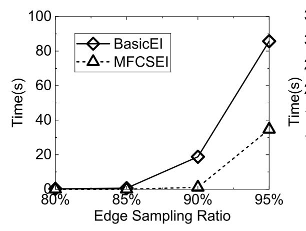
(b) MFCSED

Fig. 6: Optimization Strategy Evaluation for MFCSEI and MFCSED

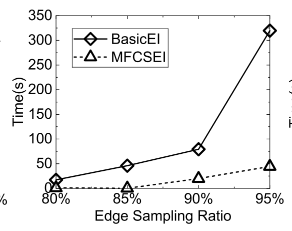
When A grows, the time of Basic is also increasing. Because when $|A|$ is larger,the fair cliques would have larger size so that more search time is needed.

Exp-3 Impact of Query Vertices. To examine the impact of different query vertices, we test the running time of three algorithms by varying ρ_q in Wiki and Facebook. As shown in Fig. 5, when q has smaller degree, MFCSEI and MFCSED are similar to or slightly better than Basic. When q has larger degree, MFCSEI and MFCSED are much better than Basic. In addition, Basic takes more time when q has larger degree, because in such cases the search space becomes larger.

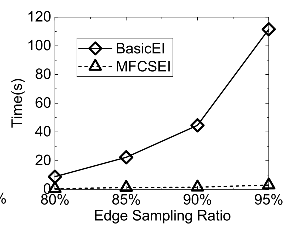
Exp-4 Optimization Strategy Evaluation. Firstly, to evaluate the pruned effect of the search space for MFCSEI, we compare the average size of $V_{\widetilde{Ego_i}(q)}$ and V_{cand} on six datasets. As shown in Fig. 6(a), in all six datasets, V_{cand} contains



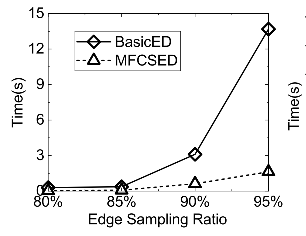
(a) DBLP



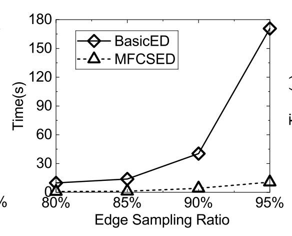
(b) Twitch



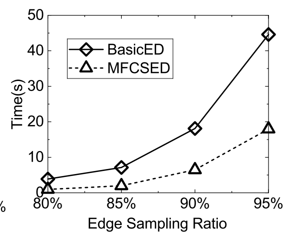
(c) LiveJournal



(d) DBLP

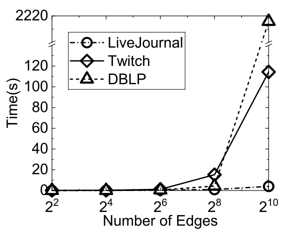


(e) Twitch

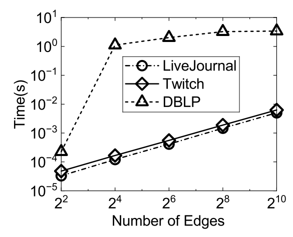


(f) LiveJournal

Fig. 7: Scalability Test



(a) MFCSEI, Edge Insertion



(b) MFCSED, Edge Deletion

Fig. 8: Time of MFCSEI and MFCSED, Multiple Edges Insertion/Deletion

much fewer vertices than $V_{\widetilde{Ego_i(q)}}$, which indicates that the search space can be effectively pruned when conducting MFCSEI compared with Basic. Secondly, to evaluate the effect of early termination and pruning strategies for MFCSEI, we test the running time of MFCSED with different strategies. Fig. 6(b) shows that all the strategies work well in different datasets.

Exp-5 Scalability Test. We test the scalability of our algorithms (i.e.,Basic, MFCSEI and MFCSED) using their default values by testing their running time when varying the graph size (randomly selecting 80%, 85%, 90%, 95% edges from DBLP, Twitch and LiveJournal respectively). Fig.7 shows that MFCSEI and MFCSED scale well as the number of edges increases.

Exp-6 Evaluating Algorithms for Deleting/Inserting Multiple Edges. We test the total running time of our incremental algorithms MFCSEI and

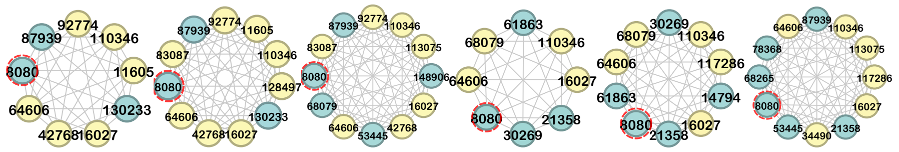

(a) (b) (c) (d) (e) (f)

Fig. 9: Case Study, Twitch, q="8080", k=3

MFCSED while inserting or deleting multiple edges in batch in three datasets namely DBLP, Twitch and LiveJournal. As shown in Fig.8, when the number of edges increases, both MFCSEI and MFCSED cost more time in those three datasets. Note that in the case of edge insertion, the curve for LiveJournal grows more slowly than the other two datasets, because the LiveJournal graph is relatively sparser. As a result, even lots of edges are inserted, the size of V_{cand} is still quite small. In the case of edge deletion, the growth trend of the curve for DBLP slows down as the number of edges increases, because the size of cliques in DBLP is relatively smaller. Thus, when more edges are deleted, the DBLP graph would contain fewer cliques, which leads to less time for searching for fair cliques containing the query vertex.

Exp-7 Case Study. In Twitch, set q="8080", k=3, Fig. 9(b) shows the original MWFC. Fig. 9(a) and Fig. 9(c) show the MWFCs found by our MFCSED and MFCSEI algorithms after deleting and inserting 50 edges, respectively. Similarly, Fig. 9(e) shows the original MSFC, and Fig. 9(d) and Fig. 9(f) show the MSFCs found by MFCSED and MFCSEI after deleting and inserting 50 edges.

7 Conclusion

In this paper, we are the first to concern how to efficiently search maximum fair cliques which contain the given query vertex in a dynamic attributed graph. We formalize the MFCS-OneEdgeInsertion and MFCS-OneEdgeDeletion problems which are NP-hard. Besides the basic algorithm which recalculates the results, we also propose two incremental algorithms to avoid recalculating all results from scratch. Moreover, we develop several optimization strategies to greatly reduce the time of searching maximum fair cliques. Our experimental results on 6 real-life networks verify amazingly higher efficiency and better scalability of our incremental algorithms compared to the basic algorithm. In the future, we will try to further speed up the incremental algorithms for the case of inserting or deleting multiple edges or vertices each time, and study how to perform our algorithms on distributed computing platform.

Acknowledgments. This work was supported by NSFC Grants 62202277, U23A20302, U22A2025. Jingwen Chen and Jianuo Xu contributed equally to this work and are co-first authors. Xinrui Wang is the corresponding author.

References

1. Akbas, E., Zhao, P.: Truss-based community search: a truss-equivalence based indexing approach. Proc. VLDB Endow. **10**(11), 1298–1309 (2017)
2. Batagelj, V., Zaversnik, M.: An o(m) algorithm for cores decomposition of networks. CoRR **cs.DS/0310049** (2003)
3. Boginski, V., Butenko, S., Pardalos, P.M.: Mining market data: A network approach. Comput. Oper. Res. **33**(11), 3171–3184 (2006)
4. Bose, A.J., Hamilton, W.L.: Compositional fairness constraints for graph embeddings. In: ICML. vol. 97, pp. 715–724. PMLR (2019)
5. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph (algorithm 457). Commun. ACM **16**(9), 575–576 (1973)
6. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. National security agency technical report **16**(3.1), 1–29 (2008)
7. Das, A., Svendsen, M., Tirthapura, S.: Incremental maintenance of maximal cliques in a dynamic graph. VLDB J. **28**(3), 351–375 (2019)
8. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in sparse graphs in near-optimal time. In: ISAAC. vol. 6506, pp. 403–414. Springer (2010)
9. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. ACM J. Exp. Algorithmics **18** (2013)
10. He, Y., Lin, L., Yuan, P., Li, R., Jia, T., Wang, Z.: CCSS: towards conductance-based community search with size constraints. Expert Syst. Appl. **250**, 123915 (2024)
11. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: SIGMOD. pp. 1311–1322. ACM (2014)
12. Hussein, F., El-Salhi, S., Alazazma, R., Abu-Hantash, T., Abu-Hantash, H., Thaher, H.: An android application using machine learning algorithm for clique detection in issues related to transportation. Int. J. Interact. Mob. Technol. **16**(14), 4–22 (2022)
13. Jensen, T.R., Toft, B.: Graph coloring problems. John Wiley & Sons (2011)
14. Kleindessner, M., Samadi, S., Awasthi, P., Morgenstern, J.: Guarantees for spectral clustering with fairness constraints. In: ICML. vol. 97, pp. 3458–3467. PMLR (2019)
15. Li, Y., Chen, H., Xu, S., Ge, Y., Tan, J., Liu, S., Zhang, Y.: Fairness in recommendation: Foundations, methods, and applications. ACM Trans. Intell. Syst. Technol. **14**(5), 95:1–95:48 (2023)
16. Liang, Y.: Outlier item detection in bundle recommendation via the attention mechanism. High-Confidence Computing **4**(3), 100200 (2024)
17. Pan, M., Li, R., Zhang, Q., Dai, Y., Tian, Q., Wang, G.: Fairness-aware maximal clique enumeration. In: ICDE. pp. 259–271. IEEE (2022)
18. Rabaa, A.E., Elbassuoni, S., Hanna, J., Mouawad, A.E., Olleik, A., Amer-Yahia, S.: A framework to maximize group fairness for workers on online labor platforms. Data Sci. Eng. **8**(2), 146–176 (2023)
19. Rahman, S.I., Ahmed, S., Fariha, T.A., Mohammad, A., Haque, M.N.M., Chellappan, S., Noor, J.: Unsupervised machine learning approach for tailoring educational content to individual student weaknesses. High-Confidence Computing **4**(4), 100228 (2024)
20. Rossi, R.A., Gleich, D.F., Gebremedhin, A.H.: Parallel maximum clique algorithms with applications to network analysis. SIAM J. Sci. Comput. **37**(5) (2015)

21. Sun, S., Li, W., Wang, Y., Liao, W., Yu, P.S.: Continuous monitoring of maximum clique over dynamic graphs. IEEE Trans. Knowl. Data Eng. **34**(4), 1667–1683 (2022)
22. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theor. Comput. Sci. **363**(1), 28–42 (2006)
23. Tsioutsiouliklis, S., Pitoura, E., Tsaparas, P., Kleftakis, I., Mamoulis, N.: Fairness-aware pagerank. In: WWW. pp. 3815–3826. ACM / IW3C2 (2021)
24. Yu, T., Jiang, T., Bah, M.J., Zhao, C., Huang, H., Liu, M., Zhou, S., Li, Z., Zhang, J.: Incremental maximal clique enumeration for hybrid edge changes in large dynamic graphs. IEEE Trans. Knowl. Data Eng. **36**(4), 1650–1666 (2024)
25. Zheng, X., Liu, T., Yang, Z., Wang, J.: Large cliques in arabidopsis gene coexpression network and motif discovery. Journal of Plant Physiology **168**(6), 611–618 (2011)
26. Zhou, Y., Fang, Y., Luo, W., Ye, Y.: Influential community search over large heterogeneous information networks. Proc. VLDB Endow. **16**(8), 2047–2060 (2023)
27. Zhou, Y., Guo, Q., Fang, Y., Ma, C.: A counting-based approach for efficient k-clique densest subgraph discovery. Proc. ACM Manag. Data **2**(3), 119 (2024)