Fairness-Aware Maximal Clique in Large Graphs: Concepts and Algorithms

Qi Zhang[®], Rong-Hua Li[®], Minjia Pan, Yongheng Dai, Qun Tian, and Guoren Wang[®]

Abstract—Cohesive subgraph mining on attributed graphs is a fundamental problem in graph data analysis. Existing cohesive subgraph mining algorithms on attributed graphs do not consider the *fairness* of attributes in the subgraph. In this article, we, for the first time, introduce fairness into the widely-used clique model to mine fairness-aware cohesive subgraphs. In particular, we propose three novel fairness-aware maximal clique models on attributed graphs, called weak fair clique, strong fair clique and relative fair clique, respectively. To enumerate all weak fair cliques, we develop an efficient backtracking algorithm called WFCEnum equipped with a novel colorful *k*-core based pruning technique. We also propose an efficient enumeration algorithm called SFCEnum to find all strong fair cliques based on a new attribute-alternatively-selection search technique. To further improve the efficiency, we also present several non-trivial ordering techniques for both weak and strong fair clique enumerations. To enumerate all relative fair cliques, we design an enhanced colorful *k*-core based pruning technique for 2D attributes, and develop two efficient search algorithms: RFCRefineEnum and RFCAlterEnum for arbitrary dimension attributes. The results of extensive experiments on four real-world graphs demonstrate the efficiency, scalability and effectiveness of the proposed algorithms.

Index Terms-Attributed graph, fairness, maximal clique enumeration

1 INTRODUCTION

Complex networks in the real world, such as social networks, can be modeled as graphs. Graph analysis techniques have been extensively studied to help understand the features of networks. Community detection, which aims at finding cohesive subgraph structures in networks, is a fundamental problem in graph analysis that has attracted much attention for decades [1], [2], [3]. As an elementary model, clique has been widely used to reveal dense community structures of graphs [4], [5]. Mining cliques in a graph has a wide range of applications, including mining overlapping communities in social networks [6], identifying protein complexes in protein networks [7], and finding groups with abnormal transactions in financial networks [8].

Many real-life networks are often attributed graphs where vertices or edges are associated with attribute information. There are a large number of studies that focus on finding communities on attributed graphs [9], [10], [11], [12], [13], [14], [15], [16]. However, those works either require a high correlation of attributes in a community or aim to find communities

(Corresponding authors: Guoren Wang and Rong-Hua Li.) Recommended for acceptance by T. Weninger. Digital Object Identifier no. 10.1109/TKDE.2022.3232165 satisfying some attribute constraints. None of them takes into account the *fairness* of attributes in the community.

Recently, the concept of fairness is mainly considered in the machine learning community [17], [18], [19]. Many studies reveal that a rank produced by a biased machine learning model can result in systematic discrimination and reduce visibility for an already disadvantaged group (e.g., incorporations of gender and racial and other biases) [20], [21], [22]. Therefore, many different definitions of fairness, such as individual fairness, group fairness [17], and related algorithms were proposed to generate a fairness ranking. Some other studies focus on the fairness in classification models, such as demographic parity [19] and equality of opportunity [18]. All these studies suggest that the concept of fairness is very important in machine learning models.

Motivated by the concept of fairness in machine learning, we introduce fairness for an important graph mining task, i.e., mining cliques in a graph. Mining fair cliques has a variety of applications. For example, consider an online social network where each user has an attribute denoting his/her gender. We may want to find a clique community in which both the number of males and females reach a certain threshold, or the number of males is exactly the same or slightly different from the number of females. Compared to the traditional clique communities, the fair clique communities can overcome gender bias. In a collaboration network, each vertex has an attribute representing his/her research topic. The fair cliques can be used to identify research groups who work closely and also have diverse research topics, because the fair cliques have already considered the fairness over different research topics. Finding such fair cliques can help identify the groups of experts from diverse research areas to conduct a particular task.

In this article, we focus on the problem of finding fairnessaware cliques in attributed graphs where each vertex in the graph has one attribute. We propose three new models to characterize the fairness of a clique, called weak fair clique,

Qi Zhang, Rong-Hua Li, Minjia Pan, and Guoren Wang are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: qizhangcs@bit.edu.cn, {lironghuabit, wanggrbit}@126.com, panminjia_cs@163.com.

Yongheng Dai and Qun Tian are with the Diankeyun Technologies Ltd, Beijing 100043, China. E-mail: toyhdai@163.com, tianqun@cetccloud.com.

Manuscript received 14 May 2022; revised 4 November 2022; accepted 17 December 2022. Date of publication 10 January 2023; date of current version 6 October 2023.

This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3301301, in part by NSFC under Grants U2241211, 62072034, and U1809206, and in part by CCF-Huawei Populus Grove Fund.

^{1041-4347 © 2023} IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

strong fair clique and relative fair clique, respectively. A weak fair clique is a maximal subgraph which 1) is a clique, and 2) requires the number of vertices of every attribute value is no less than a given threshold k, thus it can guarantee the fairness over all attributes to some extent. A strong fair clique is a maximal subgraph in which 1) the vertices form a clique, and 2) the number of vertices for each attribute value is no less than kand exactly the same, thus it can fully guarantee the fairness over all attributes. A relative fair clique is a maximal subgraph in which 1) the vertices form a clique, 2) the number of vertices for each attribute value is no less than k, and 3) the difference in the number of vertices for all attributes is no larger than a given threshold δ . Thus, the relative fair clique is a compromise model between the weak and strong fair cliques, which not only guarantees the coverage of each attribute, but also implements a more flexible balance between all attributes. We show that finding all weak, strong and relative fair cliques is NP-hard. Furthermore, the problem of enumerating all strong and relative fair cliques is often much more challenging than the problem of enumerating all weak fair cliques. To solve our problems, we first propose a backtracking enumeration algorithm called WFCEnum with a novel colorful k-core based pruning technique to find all weak fair cliques. Then, we propose a SFCEnum algorithm to enumerate all strong fair cliques based on a new attribute-alternatively-selection search strategy. We also develop several non-trivial ordering techniques to further speed up the WFCEnum and SFCEnum algorithms. Additionally, to enumerate all relative fair cliques, we design an enhanced colorful k-core based pruning technique for 2D attribute, and present two efficient search algorithms, i.e., RFCRefineEnum and RFCAlterEnum, to handle any dimension attribute. Below, we summarize the main contributions of this paper.

New Models. We propose a weak fair clique, a strong fair clique and a relative fair clique to characterize the fairness of a cohesive subgraph. To the best of our knowledge, we are the first to introduce the concept of fairness for cohesive subgraph models.

Novel Algorithms. We first propose a novel concept called colorful k-core and develop a linear-time algorithm to compute the colorful *k*-core. We show that the weak fair cliques, strong fair cliques and relative fair cliques must be contained in the colorful k-core, thus we can use it to prune unpromising vertices before enumerating weak, strong or relative fair cliques. Then, we propose a backtracking algorithm WFCEnum to find all weak fair cliques with a colorful k-core induced ordering. To enumerate all strong fair cliques, we further develop a novel fairness k-core based pruning technique which is more effective than the colorful k-core pruning. We also present a backtracking algorithm SFCEnum with a new attribute-alternatively-selection search strategy to enumerate all strong fair cliques. In addition, a heuristic ordering method is also proposed to further improve the efficiency of the strong fair clique enumeration algorithm. For the problem of relative fair clique enumeration, we develop two efficient algorithms, i.e., RFCRefineEnum based on a weak fair clique refinement technique and RFCAlterEnum equipped with an attribute-alternatively-selection strategy. We also design an enhanced colorful k-core based pruning technique for 2D attributes which can also be used to find all weak fair cliques.

Extensive Experiments. We conduct extensive experiments to evaluate the efficiency and effectiveness of our algorithms using four real-world networks. The results indicate that the colorful k-core based pruning technique is very powerful which can significantly prune the original graph. The also show that the WFCEnum, SFCEnum, results RFCRefineEnum and RFCAlterEnum algorithms are efficient in practice. These algorithms can enumerate all fair cliques on a large graph with 2,523,387 vertices and 7,918,801 edges in less than 3 hours. In addition, we conduct a case study on DBLP to evaluate the effectiveness of our algorithms. The results illustrate that the proposed fair clique enumeration algorithms, i.e., WFCEnum, SFCEnum, RFCRefineEnum and RFCAlterEnum, can find fair communities with different research areas. Moreover, SFCEnum can further keep balance of attribute values in the subgraph, and RFCRefineEnum and RFCAlterEnum can explore the communities which not only cover each attribute, but also appropriately avoid the imbalance of attributes.

Reproducibility. The source code of this paper is released at Github: https://github.com/honmameiko22/fairnessclique for reproducibility purpose.

2 PRELIMINARIES

Let G = (V, E, A) be an undirected, unweighted attributed graph with n = |V| and m = |E|. Each vertex u in G has an attribute A and we denote its value as u.val. Let A_{val} be the set of all possible values of attribute A, namely, $A_{val} =$ $\{u.val|u \in V\}$. The cardinality of A_{val} is denoted by A_n , i.e., $A_n = |A_{val}|$. For brevity, we also represent A_{val} as $A_{val} =$ $\{a_i|0 \le i < A_n\}$. We denote the set of neighbors of a vertex u by N(u), and the degree of u by d(u) = |N(u)|. For a vertex subset $S \subseteq V$, the subgraph induced by S is defined as $G_S =$ (S, E_S, A) , where $E_S = \{(u, v)|(u, v) \in E, u, v \in S\}$ and A is the vertex attribute in G.

In a graph G, a clique C is a complete subgraph where each pair of vertices in C is connected. Perhaps, clique is the most fundamental model to characterize cohesive subgraphs in a graph. Based on the clique model, we present three novel fairness-aware clique models which are able to capture the fairness property of cohesive subgraphs.

Definition 1. (Weak fair clique) Given an attributed graph Gand an integer k, a clique C of G is a weak fair clique of G if (1) for each value $a_i \in A_{val}$, the number of vertices whose value equals a_i is no less than k; (2) there is no clique $C' \supset C$ satisfying (1).

Clearly, by Definition 1, the weak fair clique model exhibits the *fairness property* over all types of vertices (with different attribute values), as it requires the number of vertices for each attribute in the subgraph must be no less than *k*. However, the weak fair clique model may not strictly guarantee fairness for all attributes because there may be an excessive number of nodes with some attributes. Below, we propose a strong fair clique model which strictly requires the subgraph has the same number of vertices for each attribute.

Definition 2. (Strong fair clique) Given an attributed graph Gand an integer k, a clique C of G is a strong fair clique of G if (1) for each $a_i \in A_{val}$, the number of vertices whose value equals

 a_i is no less than k; (2) the number of vertices for each a_i is exactly the same; (3) there is no clique $C' \supset C$ satisfying (1) and (2).

With Definition 2, the strong fair clique model requires the subgraph has the strictly same number of vertices for each attribute. Thus, it can overcome the imbalance between attributes in a clique caused by the excessive number of vertices for some attributes in the weak fair clique. However, the strong fair clique model guaranteeing fairness for all attributes is too strict to work in some real-life applications flexibly. For example, in an online social network with gender as the attribute, we only want to find a clique community in which the number of males and females is roughly equal rather than strictly equal. To this end, we propose a relative fair clique to achieve a good compromise, which absorbs the advantages of the weak and strong fair clique models.

- **Definition 3.** (Relative fair clique) Given an attributed graph G and two integers k, δ, a clique C of G is a relative fair clique of G if (1) for each value $a_i \in A_{val}$, the number of vertices whose value equals a_i is no less than k; (2) for arbitrary two attribute a_i and a_j , the difference of the number of vertices with a_i and a_j in C is no larger than δ , i.e., $|cnt_C(a_i) cnt_C(a_j)| \le \delta$; (3) there is no clique $C' \supset C$ satisfying (1) and (2).
- **Example 1.** Consider the attributed graph *G* in Fig. 2. We suppose that k = 3 and $\delta = 1$. By Definition 3, we can easily derive that the clique C_1 induced by $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ is a relative fair clique that involves 3 vertices with *a* and 4 vertices with *b*. While the clique C_2 induced by $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ is not a relative fair clique since it contains 5 vertices with *a* and 3 vertices with *b*, which violates the condition (2) of Definition 3. The clique C_3 induced by $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ is also not because C_1 is a larger clique that contains C_3 , which violates the condition (3) of Definition 3. Clearly, C_2 is a weak fair clique and C_3 is a strong fair clique, we have $C_2 \supset C_1 \supset C_3$. Thus, C_2 is indeed a compromise clique between C_1 and C_3 .

Remark. According to Definitions 1, 2, and 3, the parameter k in our fair clique models provides a lower bound on the size of a clique. There are at least $k \times A_n$ vertices in a weak/strong/relative fair clique. Note that the guarantee of fairness in our models lies in that no matter how large a clique is, every attribute owns at least *k* vertices. The weak fair clique model is suitable to the applications which require a lower-bound guarantee of fairness. The strong fair clique, however, aims at finding absolutely fair cliques, which can be applied in the scenarios like finding a group of people where the number of females equals that of males. In comparison, the relative fair clique achieves a compromise between the weak fair clique and strong fair clique models. Specifically, when $\delta = \infty$, a relative fair clique degenerates to a weak fair clique, and it evolves into a strong fair clique in the case of $\delta = 0$. Hence, a relative fair clique must be contained in weak fair cliques, and a strong fair clique must be contained in relative fair cliques.

Note that in the relative fair clique model, we also require the number of vertices for each attribute in the clique must be no less than k. This is because if we only guarantee that the difference of the number of each attribute is Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Dow

below a given threshold δ , we may miss fairness in some cases. For example, suppose that we have three attributes: A, B and C, and the given difference threshold is $\delta = 5$. Then, we may find a 5-clique that has 5 vertices with A, 0 vertex with B, and 0 vertex with C which is clearly unfair for the attributes B and C. Hence, all our definitions of fairness-aware cliques need to guarantee that each attribute has at least *k* vertices.

Problem Statement. Given an attributed graph *G* and two integers *k* and δ , our goal is to enumerate all weak fair cliques and strong fair cliques with *k*, and enumerate all relative fair cliques in *G* with *k* and δ .

Challenges. We first discuss the hardness of the weak fair clique enumeration problem. Considering a special case: k = 0. Clearly, the weak fair clique enumeration problem degenerates to the traditional maximal clique enumeration problem which is NP-hard. Thus, finding all weak fair cliques is also NP-hard. Enumerating strong fair cliques is more challenging than enumerating all weak fair cliques for the following reasons. (1) The number of strong fair cliques is often much larger than that of weak fair cliques. By definition, we can see that a strong fair clique is always contained in a weak fair clique. On the contrary, a weak fair clique is not necessarily a strong fair clique. (2) Each weak fair clique must be a traditional maximal clique, but the strong fair clique may not be a traditional maximal clique, which means that it is difficult to check the maximality of strong fair cliques. For relative fair clique enumeration problem, when $\delta = \infty$, it degenerates to the weak clique enumeration problem which is NP-hard. Moreover, like the strong fair clique model, the number of relative fair cliques is also much larger than that of weak fair cliques and it is also difficult to check the maximality.

Unlike traditional maximal cliques, our fair clique models have an additional attribute value constraint, thus a potential solution is to apply attribute information to prune the search space. The challenges of our problems are (1) how can we efficiently prune unpromising vertices, and (2) how to maintain the fair clique property during the search procedure. To tackle the above challenges, we will propose the WFCEnum algorithm with a new colorful k-core based pruning technique for weak fair clique enumeration; propose the SFCEnum algorithm with a novel attribute-alternatively-selection strategy for enumerating all strong fair cliques; and propose a RFCRefineEnum algorithm based on a weak fair clique refinement technique and a RFCAlterEnum algorithm with an attribute-alternatively-selection strategy to enumerate all relative fair cliques. All the proposed algorithms are able to correctly find all fair cliques and significantly improve the efficiency compared to the baseline enumeration algorithm.

3 WEAK FAIR CLIQUE ENUMERATION

In this section, we present the WFCEnum algorithm to enumerate all weak fair cliques. The key idea of WFCEnum is that it first prunes the vertices that are not contained in any weak fair clique based on a novel concept called colorful *k*-core. Then, it performs a carefully-designed backtracking search procedure to enumerate all results. Below, we first introduce the concept of colorful *k*-core, followed by a heuristic search order and the WFCEnum algorithm.



Fig. 1. Running example.

3.1 The Colorful *K*-Core Pruning

Before introducing the colorful *k*-core based pruning technique, we first briefly review the problem of vertex coloring for a graph. The goal of vertex coloring is to color the vertices such that no two adjacent vertices have the same color [23], [24] (See Fig. 1). Given a graph G = (V, E), we denote by color(u) the color of a vertex $u \in V$. Based on the vertex coloring, we define the *colorful degree* of a vertex as follows.

Definition 4. (Colorful degree) Given an attributed graph G = (V, E, A) and an attribute value $a_i \in A_{val}$. The colorful-degree of vertex u based on a_i , denoted by $D_{a_i}(u, G)$, is the number of colors of u's neighbors whose attribute value is a_i , i.e., $D_{a_i}(u, G) = |\{color(v)|v \in N(u), v.val = a_i\}|.$

Clearly, each vertex u has A_n colorful degrees. Let $D_{\min}(u, G)$ denotes the minimum colorful degree of a vertex u, i.e., $D_{\min}(u, G) = \min\{D_{a_i}(u, G)|a_i \in A_{val}\}$. We omit the symbol G in $D_{a_i}(u, G)$ and $D_{\min}(u, G)$ when the context is clear. Below, we give the definition of *colorful k-core*.

Definition 5. (Colorful k-core) Given an attributed graph G = (V, E, A) and an integer k, a subgraph $H = (V_H, E_H, A)$ of G is a colorful k-core if: (1) for each vertex $u \in V_H$, $D_{\min}(u, H) \ge k$; (2) there is no subgraph $H' \subseteq G$ that satisfies (1) and $H \subset H'$.

Based on Definition 5, we have the following lemma.

Lemma 1. Given an attributed graph G = (V, E, A) and a parameter k, any weak fair clique must be contained in the colorful (k-1)-core of G.

Equipped with Lemma 1, we propose a novel algorithm, called ColorfulCore, to compute the colorful-k-core of G, which can be used to prune unpromising vertices in the weak fair clique enumeration procedure. The pseudo-code of ColorfulCore is shown in Algorithm 1. The algorithm computes the colorful-k-core of G by iteratively peeling vertices from the remaining graph based on their colorful degrees, which is a variant of the classic core decomposition algorithm [25], [26] (lines 8-20). Specifically, it first performs greedy coloring on G which

V₂ V₃ a v₁ a V₃ b b b a V₄ V₁₂ V₁₃ V₅ a V₆ b c v₇ V₆ V₉ v₉ V₁₀ a V₁₀

colors vertices based on the order of degree [27], [28] (line 1). Note that finding the optimal coloring is an NPhard problem [23], [24], thus we use a greedy algorithm to compute a heuristic coloring which is sufficient for defining the colorful k-core. A priority queue Q is employed to maintain the vertices with smaller D_{\min} which will be removed during the peeling procedure (line 2). ColorfulCore computes the colorful degrees of all vertices to initialize Q (lines 3-10). M_u records the number of u's neighbors whose attribute values and colors are the same. After that, the algorithm computes the colorful k-core of G by iteratively peeling vertices from the remaining graph based on their colorful degrees (lines 11-20). Finally, ColorfulCore returns the remaining graph \hat{G} as the colorful k-core. Below, we analyze the complexity of Algorithm 1.

Algorithm 1. ColorfulCore

Input: G = (V, E, A), an integer k**Output:** The colorful k-core \hat{G}

- Color all vertices by invoking a degree-based greedy coloring algorithm;
- 2: Let Q be a priority queue; $Q \leftarrow \emptyset$;
- 3: for $u \in V$ do
- 4: for $v \in N(u)$ do
- 5: **if** $M_u(v.val, color(v)) = 0$ **then** $D_{v.val}(u)$ ++;
- 6: $M_u(v.val, color(v)) + +;$
- 7: $D_{\min}(u) \leftarrow \min\{D_{a_i}(u)|a_i \in A_{val}\};$
- 8: for $u \in V$ do
- 9: **if** $D_{\min}(u) < k$ then
- 10: $\mathcal{Q}.push(u)$; Remove u from G;
- 11: while $Q \neq \emptyset$ do
- 12: $u \leftarrow Q.pop();$
- 13: for $v \in N(u)$ do
- 14: **if** *v is not removed* **then**
- 15: $M_v(u.val, color(u)) -;$

16: **if**
$$M_v(u.val, color(u)) \le 0$$
 then

17:
$$D_{u.val}(v) \leftarrow D_{u.val}(v) - 1;$$

- 18: $D_{\min}(v) \leftarrow \min\{D_{a_i}(v)|a_i \in A_{val}\};$
- 19: **if** $D_{\min}(v) < k$ then
- 20: Q.push(v); Remove v from G;
- 21: The colorful *k*-core $\hat{G} \leftarrow$ the remaining graph of *G*;

22: return *G*;

Theorem 1. Algorithm 1 consumes O(E + V) time using $O(V \times A_n \times \text{color})$ space, where color denotes the total number of colors.

3.2 The Colorful *K*-Core Based Ordering

WFCEnum finds all weak fair cliques by performing a backtracking search procedure. Hence, the search order of vertices is vital as the search spaces with various orderings are significantly different. Below, we propose a heuristic order based on the colorful *k*-core, called ColorOD, which can significantly improve the performance of WFCEnum as confirmed in our experiments.

Consider a vertex u and its neighbor v with $D_{\min}(u, G) \ge (k-1) > D_{\min}(v, G)$. According to Lemma 1, u may be contained in a weak fair clique but v is impossible. Thus, we can construct a smaller subgraph induced by u's neighbors

Fig. 2. Running example: colorful *G*. can construct a smaller subgraph induced by *u*'s neighbors Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

whose D_{\min} values are no less than $D_{\min}(u, G)$ to search weak fair cliques. Inspired by this, we design a search order denoted by ColorOD; and we propose an algorithm, called CalColorOD, to calculate such an order. Similar to the idea of ColorfulCore, CalColorOD iteratively removes a vertex with the minimum D_{\min} from the remaining graph. The vertices-removal ordering by this procedure is the ColorOD.

Algorithm 2. CalColorOD

Input: A connected graph G = (V, E)Output: The ColorOD ordering O1: Let *B* be an array with $B(i) = false, 1 \le i \le |V|$; 2: $\mathcal{O} \leftarrow \emptyset$; $H \leftarrow \emptyset$; cnt $\leftarrow 0$; 3: for $u \in V$ do 4: Calculate $D_{\min}(u)$ as lines 4-7 in Algorithm 1; 5: $H.push(u, D_{\min}(u));$ 6: while $H \neq \emptyset$ do 7: $(u, D_{\min}(u)) \leftarrow H.pop();$ $\mathcal{O}[u] = \mathsf{cnt}; B(u) \leftarrow true; \mathsf{cnt}+;$ 8: 9: for $v \in N(u)$ do 10: if B(v) = false then $M_v(u.val, color(u)) - -;$ 11: 12: if $M_v(u.val, color(u)) \leq 0$ then 13: $D_{u.val}(v) - -; dif \leftarrow D_{\min}(v) - D_{u.val}(v);$ 14: if $dif \neq 0$ then 15: $D_{\min}(v) \leftarrow D_{u.val}(v); H.update(v, dif);$ 16: return *O*;

Algorithm 2 outlines the pseudo-code of CalColorOD. For each vertex u, we use $\mathcal{O}(u)$ to indicate the rank of u in our order \mathcal{O} . A heap-based structure H is employed to maintain the vertices with their D_{\min} values, which always pops out the pair $(u, D_{\min}(u))$ with minimum D_{\min} . CalColorOD first calculates $D_{\min}(u)$ for every vertex u and pushes $(u, D_{\min}(u))$ into H (lines 3-5). Then, CalColorOD iteratively pops out the vertex with minimum D_{\min} from Hand records its rank in \mathcal{O} (lines 6-15). As a vertex is removed, we maintain the D_{\min} values for its neighbors and update H (lines 9-15). It is easy to check that the time and space complexities of Algorithm 2 are the same as those of Algorithm 1.

The reason why ColorOD works is that the search procedure beginning with vertices that have low ranks in ColorOD tends to be less possible to form weak fair cliques. Note that the main searching time of the enumeration algorithm is spent on the vertices that have a dense and large neighborhood. ColorOD can guarantee that the unpromising vertices are explored first, thus reducing the number of candidates of the vertices that have a dense and large neighborhood.

3.3 The Weak Fair Clique Enumeration Algorithm

The main idea of WFCEnum is to prune the unpromising vertices first, and then perform the backtracking procedure to find all weak fair cliques. Unlike the traditional maximal clique enumeration, WFCEnum is equipped with a colorful *k*-core-based pruning rule and a carefully-designed ColorOD ordering technique, which can significantly reduce the search space. The pseudo-code of WFCEnum is outlined in Algorithm 3.

Algorithm 3. WFCEnum

Input: G = (V, E, A), an integer k **Output:** The set of weak fair cliques *Res* 1: $Res \leftarrow \emptyset; R \leftarrow \emptyset; X \leftarrow \emptyset; C \leftarrow \emptyset;$ 2: $\hat{G} = (\hat{V}, \hat{E}) \leftarrow \text{ColorfulCore}(G, k-1);$ 3: Initialize an array B with $B(i) = false, 1 \le i \le |\hat{V}|$; 4: for $u \in \hat{V}$ do 5: if B(u) = false then 6: $C \leftarrow \mathsf{ConnectedGraph}(u, B);$ 7: $\mathcal{O} \leftarrow \mathsf{CalColorOD}(C);$ 8: $R \leftarrow \emptyset; X \leftarrow \emptyset; \mathsf{BackTrack}(R, C, X, \mathcal{O});$ 9: return Res; 10: **Procedure** BackTrack (R, C, X, \mathcal{O}) 11: if $C = \emptyset$ and $X = \emptyset$ then $Res \leftarrow Res \cup R$; 12: for $u \in C$ in non-descending ColorOD order do $\hat{R} \leftarrow R \cup u; \hat{C} \leftarrow \emptyset; flag \leftarrow false;$ 13: 14: Let \hat{C}_{cnt} , \hat{R}_{cnt} be the arrays of size A_n ; 15: for $v \in C$ do if $v \in N(u)$ and $\mathcal{O}(v) > \mathcal{O}(u)$ then 16. 17: $C \leftarrow C \cup v; C_{cnt}(v.val) + +;$ 18: if $|\hat{C}| + |\hat{R}| < k \times A_n$ then continue; for $v \in \hat{R}$ do $\hat{R}_{cnt}(v.val)$ ++; 19: 20: for $a_i \in A_{val}$ do 21: if $\hat{R}_{cnt}(a_i) + \hat{C}_{cnt}(a_i) < k$ then 22: $flag \leftarrow true; break;$ 23: if flag = true then continue; 24: $\hat{X} \leftarrow X \cap N(u);$ 25: BackTrack $(\hat{R}, \hat{C}, \hat{X}, \mathcal{O})$; 26: $X \leftarrow X \cup u;$

The WFCEnum algorithm works as follows. It first initializes four sets R, X, C, and Res (line 1). The set R represents the currently-found clique which may be extended to a weak fair clique. X is the set of vertices in which every vertex can be used to expand the current clique R but has already been visited in previous search paths. C is the candidate set that can be used to extend the current clique R in which each vertex must be neighbors of all vertices in R. After initialization, WFCEnum performs ColorfulCore to prune the vertices that are definitely not contained in any weak fair clique (line 2). The algorithm invokes the BackTrack procedure to find all weak fair cliques in the pruned graph G (lines 4-9). Note that G may have several connected colorful (k-1)-cores, so BackTrack should be performed on each connected component in \hat{G} . An array B is used to indicate whether a vertex u has been searched, and it is initialized as false for each vertex. For an unvisited vertex u, WFCEnum identifies the connected colorful-(k-1)-core C containing u and sets B as true for all vertices within C to denote that C will not be searched again (line 6). WFCEnum then calls CalColorOD to derive the search order ColorOD of vertices in C, and performs the BackTrack procedure on C to enumerate all weak fair cliques (lines 7-8).

The workflow of BackTrack is depicted in lines 10-26 of Algorithm 3. It first identifies whether the current R is a weak fair clique (line 11). R is an answer if and only if $C = \emptyset$ and $X = \emptyset$. C is empty means that no vertex can be added into R. In addition, the set X must be empty, otherwise any vertex in X can be added into R and makes R non-maximal. If R is not a weak fair clique, we add each vertex $u \in C$ into R and start the next iteration of BackTrack (lines 12-26). Note that each

candidate in C is a neighbor of all vertices in R, therefore after adding u into R, C must be updated to keep out those vertices that are not adjacent with u (lines 15-17). Here, we only consider the vertices whose rank is larger than u's rank to avoid finding the same clique repeatedly. After obtaining the updated sets \hat{C} and \hat{R} , if $|\hat{C}| + |\hat{R}| < k \times A_n$ holds, BackTrack terminates as the sets cannot reach the minimum size of a weak fair clique (line 18). On the other hand, we use R_{cnt} and C_{cnt} to denote the number of vertices whose attribute value is a_i in \hat{R} and \hat{C} , respectively (line 17 and line 19). By checking the count for each $a_i \in A_{val}$, we can quickly determine whether the current/next clique is promising. For any $a_i \in A_{val}$, if $\hat{R}_{cnt}(a_i) + \hat{C}_{cnt}(a_i) < k$ holds, we cannot obtain a weak fair clique even if we add the whole set C into R. This is because the condition (1) of Definition 1 is not satisfied, thus BackTrack terminates (lines 20-23). Otherwise, the procedure derives the set \hat{X} by adding u's neighbors into X, and then performs the next iteration (lines 24-25). After exploring the vertex u, BackTrack adds it into X because u has already been searched in the current search path and cannot be processed in the following recursions (line 26).

4 STRONG FAIR CLIQUE ENUMERATION

In this section, we first develop an efficient strong fair clique enumeration algorithm with a novel pruning technique for the two-dimensional (2D) case, where the attributed graph has only two types of attributes (i.e., $|A_n| = 2$). Then, we will show how to extend our enumeration algorithm to handle the high-dimensional case ($|A_n| > 2$).

4.1 The Pruning Technique for 2D Case

Suppose that the attributed graph G = (V, E, A) has two types of attributes, i.e., $A_{val} = \{a_1, a_2\}$. The neighbors of a vertex u can be divided into h_u groups by coloring where each group contains vertices with the same color. Clearly, by the property of coloring, only one vertex can be selected from a group to form a clique with *u*. Below, we give a new definition of fairness degree of a vertex.

Definition 6. (Fairness degree) Given a colored attributed graph G = (V, E, A) with $A_{val} = \{a_1, a_2\}$, the fairness degree of u, denoted by FD(u), is the largest number of groups from which we select vertices so that the number of vertices with a_1 is the same as the number of vertices with a_2 .

By Definition 6, we can easily verify that the fairness degree of a vertex u, i.e., FD(u), is an upper bound of the size of the strong fair clique containing *u*. Therefore, for any vertex *u*, if $FD(u) < 2 \times (k-1)$, then *u* cannot be contained in any strong fair clique, because any vertex in a strong fair clique must have a fairness degree no less than $2 \times (k-1)$ by Definition 2. As a consequence, we can safely prune the vertex whose fairness degree is less than $2 \times (k-1)$.

A remaining question is how can we efficiently compute the fairness degree for a vertex *u*. Below, we develop an efficient approach to answer this question.

Based on the attribute values, the h_u color groups can be divided into three categories: (1) OA1Group: is a group that involves vertices of attribute a_1 only; (2) OA2Group: is a group that contains vertices of attribute *a*₂ only; (3) MixGroup: is a group that contains vertices of must be contained in the fairness *k*-core. Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

both a_1 and a_2 . Let c_1 , c_2 , and c_m be the number of the OA1Group groups, the OA2Group groups, and the MixGroup groups respectively. Suppose without loss of generality that $c_1 \leq c_2$. Then, if $c_m \leq (c_2 - c_1)$ holds, we can easily derive that $FD(u) = 2 \times (c_m + c_1)$. Otherwise, we have $FD(u) = 2 \times ((c_m - (c_2 - c_1))/2 + c_2)$. Based on these results, we can calculate the fairness degree for each vertex by using the three quantities c_1 , c_2 , and c_m . The pseudo-code of our FairDegCal algorithm to compute the fairness is given in lines 17-29 of Algorithm 4.

Algorithm 4. FairnessCore

Input: G = (V, E, A), an integer k **Output:** The reduced graph \hat{G}

1: $\overline{G} = (\overline{V}, \overline{E}) \leftarrow \mathsf{ColorfulCore}(G, k);$

2: Let *FD* be an array of size $|\overline{V}|$; Let Q be a queue;

```
3: for u \in \overline{V} do
```

- 4: for $v \in N(u)$ do
- 5: Group(u, color(v), v.val)++;
- 6: $FD(u) \leftarrow \mathsf{FairDegCal}(u, \mathsf{Group});$
- 7: if $FD(u) < 2 \times k$ then
- 8: Remove *u* from \overline{G} ; *Q*.*push*(*u*);
- 9: while $Q \neq \emptyset$ do
- $u \leftarrow Q.pop();$ 10:
- 11: for $v \in N(u)$ do
- 12: if *v* is removed then continue;
- 13: Group(v, color(u), u.val) - -;
- 14: Calculate FD(v) and update Q as lines 6-8;
- 15: $G \leftarrow$ the remaining graph of G;
- 16: return \hat{G} ;
- 17: **Procedure** FairDegCal(*u*, Group)
- 18: $c_1 \leftarrow 0$; $c_2 \leftarrow 0$; $c_m \leftarrow 0$;
- 19: for each color cr do
- 20: if $Group(u, cr, a_1) \ge 1$ and $Group(u, cr, a_2) = 0$ then $c_1 \leftarrow c_1 + 1;$
- 21: if $Group(u, cr, a_2) \ge 1$ and $Group(u, cr, a_1) = 0$ then $c_2 \leftarrow c_2 + 1;$
- 22: if $Group(u, cr, a_1) \ge 1$ and $Group(u, cr, a_1) \ge 1$ then $c_m \leftarrow c_m + 1;$
- 23: if $c_1 \leq c_2$ then
- 24: if $c_m \ge (c_2 - c_1)$ then
 - $FD(u) \leftarrow 2 \times ((c_m (c_2 c_1))/2 + c_2);$
- 25: else $FD(u) \leftarrow 2 \times (c_m + c_1);$
- 26: else
- 27: if $c_m \ge (c_1 - c_2)$ then $FD(u) \leftarrow 2 \times ((c_m - (c_1 - c_2))/2 + c_1)$;
- 28: else $FD(u) \leftarrow 2 \times (c_m + c_2);$

29: return *FD*(*u*);

With the fairness degree, we can iteratively prune the vertices with fairness degrees smaller than $2 \times (k-1)$. Below, we introduce a concept called fairness k-core to characterize the reduced subgraph after iteratively peeling the unqualified vertices.

Definition 7. (fairness k-core) Given an attributed graph G =(V, E, A) with $A_{val} = \{a_1, a_2\}$ and an integer k, a subgraph $H = (V_H, E_H, A)$ of G is a fairness k-core if: (1) for each $u \in$ V_H , $FD(u) \ge 2k$; (2) there is no subgraph $H' \subseteq G$ that satisfies (1) and $H \subset H'$.

By Definition 7, we can show that any strong fair clique

Lemma 2. Given an attributed graph G = (V, E, A) with $A_{val} =$ $\{a_1, a_2\}$ and a parameter k, any strong fair clique must be contained in the fairness (k-1)-core of G.

Similar to the colorful k-core computation algorithm, we can also devise a peeling algorithm to compute the fairness k-core by iteratively removing the vertices that have fairness degrees smaller than 2k. The pseudo-code of our algorithm is outlined in Algorithm 4. Note that a strong fair clique is always contained in a weak fair clique, thus we can first invoke ColorfulCore to prune vertices that are definitely not included in the weak fair cliques before computing the fairness k-core of G (line 1).

Theorem 2. Algorithm 4 consumes $O((E+V) \times \text{color})$ time using $O(V \times \text{color})$ space.

Fairness *k*-Core Ordering. Similar to the ColorOD, we can derive an ordering based on the fairness k-core, called FairOD, for strong fair clique enumeration. In particular, FairOD is derived by iteratively removing the vertex with the minimum fairness degree which is very similar to the computational procedure of ColorOD. We omit the details for brevity.

4.2 The Enumeration Algorithm for 2D Case

Armed with the fairness k-core based pruning technique and the FairOD ordering, we propose the SFCEnum algorithm which alternatively picks a vertex of a specific attribute in the backtracking procedure to enumerate all strong fair cliques. The SFCEnum is shown in Algorithm 5. We use R to represent the currently-found clique and C to denote the candidate set. Similar to WFCEnum, SFCEnum first applies FairnessCore to prune the vertices that are definitely not contained in strong fair cliques (line 2) and then performs the StrongBackTrack procedure for each connected fairness (k-1)-core in G to find all results (lines 4-8).

The pseudo-code of StrongBackTrack is outlined in lines 10-27 of Algorithm 5. Since a strong fair clique requires that the numbers of vertices for each attribute a_i are exactly the same, we develop a novel attribute-alternatively-selection mechanism to select vertices in each iteration. That is, StrongBackTrack admits an input parameter a_{ϕ} , which is initialized to a_0 (line 8), to indicate the attribute value of the vertices to be selected in the current iteration. In the next iteration, we pick the vertices with the attribute value $a_{\phi+1}$ to construct strong fair cliques (line 27). StrongBackTrack divides the candidates in C into A_n sets, where the attribute values of vertices in each set are the same, i.e., $C_A(a_i) =$ $\{u|u \in C, u.val = a_i\}$ (line 14). For each candidate u in $C_A(a_{\phi})$, we pick one vertex at a time as a part of the currently-found clique and update the candidate set based on the FairOD ordering (lines 16-27).

After adding *u* into the current clique, we can combine the set \hat{R} and \hat{C} to determine whether to call StrongBackTrack for a more in-depth search (lines 16-27). Specifically, we classify the candidates in \hat{C} according to their attribute values and record a_{\min} as the attribute value with the minimum number of vertices (denoted by c_{\min}) (line 20). Note that if there are multiple attribute values satifying $|C_A(a_i)| = c_{\min}$, we pick a_i with the largest *i* as a_{\min} . Clearly, c_{\min} determines how large a strong fair clique can empty candidate set C does not mean that we find a strong Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

be. We use R_c to denote the largest size of possible strong fair cliques. If $|\hat{R}| \% A_n = 0$, the numbers of vertices with various attribute values are the same in the current set R_{i} thus there are at most $c_{\min} \times A_n$ vertices can be added into \hat{R} , and further we have $R_c = c_{\min} \times A_n + |\hat{R}|$ (line 21). Otherwise, we calculate R_c and try to search a larger clique (lines 22-27). By the attribute-alternatively-selection strategy, in the current iteration with a_{ϕ} , the number of vertices with attribute value a_f ($a_f \in \{a_0, ..., a_{\phi}\}$) is always one more than that of vertices with a_b ($a_b \in \{a_{\phi+1}, ..., a_{n-1}\}$) in *R*. If $a_{min} = a_f$, we can add one vertex, for each a_b , into *R* to obtain a clique with size $(|\hat{R}|/A_n + 1) \times A_n$, which is denoted by R_M . Note that there are still $c_{min} \times A_n$ vertices that may form a larger clique with R_M . Therefore, we calculate R_c as shown in line 24. Similarly, when $a_{min} = a_{br}$ we have at most $(c_{min} - 1) \times A_n$ vertices that may add into R_M to construct a strong fair clique with size R_c (line 25). After calculating R_{c} , we can terminate the search procedure early if $R_c < k \times A_n$, because it violates the definition of strong fair clique in this case. Otherwise, we recursively perform StrongBackTrack with the attribute value $a_{\phi+1}$ (line 27).

Alg	gorithm 5. SFCEnum
	Input: $G = (V, E, A)$, an integer k
	Output: The set of all strong fair cliques <i>Res</i>
1:	$Res \leftarrow \emptyset; R \leftarrow \emptyset; C \leftarrow \emptyset;$
2:	$\hat{G} = (\hat{V}, \hat{E}) \leftarrow FairnessCore(G, k-1);$
3:	Initialize an array <i>B</i> with $B(i) = false, 1 \le i \le \hat{V} $;
4:	for $u \in \hat{V}$ do
5:	if $B(u) = false$ then
6:	$C \leftarrow ConnectedGraph(u, B);$
7:	$\mathcal{O} \leftarrow FairOD(C);$
8:	$R \leftarrow \emptyset; C \leftarrow \emptyset; StrongBackTrack(R, C, a_0, \mathcal{O});$
9:	return Res;
10:	Procedure StrongBackTrack $(R, C, a_{\phi}, \mathcal{O})$
11:	if $ R \% A_n = 0$ and $ R \ge k \times A_n$ then
12:	if $IsMaximal(C)$ then
13:	$Res \leftarrow Res \cup R$; return;
14:	for $u \in C$ then $C_A(u.val) \leftarrow C_A(u.val) \cup u$;
15:	for $u \in C_A(a_{\phi})$ do
16:	$\ddot{R} \leftarrow R \cup u;$
17:	for $v \in C$ do
18:	if $v \in N(u)$ and $\mathcal{O}(v) > \mathcal{O}(u)$ then
19:	$C \leftarrow C \cup v; C_A(v.val) \leftarrow C_A(v.val) \cup v;$
20:	$c_{\min} \leftarrow \min(C_A(a_i)); a_{\min} \leftarrow \arg\min_{a_i} C_A(a_i) ;$
21:	if $ R \% A_n = 0$ then $R_c \leftarrow c_{\min} \times A_n + R $;
22:	else
23:	if $a_{\min} \in \{a_0, a_1,, a_{\phi}\}$ then
24:	$R_c \leftarrow c_{\min} \times A_n + (\ddot{R} /A_n + 1) \times A_n;$
25:	else $R_c \leftarrow (c_{\min} - 1) \times A_n + (\ddot{R} /A_n + 1) \times A_n;$
26:	if $R_c < k \times A_n$ then continue;
27:	StrongBackTrack($\hat{R}, C, a_{\phi+1}, \mathcal{O}$);

Maximality Checking. The results of all traditional maximal cliques and our weak fair cliques lie in the leaves of the backtracking enumeration tree. We can check whether a weak fair clique is found by $C = \emptyset$ and $X = \emptyset$ (see line 11 of Algorithm 3). However, such a maximality checking method cannot be used for strong fair cliques. The reasons are twofold: (1) an empty candidate set C does not mean that we find a strong fair clique because the number of vertices in R corresponding to each attribute value may not be the same; (2) even if X is not empty, R can be a strong fair clique. That is to say, strong fair cliques can appear in the intermediate nodes of the backtracking enumeration tree. Therefore, we need to develop new solution to check the maximality for strong fair cliques. We propose a maximality checking technique as follows.

Algorithm 6. IsMaximal(C)

1:	if $ C < A_n$ then return true;
2:	else
3:	for each $a_i \in A_{val}$ do
4:	$C_i \leftarrow \{u u \in C, u.val = a_i\};$
5:	if $ C_i = 0$ return true;
6:	$Record \leftarrow C_0;$
7:	for each $a_i \in \{A_{val} - \{a_0\}\}$ do
8:	$SwapRecord \leftarrow \emptyset;$
9:	for $v_i \in C_i$ do
10:	for $r \in Record$ do
11:	if v_i is a neighbor of all vertices in r then
12:	$SwapRecord \leftarrow SwapRecord \cup \{r \cup v_i\};$
13:	$Record \leftarrow SwapRecord;$
14:	if $Record \neq \emptyset$ return false;

Once the StrongBackTrack procedure finds a clique whose size is equal to $k' \times A_n$ with $k' \ge k$, we need to check the maximality according to Definition 2. Since the vertices in C are neighbors of all vertices in R, if we find any clique in C with every attribute, R is definitely not a strong fair clique as it violates the constraint (3) in Definition 2. Based on this, we propose a verification method, called IsMaximal, which is shown in Algorithm 6. Specifically, if the size of C is less than A_n , which means adding all vertices in C will destroy the fairness property of R, R is maximal and thus the algorithm returns true (line 1). Otherwise, we need to explore the common neighbors to find if there exist cliques with size at least $A_n + |R|$ that are also strong fair cliques. The IsMaximal algorithm uses C_i to rep resent the vertices in C with the attribute value a_i . Clearly, i $|C_i| = 0$ holds for an arbitrary attribute a_i , the attribute constraint will not be satisfied and the procedure outputs true, indicating R is maximal (lines 3-5). Otherwise, StrongBackTrack tries to construct cliques from C. The variables Record and SwapRecord are used to maintain the current partial cliques Finally, if *Record* is not empty, we can find a clique with size a least $A_n + |R|$. In such case, R is not a strong fair clique and the StrongBackTrack procedure returns false (lines 6-14).

4.3 Handling the High-Dimensional Case

We note that the idea of the fairness degree based pruning rule is not easy to extend to the high-dimensional case because there may be $2^{A_n} - 1 - A_n$ MixGroups in the worst case. Therefore, it is very difficult to compute the exact fairness degree for each vertex when $A_n > 2$. To circumvent this problem, we propose a heuristic greedy algorithm to calculate an approximation of the fairness degree for each vertex *u*, instead of deriving the exact fairness degree.

Specifically, we let GD(u) be the approximate fairness degree computed by our greedy algorithm. By coloring, the neighbors of a vertex u can be classified into h_u color groups. For each color *cr*, we have a group, denoted by high-dimensional case. However, we can use ColorfulCore Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

Group(cr). For a color group Group(cr), we let S(cr) be the set of attributes of the vertices in Group(cr). For an attribute a_i , if $a_i \in S(cr)$ and |S(cr)| = 1 hold, we know that the group Group(cr) only contains the vertices with the attribute a_i . For each attribute a_i , we maintain a counter $cnt(a_i)$ to record the number of color groups that only contain vertices with a_i . Clearly, |S(cr)| > 1 indicates a mix group Group(cr). The greedy algorithm greedily assigns Group(cr) to the attribute with the minimum number of color groups. In other words, the algorithm increases the counter of a_m by 1 where $a_m = \arg \min_{a_i \in S(cr)} \operatorname{cnt}(a_i)$. Finally, GD(u) is obtained by taking the minimum counter over all attributes, i.e., $GD(u) = \min\{\mathsf{cnt}(a_i), a_i \in A_{val}\}.$

It is easy to see that the approximate fairness degree GD(u) of a vertex u is always no larger than the exact fairness degree of u, thus it cannot be directly used to prune vertices for strong fair clique enumeration. This is because GD(u) is not an upper bound of the size of the strong fair cliques containing *u*. However, we can use the approximate fairness degrees to derive a good heuristic ordering, because the vertices with high exact fairness degrees tend to have high approximate fairness degrees. Such a heuristic ordering can be applied to reduce the search space for strong fair clique enumeration, as confirmed in our experiments. Specifically, to obtain the heuristic ordering denoted by HeurOD, we can iteratively delete the vertex with the minimum GD (similar to the procedure of computing ColorOD and FairOD). The pseudo-code of our greedy algorithm to generate HeurOD is given in Algorithm 7.

า า	Alg	orithm 7. CalHeurOrd
-		Input: A connected graph $G = (V, E)$
е		Output: The HeurOD ordering \mathcal{O}
s	1:	$\mathcal{O} \leftarrow \emptyset; \mathcal{Q} \leftarrow \emptyset;$
ę	2:	Let <i>B</i> be an array with $B(i) = false, 1 \le i \le V $;
_	3:	for $u \in V$ do
f	4:	for $v \in N(u)$ do
_	5:	$S_u(color(v), v.val) \leftarrow S_u(color(v), v.val) + 1;$
-	6: 7:	Let cnt be an array with cnt(i) = 0, $0 \le i < A_n$; for each color cr do
<	8:	for $a_i \in A_{val} \operatorname{do}$
1	9:	if $S_u(cr, a_i) \ge 1$ then
	10:	$a_m = \arg\min_{a_i \in S_u(cr,a_i)} \operatorname{cnt}(a_i);$
t	11:	$cnt(a_m) \leftarrow cnt(a_m) + 1;$
5	12:	$GD(u) = \min\{\operatorname{cnt}(a_i), a_i \in A_{val}\};$
	13:	$\mathcal{Q}.push(u,GD(u));$
	14:	while $\mathcal{Q} \neq \emptyset$ do
	15:	$u \leftarrow \mathcal{Q}.pop(); \mathcal{O}.push(u); B(u) \leftarrow true;$
	16:	for $v \in N(u)$ do
,	17:	if $B(v) = false$ then
	18:	$S_v(color(u), u.val) \leftarrow S_v(color(u), u.val) - 1;$
, t	19:	Calculate $GD(v)$ and update Q as lines 6-13;

Theorem 3. Algorithm 7 takes $O((V + E) \times A_n \times \text{color})$ using $O(V \times A_n \times \text{color})$ space.'

20: return \mathcal{O} ;

The Enumeration Algorithm. Algorithm 5 can be easily extended to handle the high-dimensional case. Note that FairnessCore and FairOD in Algorithm 5 do not work for the high-dimensional case. However, we can use ColorfulCore

(Algorithm 1), which is designed for pruning unpromising vertices in weak fair clique enumeration, to reduce search space because a strong fair clique is always contained in a weak fair clique. In addition, we use the ordering HeurOD computed by Algorithm 7 for strong fair clique enumeration with $A_n > 2$. Clearly, the StrongBackTrack procedure with the attribute-alternatively-selection strategy in Algorithm 5 can be directly applied to handle the $A_n > 2$ case. Therefore, we only need to slightly modify Algorithm 5 to enumerate strong fair cliques for the high-dimensional attributes. Specifically, in Algorithm 5, we use ColorfulCore instead of FairnessCore to prune the unpromising vertices (line 2), and invoke Algorithm 7 to obtain the HeurOD ordering to reduce the search space (line 7).

5 RELATIVE FAIR CLIQUE ENUMERATION

In this section, we first develop an enhanced pruning technique for the case of two-dimensional (2D) attributes to prune the unpromising vertices in the original graph. Then, two search frameworks with different strategies, namely, RFCRefineEnum and RFCAlterEnum, are proposed to enumerate relative fair cliques for both 2D and high-dimensional attributes.

5.1 The Enhanced Pruning Technique for 2D Case

Suppose that the attributed graph G = (V, E, A) with $A_{val} = \{a_1, a_2\}$, and we also divide the neighbors of a vertex u into h_u groups where each group contains vertices with the same color. Below, we define the enhanced colorful degree as follows.

Definition 8. (Enhanced colorful degree) Given a colored attributed graph G = (V, E, A) with $A_{val} = \{a_1, a_2\}$, the enhanced colorful degree of u, denoted by ED(u), is the minimum number of groups that assigned to either to attribute a_1 or to attribute a_2 .

For a vertex u, as only one vertex in a group can be selected to form a clique with u, the number of groups assigned to an arbitrary attribute is no greater than the number of u's neighbors with this attribute. And further, the enhanced colorful degree is no larger than the minimum colorful degree, thus it determines a tighter upper bound of the size of the relative fair clique containing u. By Definition 3, the enhanced colorful degree of any vertex in a relative fair clique is no less than (k - 1). Consequently, we can safely prune the vertex whose enhanced colorful degree is less than (k - 1). Below, we introduce an algorithm, called EnhancedColCal, to compute the enhanced colorful degree for a vertex u.

The pseudo-code of EnhancedColCal is outlined in Algorithm 8. Similar to FairDegCal, we divide h_u color groups into three categories, i.e., OA1Group, OA2Group and MixGroup, and denote the number of the groups in these three categories by c_1 , c_2 , and c_m . The main idea of EnhancedColCal is to assign each color group in the MixGroup to OA1Group or OA2Group when c_1 or c_2 is less than k. We take c_1 as an example. In the case of $c_1 < k$, if $c_m \ge k - c_1$ holds, we assign $k - c_1$ groups in MixGroup to OA1Group (line 6); otherwise, we assign all groups in MixGroup to OA2Group with attribute a_2 , we also use c_m to expand c_2 as we expand c_1 (lines 9-13). Finally, we can easily derive that $ED(u) = \min \{c_1, c_2\}$.

Based on the enhanced colorful degree, we define the enhanced colorful *k*-core in the following.

Definition 9. (Enhanced colorful k-core) Given an attributed graph G = (V, E, A) with $A_{val} = \{a_1, a_2\}$ and an integer k, a subgraph $H = (V_H, E_H, A)$ of G is an enhanced colorful k-core if: (1) for each $u \in V_H$, $ED(u) \ge k$; (2) there is no subgraph $H' \subseteq G$ that satisfies (1) and $H \subset H'$.

By Definition 9, we hold the following lemma, that is, any relative fair clique must be contained in the enhanced color-ful (k - 1)-core.

Algorithm 8. EnhancedColCal

1: **Procedure** EnhancedColCal(*u*, Group, *k*) 2: $c_1 \leftarrow 0$; $c_2 \leftarrow 0$; $c_m \leftarrow 0$; 3: Calculate c_1, c_2, c_m as lines 19-22 of Algorithm 4; 4: if $c_1 < k$ then 5: if $c_m \ge (k - c_1)$ then 6: $c_1 \leftarrow k; c_m \leftarrow c_m - (k - c_1);$ 7: else 8: $c_1 \leftarrow c_1 + c_m; c_m \leftarrow 0;$ 9: if $c_2 < k$ then 10: if $c_m \geq (k - c_2)$ then 11: $c_2 \leftarrow k; c_m \leftarrow c_m - (k - c_2);$ 12: else 13: $c_2 \leftarrow c_2 + c_m; c_m \leftarrow 0;$ 14: $ED(u) \leftarrow \min\{c_1, c_2\};$ 15: return ED(u);

Lemma 3. Given an attributed graph G = (V, E, A) with $A_{val} = \{a_1, a_2\}$ and a parameter k, any relative fair clique must be contained in the enhanced colorful (k - 1)-core of G.

We also derive a peeling algorithm, i.e., EnhancedCore, to compute the enhanced colorful *k*-core. The pseudo-code of EnhancedCore is similar to that of ColorfulCore (Algorithm 4) and we only need to make slightly modifying as follows. Specifically, in line 6, we perform the procedure EnhancedColCal (Algorithm 8) instead of FairDegCal to calculate the enhanced colorful degrees of all vertices. In line 7, we modify the condition to be ED(u) < k to add the vertices with initial enhanced colorful degrees less than k to the queue Q. Then, we iteratively remove the vertices with the enhanced colorful degrees less than k, and maintain the enhanced colorful degrees for their neighbors and the queue Q (line 14). Due to the space limitation, we omit the pseudo-code of EnhancedCore.

Example 2. Reconsider the attributed graph in Fig. 2. Suppose that we search all relative fair cliques with k = 4. We need to calculate 3-colorful core or 3-enhanced colorful core first. Take vertex v_9 as an example. v_9 has four neighbors with attribute a, i.e., v_{10} , v_{11} , v_{13} and v_{14} , and three neighbors with attribute b, i.e., v_5 , v_6 and v_{12} . Based on Definition 4, we have $D_a(v_9) = 4$ and $D_b(v_9) = 3$, and further $D_{\min}(v_9) = D_b(v_9) = 3$. Due to $D_{\min}(v_9) = 3$, v_9 cannot be removed according to the colorful core pruning technique (Definition 5). However, v_9 is not contained in any 4-relative fair clique. This is because v_6 with attribute b and v_{10} with attribute a have the same color (green), that is, there are no edge between them, thus v_6 and v_{10} cannot coexist in a clique. Analogously, the neighbors colored yellow, i.e.,

 v_{12} with attribute *b* and v_{14} with attribute *a*, also cannot form a clique. While considering the enhanced colorful degree, we have $ED(u_9) = 2$. Clearly, $ED(u_9) = 2 < 3$, thus EnhancedCore can safely remove v_9 from *G*. Hence, the enhanced colorful degree has a stronger pruning effect than the colorful degree. EnhancedCore repeatedly removes vertices until all the remaining vertices satisfying $ED(*) \ge$ 3. Finally, we can obtain an enhanced colorful 3-core induced by $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$.

Theorem 4. The EnhancedCore algorithm consumes $O((E + V) \times \text{color})$ time using $O(V \times \text{color})$ space.

Remark. Note that the EnhancedCore pruning technique is more efficient than ColorfulCore, because the enhanced colorful degree provides a tighter upper bound on the minimum number of neighbors of *u* for arbitrary attributes benefitting from the property of graph coloring. In addition, the EnhancedCore can work on all proposed fairness-aware clique models in the case of 2D attributes. Specifically, in weak fair clique enumeration, we can use EnhancedCore instead of ColorfulCore to achieve a stronger pruning effect in line 2 of Algorithm 3. In Algorithm 4 for strong fair clique enumeration, we can also apply EnhancedCore in line 1. For the relative fair clique search, we will introduce the enumeration algorithms in the following subsections which are also equipped with the EnhancedCore pruning technique.

Algorithm 9. RFCRefineEnum

Input: G = (V, E, A), two integers k and δ **Output:** The set of relative fair cliques *Res* 1: $Res \leftarrow \emptyset; C \leftarrow \emptyset;$ 2: if $\delta = 0$ then $Res \leftarrow SFCEnum(G, k)$; return Res; 3: if $|A_{val}| = 2$ then $\hat{G} = (\hat{V}, \hat{E}) \leftarrow \text{EnhancedCore}(G, k-1);$ 4: else $\hat{G} = (\hat{V}, \hat{E}) \leftarrow \text{ColorfulCore}(G, k-1);$ 5: $C \leftarrow \mathsf{WFCEnum}(\hat{G}, k);$ 6: for $C_i \in C$ do 7: for $a_i \in A_{val}$ do 8: $V(a_i) \leftarrow \emptyset; Cnt(a_i) \leftarrow 0;$ 9: for $u \in C_i$ do Cnt(u.val)++; $V(u.val) \leftarrow V(u.val) \cup \{u\}$; 10: $a_{min} \leftarrow \arg\min_{a_i \in A_{val}} Cnt(a_i); a_{max} \leftarrow a_{min} + \delta;$ 11: 12: $L_A \leftarrow \{a_i \in A_{val} | Cnt(a_i) > a_{max}\};$ 13: if $L_A = \emptyset$ then 14: $Res \leftarrow C_i$; continue; 15: $\mathcal{V}(L_A) \leftarrow \emptyset; \mathcal{V}(L_A) \leftarrow \mathcal{V}(L_A) \cup \{V(a_i) | a_i \in L_A\};$ 16: $\mathcal{V}(C_P) \leftarrow \emptyset; \mathcal{V}(C_P) \leftarrow \mathcal{V}(C_P) \cup \{V(a_i) | a_i \notin L_A\};$ 17: Let a_s be the first attribute element in L_A ; 18: $\mathsf{DeepRFCRefine}(\mathcal{V}(C_P), \mathcal{V}(L_A), L_A, a_{max}, Res, a_s, 0);$ 19: return Res; 20: **Procedure DeepRFCRefine** $(C, \mathcal{V}(L_A), L_A, a_{max}, Res, a_c, cnt_c)$ 21: if a_c is the last attribute element in L_A then 22: $Res \leftarrow Res \cup C$; return; 23: for $u \in V(a_c)$ do $C \leftarrow C \cup \{u\}; V(a_c) \leftarrow V(a_c) - \{u\};$ 24: 25: if $cnt_c + 1 < a_{max}$ then 26: DeepRFCRefine($C, \mathcal{V}(L_A), L_A, a_{max}, Res, a_c, cnt_c + 1$); 27: else 28: $a_{nc} \leftarrow$ the next attribute element of a_c in L_A ; 29: DeepRFCRefine $(C, \mathcal{V}(L_A), L_A, a_{max}, Res, a_{nc}, 0)$; 30: $C \leftarrow C - \{u\}; V(a_c) \leftarrow V(a_c) \cup \{u\};$

5.2 The RFCRefineEnum Algorithm

Reviewing Definition 3, a relative fair clique must be contained in weak fair cliques. Therefore, a feasible idea is to find all the weak fair cliques, and then enumerate the relative fair cliques contained in them. Following this idea, we propose an algorithm, called RFCRefineEnum, which is shown as Algorithm 9.

The RFCRefineEnum algorithm works as follows. If $\delta = 0$, it performs SFCEnum to find all relative fair cliques since the relative fair clique model is equivalent to the strong fair clique model in this case (line 2); otherwise, the RFCRefineEnum performs EnhancedCore or ColorfulCore to prune the original graph for 2D or high-dimensional attributes (lines 3-4). Then it invokes WFCEnum to find all weak fair cliques and refines relative fair cliques contained in them (lines 5-18). For each weak fair clique C_i , the RFCRefineEnum computes the number of vertices $Cnt(a_i)$ for each attribute a_i , and identifies the minimum $Cnt(a_i)$ as a_{min} . Based on a_{min} and δ , at most how many vertices of each attribute has in a relative fairness clique is determined, which we denoted by a_{max} (lines 7-11). The algorithm then collects those attributes with the number of vertices greater than a_{max} into L_A , which we call the lacking attribute set (line 12). Clearly, if L_A is empty, the current weak fair clique is a (k, δ) -relative fair clique, and we add it into the result set *Res* (line 14). In the negative case, **RFCRefineEnum** refines the vertices with lacking attributes and non-lack attributes into $\mathcal{V}(L_A)$ and $\mathcal{V}(C_P)$, respectively (lines 15-16). It then selects a lacking attribute $a_s \in L_A$ and performs the DeepRFCRefine procedure to expand the partial clique induced by $\mathcal{V}(C_P)$ to search all relative fair cliques (lines 17-18).

Algorithm 10. RFCAlterEnum

Input: G = (V, E, A), two integers k and δ Output: The set of relative fair cliques Res 1: $Res \leftarrow \emptyset; R \leftarrow \emptyset; X \leftarrow \emptyset; C \leftarrow \emptyset;$ 2: if $\delta = 0$ then $Res \leftarrow SFCEnum(G, k)$; return Res; 3: if $|A_{val}| = 2$ then $\hat{G} = (\hat{V}, \hat{E}) \leftarrow \text{EnhancedCore}(G, k-1);$ 4: else $\hat{G} = (\hat{V}, \hat{E}) \leftarrow \text{ColorfulCore}(G, k-1);$ 5: Initialize an array B with $B(i) = false, 1 \le i \le |V|$; 6: for $u \in \hat{V}$ do 7: if B(u) = false then $C \leftarrow \mathsf{ConnectedGraph}(u, B);$ 8: 9: $\mathcal{O} \leftarrow \mathsf{CalColorOD}(C);$ 10: $R \leftarrow \emptyset; X \leftarrow \emptyset;$ 11: DeepRFCAlter $(R, C, X, \mathcal{O}, a_0, -1)$; 12: return Res;

In the DeepRFCRefine procedure, there are two important parameters: a_c and cnt_c . The parameter a_c indicates that the current round needs to pick a vertex with the lacking attribute a_c into the partial clique *C*. And cnt_c is used to record the number of vertices with the lacking attribute a_c . In each recursion of DeepRFCRefine, it tries to add each vertex *u* with attribute a_c to *C* to perform a deeper search for relative fair clique enumeration (lines 23-30). If $cnt_c + 1 < a_{max}$, that means the number of vertices with a_c in *C* has not yet reached a_{max} , thus we perform the DeepRFCRefine with the parameters: a_c and $cnt_c + 1$ (line 26). On the other hand, once the number of vertices with a_c in *C* is up to a_{max} , we invoke the DeepRFCRefine to select vertices for the next lacking attribute a_{nc} with

 cnt_c equals 0 (lines 28-29). When all lacking attributes in L_A are processed, a relative fair clique C is found and the DeepRFCRefine adds it into the result set Res (lines 20-21).

5.3 The RFCAlterEnum Algorithm

The RFCRefineEnum algorithm is not very efficient for relative fair clique enumeration because a relative fair clique may be contained in many weak fair cliques, which causes a lot of repeated enumeration calculation in RFCRefineEnum. To solve this issue, we propose the RFCAlterEnum algorithm which applies the attribute-alternative-selection search method in SFCEnum to find all relative fair cliques.

Algorithm 11. DeepRFCAlter

1: **Procedure** DeepRFCAlter $(R, C, X, \mathcal{O}, a_{\phi}, a_{max})$ 2: for $u \in C$ do $C_A(u.val) \leftarrow C_A(u.val) \cup u$; 3: for $u \in R$ do $R_A(u.val) \leftarrow R_A(u.val) \cup u$; 4: if $|C_A(a_{\phi})| = 0$ and $a_{max} = -1$ then $a_{min} \leftarrow |R_A(a_\phi)|;$ 5: 6: $a_{max} \leftarrow a_{min} + \delta;$ 7: for $a_i \in A_{val}$ do if $|R_A(a_i)| = a_{max}$ then $C \leftarrow C - C_A(a_i)$; $C_A(a_i) \leftarrow \emptyset$; 8: 9: if $C = \emptyset$ then 10: $isMaximal \leftarrow true;$ 11: if $X \neq \emptyset$ then 12: $a_{min} \leftarrow \min_{a_i \in A_{val}} |R_A(a_i)|;$ 13: for $u \in X$ do if $u.val = a_{min}$ or $|R_A(u.val)| + 1 < = a_{max}$ then 14:15: $isMaximal \leftarrow false; break;$ 16: if isMaximal = true then $Res \leftarrow Res \cup R$; return; if $C_A(a_{\phi}) = \emptyset$ then DeepRFCAlter $(R, C, X, \mathcal{O}, a_{\phi+1}, a_{max})$; 17: return; 18: for $u \in C_A(a_\phi)$ do $\hat{R} \leftarrow R \cup u; \hat{C} \leftarrow \emptyset; flag \leftarrow false;$ 19: for $v \in C$ do 20: 21: if $v \in N(u)$ and $\mathcal{O}(v) > \mathcal{O}(u)$ then 22: $\hat{C} \leftarrow \hat{C} \cup v; \hat{C}_{cnt}(v.val) + +;$ 23: if $|C| + |R| < k * A_n$ then continue; 24: for $v \in \hat{R}$ do $\hat{R}_{cnt}(v.val)$ ++; 25: for $a_i \in A_{val}$ do 26: if $\hat{R}_{cnt}(a_i) + \hat{C}_{cnt}(a_i) < k$ then 27: $flag \leftarrow true; break;$ 28: if flag = true then continue; 29: $\ddot{X} \leftarrow X \cap N(u);$ DeepRFCAlter $(\hat{R}, \hat{C}, \hat{X}, \mathcal{O}, a_{\phi+1}, a_{max});$ 30: 31: $X \leftarrow X \cup u;$

The RFCAlterEnum algorithm is outlined in Algorithm 10. Similar to WFCEnum and SFCEnum, *R* is the currentlyfound clique and C is the candidate set that can be used to extend R. All the relative fair cliques are stored in the set *Res.* To avoid the repeated enumeration, we still use the set X to maintain the vertices that can be used to expand the current clique R but have already been visited in previous search paths. The RFCAlterEnum algorithm performs SFCEnum directly to find all relative fair cliques for $\delta = 0$ like the RFCRefineEnum (line 2). In other cases, it first removes the vertices that are definitely not contained in any relative fair clique with the pruning techniques. For the graph G with two types of attributes, that is, $|A_{val}| = 2$, technique FairnessCore (Algorithm 4) and the ordering Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

RFCAlterEnum performs EnhancedCore to prune the original graph (line 3), and ColorfulCore is called for highdimensional attributes (line 4). Then, the RFCAlterEnum alternatively selects a vertex of a specific attribute in each backtracking round to enumerate all relative fair cliques, i.e., the DeepRFCAlter procedure (lines 6-11).

The workflow of the DeepRFCAlter procedure is depicted in Algorithm 11. The input parameter a_{ϕ} is used to indicate the attribute value of the vertices to be selected in the current iteration. a_{max} is the upper bound of the number of vertices for an arbitrary attribute a_i in the current search space, which is initialized to -1 (line 11 in Algorithm 10). In each iteration with attribute a_{ϕ} , the DeepRFCAlter procedure first divides the vertices in the candidate set C and the current partial clique R into A_n collections according to their attributes, respectively (lines 2-3). For the specified a_{ϕ} , if the current candidate set has no vertex with a_{ϕ} and a_{max} is equal to the initial -1, that means the lower bound of the number of vertices for an arbitrary attribute a_i is determined. And further, a_{max} is also fixed based on the difference threshold δ (lines 4-5). The DeepRFCAlter then identifies whether the number of vertices for attribute $a_i \in A_{val}$ in the current clique R has reached a_{max} . In the affirmative case, adding any vertex with a_i to R would violate the definition of a relative fair clique, and thus the procedure removes all vertices with a_i from the candidate set C (lines 6-7). Since a_{ϕ} is specified for the current round, for each candidate u in $C_A(a_{\phi})$, the DeepRFCAlter picks one vertex at a time to add to the currently-found clique and call itself to perform a deeper search for the next attribute $a_{\phi+1}$ (lines 18-31). Note that if $C_A(a_{\phi})$ is empty, the DeepRFCAlter directly invokes a recursion by specifying the attribute $a_{\phi+1}$ (lines 16-17).

Maximality checking. Once the candidate set C is empty, we check the maximality of R. As previously mentioned, the vertices in X can expand R but have already been visited in previous search paths. Thus, we check the maximality by adding each vertex in X to R (lines 8-15). A variable *isMaximal*, initialized as true, is used to indicate whether R is a relative fair clique (line 9). Consider a vertex $u \in X$, the maximality checking is discussed in two aspects according to whether the attribute of u is the attribute with the least number of vertices in R(line 13). In the case of $u.val = a_{min}$, adding u can increase a_{min} by 1 to obtain a larger relative fair clique. Therefore, R is not an answer because it does not satisfy maximality, i.e., the condition (3) in Definition 3. On the other hand, that is, $u.val \neq$ a_{min} , the DeepRFCAlter procedure identifies whether the number of vertices in R with the attribute u.val is up to a_{max} . If no, adding u into R still satisfies the definition of a relative fair clique, thus *R* is not an answer due to the violation of the maximality. Once there is a vertex u that can make R break the maximality, we set the variable *isMaximal* to false. After checking all the vertices in the set X, the DeepRFCAlter adds *R* into the answer set *Res* if *isMaximal* equals true.

EXPERIMENTS 6

6.1 Experimental Setup

We implement WFCEnum (Algorithm 3) for weak fair clique enumeration. For strong fair clique enumeration, we implement SFCEnum (Algorithm 5) equipped with 1) the pruning technique FairnessCore (Algorithm 4) and the ordering

TABLE 1	
Datasets	

Dataset	n = V	m = E	d_{\max}	Description
Slashdot	82,169	504,230	2,252	Social network
Themarker	· 69,414	1,644,843	8,930	Social network
Aminer	423,469	1,231,112	712	Collaboration network
WikiTalk	2,394,385	5,021,410	100,029	Communication
				network
Flixster	2,523,387	7,918,801	1,474	Social network
Pokec	1,632,803	22,301,964	14,854	Social network

FairOD for the 2D case; and 2) the pruning technique ColorfulCore and the heuristic ordering HeurOD calculated by Algorithm 7 for the high-dimensional case. For relative fair clique enumeration, we implement RFCRefineEnum (Algorithm 9) and RFCAlterEnum (Algorithm 10) equipped with the pruning techniques EnhancedCore and ColorfulCore for 2D and high-dimensional cases. Since there is no existing algorithm that can be directly used to enumerate fairnessaware cliques, we implement three baseline algorithms, called BaseWeak, BaseStrong and BaseRelative. For the weak (relative) fair clique enumeration, BaseWeak (BaseRelative) first finds all maximal cliques using the state-of-the-art Bron-Kerbosch algorithm with pivoting technique [29], [30], and then filters them based on attribute constraint to identify weak (relative) fair cliques. For the strong fair clique enumeration, **BaseStrong** enumerates all cliques with size larger than $k \times A_n$, and then selects the strong fair cliques among them based on the attribute and maximality constraints. In addition, we also introduce two different basic orderings for our fairness-aware clique enumeration algorithms. The first ordering, called BfsOD, is obtained by performing breadth-first search (BFS) to explore the graph (i.e., the BFS visiting ordering of vertices); and the second ordering, called VidOD, is obtained by sorting the vertices based on the vertices' IDs. We compare the BaseWeak (BaseStrong) with the WFCEnum (SFCEnum) algorithms equipped with different orderings, i.e., BfsOD, VidOD and our proposed orderings. All algorithms are implemented in C++. We conduct all experiments on a PC with a 2.10GHz Inter Xeon CPU and 256GB memory. We set the time limit for all algorithms to 3 hours, and use the symbol "INF" to denote that the algorithm cannot terminate within 3 hours.

Datasets. We make use of six real-world graphs to evaluate the efficiency of the proposed algorithms. Table 1 summarizes the statistics of the datasets in our experiments. Aminer is a collaboration network, WikiTalk is a communication network and the others are social networks. The dataset Aminer is an attributed graph with 2-dimensional attribute to indicate the gender of a scholar, which can be downloaded from https://github.com/SotirisTsioutsiouliklis/ FairLaR/blob/master/Datasets/. And the other datasets are non-attributed graphs and can be downloaded from networkrepository.com/ and snap.stanford.edu. For these non-attributed graphs, we randomly assign an attribute to each vertex with roughly equal probability (i.e., 1/An) to generate attributed graphs to evaluate the efficiency of all algorithms.

Parameters. There are two parameters in our weak fair clique enumeration and strong fair clique enumeration algorithms: k and $d = A_n$. The parameter k is the threshold for the graph size. Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.

fair cliques and d is the number of attribute values (i.e., the attribute dimension). For the relative fair clique search algorithms, there is an extra parameter δ which is the maximum difference in the number of vertices of the attribute in addition to k and d. Since different datasets have various scales, the parameter k is set within different integers. For Aminer, k is chosen from the interval [9, 13] with a default value of k = 11. For Themarker, k is selected from the interval [7, 11] with a default value of k = 4. For Pokec, k is chosen from the interval [9,13] with a default value of k = 11. For the other datasets, k is chosen from the interval [9,13] with a default value k = 5. The parameter d is chosen from the interval [2, 6] with a default value of d = 2. For the parameter δ , if $\delta = 0$, the relative fair clique model degrades to the strong fair clique, and the proposed relative fair clique enumeration algorithms will invoke SFCEnum to find all relative/strong fair cliques. Thus, we choose δ from the interval [1,5] with a default value of $\delta = 3$ to evaluate the relative fair clique enumeration algorithms. When varying a parameter, the values of the other parameters are set to their default values. In particular, for Pokec, we set the default value of k to 5 for the experiments with varying d.

6.2 Efficiency Testing

Evaluation of the pruning techniques. For the 2D case (i.e., d = 2), both ColorfulCore and EnhancedCore can be used to reduce the graph size in WFCEnum, RFCRefineEnum and RFCAlterEnum algorithms. And ColorfulCore and FairnessCore can be used to reduce the graph size in the SFCEnum algorithm. In this experiment, we evaluate these pruning techniques by comparing the number of remaining vertices after pruning with varying k. The results are depicted in Figs. 3a, 3b, 3c, 3d, and 3e.

As can be seen from Fig. 3, in WFCEnum, RFCRefineEnum and RFCAlterEnum, both ColorfulCore and EnhancedCore can significantly reduce the number of vertices compared to the original graph as expected. Moreover, the number of remaining vertices decreases as k increases. For example, in Slashdot with k = 9, ColorfulCore reduces the number of vertices from 82,169 to 3,985; and EnhancedCore further reduces the number of vertices to 1,330. In general, EnhancedCore consistently outperforms ColorfulCore in terms of the pruning performance, especially for relatively small k values. When k goes larger, the pruning effect of ColorfulCore is slightly worse than that of EnhancedCore. This is because ColorfulCore can also prune a large number of vertices for a large k; for the SFCEnum algorithm, we can find that FairnessCore substantially reduces the number of vertices compared to ColorfulCore and the original graph. For instance, in Flixster with k = 9, the number of remaining vertices after applying ColorfulCore and FairnessCore is 15,258 and 10,602 respectively, while there are 2,523,387 nodes in the original graph. Generally, the pruning performance of FairnessCore is better than that of ColorfulCore with all parameter settings, especially for relatively small k values. For a larger *k*, the pruning effect of ColorfulCore is slightly worse than that of FairnessCore. This is because FairnessCore first invokes ColorfulCore to prune unpromising vertices. Since ColorfulCore is already able to prune a large number of vertices when k is large, FairnessCore cannot further prune too many vertices after invoking ColorfulCore. These results confirm that our pruning techniques are indeed very effective in reducing



Fig. 3. The number of remaining vertices after performing ColorfulCore, EnhancedCore and FairnessCore.

Note that for the high-dimensional case (i.e., $d \ge 3$), only the ColorfulCore algorithm can be used to prune the unpromising vertices in WFCEnum, SFCEnum, RFCRefineEnum and RFCAlterEnum algorithms. Therefore, we further study how the dimension d affects the pruning performance of ColorfulCore. Figs. 3f, 3g, 3h, 3i, and 3j show the number of remaining vertices after invoking ColorfulCore with varying d. As can be seen, ColorfulCore can substantially reduce the number of vertices with different d values overall datasets, which is consistent with our previous findings. In general, the number of remaining vertices decreases as d increases. This is because with a larger d, the constraints of ColorfulCore become stricter, thus more vertices can be pruned. These results further confirm the effectiveness of the proposed pruning techniques.

Evaluation of WFCEnum. Here we compare the BaseWeak and the WFCEnum algorithms equipped with BfsOD, VidOD and ColorOD by varying k and d. The results are depicted in Fig. 4. As can be seen, BaseWeak can only output the results on Slashdot and Pokec and cannot terminate within the time limit on the other datasets. Our WFCEnum algorithm, however, can work well on most datasets. The running time of BaseWeak is insensitive w.r.t. k and d, but the runtime of our WFCEnum algorithm decreases as k or d increases as expected. Moreover, we can see that the runtime of WFCEnum is several orders of magnitude lower than that of **BaseWeak** for a large k or d. For example, on Slashdot with k = 11, WFCEnum takes 268 seconds to enumerate all weak fair cliques, while BaseWeak consumes 10,665 seconds. This is because BaseWeak needs to enumerate all maximal cliques, which is the main bottleneck of the algorithm. For a large *k*, WFCEnum can prune many vertices by the colorful *k*-core based pruning technique and the search space can also be reduced during the backtracking procedure. For a large *d*, the number of weak fair cliques decreases with an increasing *d*, thus reducing time overheads. These results confirm that the proposed WFCEnum algorithm is much more efficient than BaseWeak to find all weak fair cliques on large graphs.

In addition, we can also see that WFCEnum with ColorOD is much faster than WFCEnum with BfsOD and VidOD on almost all of the datasets. For instance, when k = 11, WFCEnum with ColorOD consumes 4 seconds to output all results on Flixster, while WFCEnum with BfsOD and VidOD takes 25 and 633 seconds, respectively. On the Themarker dataset, when k = 7, the running time of WFCEnum with ColorOD is 5,550 seconds, while the two baseline algorithms cannot finish within 3 hours. These results indicate that the proposed algorithm is very efficient to enumerate all weak fair cliques in large real-life graphs. Also, the results confirm the efficiency of the proposed ordering technique ColorOD.

Evaluation of SFCEnum. We evaluate the runtime of SFCEnum with varying k and d. Since the proposed FairOD is tailored for d = 2, we only evaluate SFCEnum with FairOD by varying k. The experimental results of SFCEnum are illustrated in Fig. 5. In general, the runtime of SFCEnum decreases as k or d increases. This is because for a larger k or d, there are fewer cliques satisfying the definition of strong fair clique, thus the runtime for enumerating all strong fair cliques decreases. Additionally, we can see that the



Fig. 4. Running time of the BaseWeak algorithm and WFCEnum algorithms with different orderings. Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.



Fig. 5. Running time of the BaseStrong algorithm and SFCEnum algorithms with different orderings.

SFCEnum algorithms with FairOD and HeurOD are faster than those with BfsOD and VidOD on almost all of the datasets. For example, for k = 8 on Themarker, the SFCEnum algorithms equipped with FairOD and HeurOD consume 2,686 seconds and 2,789 seconds respectively, while the SFCEnum algorithms with BfsOD and VidOD take 4,225 and 4,834 seconds to output all strong fair cliques respectively. These results confirm the efficiency of the proposed ordering techniques.

Additionally, by comparing BaseStrong and SFCEnum, we find that the running time of BaseStrong on the datasets except for Pokec exceeds the time limit, thus we do not show them in Fig. 5. The proposed SFCEnum algorithms, however, work well on most datasets. As aforementioned, to enumerate strong fair cliques, BaseStrong needs to find all cliques with size larger than $k \times A_n$ first. The number of such cliques is often extremely large, thus the running time of BaseStrong is significantly higher than SFCEnum.

Evaluation of RFCRefineEnum and RFCAlterEnum. Here, we evaluate the proposed relative fair clique enumeration algorithms, i.e., RFCRefineEnum and RFCAlterEnum, with varying k, d and δ . The experimental results are illustrated in Fig. 6. Obviously, the runtime of RFCAlterEnum is significantly lower than that of RFCRefineEnum within all parameter settings. In

general, the runtime of RFCRefineEnum and RFCAlterEnum decreases as k or d increases as expected. This is because for a larger k or d, fewer cliques satisfying the definition of a relative fair clique, thus decreasing the runtime for enumerating all relative fair cliques. These results are consistent with the previous findings. For the parameter δ , the runtime of RFCRefineEnum and RFCAlterEnum changes very smoothly with increasing δ . This is because the RFCRefineEnum algorithm performs WFCEnum to find all weak fair cliques and then enumerates relative fair cliques contained in them. Finding weak fair cliques occupies most of the runtime of all **RFCRefineEnum**, which is independent of the parameter δ ; and the RFCAlterEnum algorithm adopts attribute-alternatively-selection strategy to enumerate relative fair cliques, thus the runtime is insensitive to the difference threshold δ . Note that for Slashdot, the RFCAlterEnum achieves the maximum runtime at $\delta = 1$. In this case, the attribute with the minimum number of nodes is a_{ϕ} and the numbers of nodes with attributes $a_0, a_1, ..., a_{\phi-1}$ reach the maximum. Thus, RFCAlterEnum needs to update candidate sets to be empty for $a_0, a_1, ..., a_{\phi-1}$, which causes a little bit of increase in running time.

From Fig. 6, we can also see that the RFCAlterEnum algorithm is faster than RFCRefineEnum within all parameter settings over all datasets. For example, in the case of k = 9, the



Fig. 6. Running time of the RFCRefineEnum and RFCAlterEnum algorithms. Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.



Fig. 7. The number of weak fair cliques, strong fair cliques and relative fair cliques on various datasets.

runtime of RFCRefineEnum and RFCAlterEnum algorithms on WikiTalk is 7,718 seconds and 2,683 seconds, respectively. Clearly, the former is around 2.877 times slower than the latter. While for d = 3 on Slashdot, the RFCRefineEnum algorithm consumes 9,628 seconds, while the RFCAlterEnum takes 2,461 seconds to output all relative fair cliques which is roughly 3.912 times faster than that of RFCRefineEnum. For $\delta = 3$ on Flixster, the RFCRefineEnum and RFCAlterEnum algorithms take 6,492 seconds and 2,849 seconds to output the results, respectively. The runtime of RFCAlterEnum is roughly 2.279 times faster than that of RFCRefineEnum. In addition, we also evaluate the proposed algorithms by comparing them with the BaseRelative algorithm. The running time of BaseRelative on all datasets exceeds the time limit, thus we do not show them in Fig. 6. From Fig. 6, the proposed RFCRefineEnum and RFCAlterEnum algorithms work well on most datasets. To search relative fair cliques, the BaseRelative algorithm needs to find all maximal cliques first, thus the running time is significantly higher than our proposed algorithms. These results confirm the efficiency of the proposed RFCRefineEnum and RFCAlterEnum algorithms.



Fig. 8. Scalability of WFCEnum, SFCEnum, RFCRefineEnum and RFCAlterEnum algorithms.

The Number of Fairness-Aware Cliques. Figs. 7a, 7b, 7c, 7d, and 7e show the numbers of weak fair cliques, strong fair cliques and relative fair cliques with different k. Clearly, there are significant numbers of fair cliques in each dataset. In general, the number of strong fair cliques is larger than that of relative fair cliques, and the number of relative fair cliques is larger than that of weak fair cliques. This finding is consistent with our analysis in Section 2, since a weak fair clique often contains a set of relative fair cliques, and a relative fair clique includes a set of strong fair cliques. Additionally, we can see that the number of fair cliques decreases when k increases. This is because with a larger k, both the fairness and clique constraints become stricter, thus resulting in fewer fair cliques. Similar results can also be observed when varying d from Figs. 7f, 7g, 7h, 7i, and 7j. Figs. 7k, 7l, 7m, 7n, and 7o also illustrate the numbers of relative fair cliques with varying δ . Obviously, there are also significant numbers of relative fair cliques in each dataset for different δ . These results confirm that our relative fair clique model indeed achieves a great compromise between the weak fair clique and strong fair clique models by introducing the difference threshold δ , which is consistent with our analysis in Section 2.

Scalability Testing. To evaluate the scalability of the proposed algorithms, we generate four subgraphs for each dataset by randomly picking 20%-80% of the edges, and evaluate the runtime of all the proposed algorithms. Fig. 8 illustrates the results on Flixster. The results on the other datasets are consistent. In Fig. 8a, the runtime of WFCEnum with BfsOD and VidOD increases sharply as the graph size increases, while for ColorOD, it increases smoothly with varying m. Moreover, the ColorOD ordering performs much better than the other orderings with all parameter settings, which is consistent with our previous findings. Analogously, when varying m, the runtime of SFCEnum with BfsOD and VidOD increases sharply with respect to the graph size in Fig. 8b. However, for SFCEnum with FairOD and HeurOD, the runtime increases smoothly with m increases. From Fig. 8c, we can also see that for relative fair clique enumeration algorithms, the runtime of RFCAlterEnum

increases very smoothly with increasing m, while the runtime Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.



Fig. 9. Memory overhead.

RFCRefineEnum of increases more sharply. Again, RFCAlterEnum is significantly faster than RFCRefineEnum, which is consistent with our previous findings. These results demonstrate the high scalability of the proposed algorithms.

Memory Overhead. Fig. 9 shows the memory overheads of WFCEnum, SFCEnum, RFCRefineEnum and RFCAlterEnum algorithms on all datasets except for Aminer. Note that the memory costs of different algorithms do not include the size of the graph. From Figs. 9a and 9b, we can see that the memory usages of WFCEnum and SFCEnum with different orderings are always smaller than the graph size. This is because both the WFCEnum and SFCEnum algorithms follow a depth-first manner, thus the space overhead is linear. Additionally, the memory overheads of WFCEnum and SFCEnum are robust with respect to different orderings. This is because the space usage in the enumeration procedure is mainly dominated by the depth of the enumeration tree. Since the tree depth is determined by the clique size, the space overhead is insensitive to different orderings. As can be seen from Fig. 9c, the memory occupancy of RFCRefineEnum and RFCAlterEnum are also significantly smaller than the graph size since they also enumerate relative fair cliques in a depth-first manner like WFCEnum and SFCEnum.

Efficiency Testing on Aminer. Here we use a real-life attributed graph Aminer to evaluate the proposed fair clique models and enumeration algorithms. As aforementioned, the attribute value of vertices in Aminer is 2dimensional, thus we only vary k and δ to conduct experiments. The number of remaining vertices after performing ColorfulCore, EnhancedCore and FairnessCore with varying k is depicted in Fig. 10a. Clearly, the ColorfulCore, EnhancedCore and FairnessCore can significantly reduce the number of vertices compared to the original graph (i.e., 423,469 vertices) as expected. Moreover, the number of remaining vertices decreases as k increases. These results further confirm the effectiveness of our pruning techniques. Figs. 10b and 10c show the running time of weak fair clique enumeration algorithms and strong fair clique enumeration algorithms with different values of k. As can be seen, BaseWeak and BaseStrong cannot terminate within the time limit under most parameter settings, while the proposed WFCEnum and SFCEnum algorithms can work well. Again, the runtime of WFCEnum and SFCEnum with different orderings decreases as k increases, which is consistent with our previous findings. For relative fair clique Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 02:48:22 UTC from IEEE Xplore. Restrictions apply.



(a) The number of remaining vertices by different pruning techniques (vary k)





(b) Comparision of BaseWeak and WFCEnum algorithms (vary k)







and RFCAlterEnum algorithms (vary δ)

(d) Comparision of RFCRefineEnum and RFCAlterEnum algorithms (vary k)



(vary k) $(\text{vary } \delta)$



enumeration algorithms, the experimental results of varying k and δ are illustrated in Figs. 10d and 10e Clearly, RFCAlterEnum is faster than RFCRefineEnum within all parameter settings. As k increases, both the runtime of RFCRefineEnum and RFCAlterEnum decreases, while as δ increases, the runtime of the two algorithms changes very smoothly, which is consistent. We also depict the numbers of fair cliques with different k and δ in Figs. 10f and 10g, and the results confirm that there are indeed many numbers of weak fair cliques, strong fair cliques, and relative fair cliques in Aminer. In addition, we show the memory overheads of fair clique enumeration algorithms in Table 2. As can be seen, the memory usages of WFCEnum, RFCRefineEnum and RFCAlterEnum are smaller than the graph size, and SFCEnum uses less than 500MB memory which is acceptable for a modern computer. Moreover, the memory overheads of WFCEnum and SFCEnum are robust with respect to different orderings, which is consistent with our previous findings. These results demonstrate the effectiveness and efficiency of our proposed fair clique models and algorithms.

6.3 Case Study

We conduct a case study on a collaboration network DBLP to evaluate the effectiveness of our algorithms. The DBLP

Algorithm	Ordering	Memory overhead (MB)
Graph size	-	12.624
WFCEnum	ColorOD BfsOD VidOD	4.856 4.856 4.856
SFCEnum	FairOD HeurOD BfsOD VidOD	470.962 474.193 474.193 474.193
RFCRefineEnum RFCAlterEnum	-	4.856 8.077

TABLE 2 Memory Overhead of Fair Clique Enumeration Algorithms on Aminer

dataset is downloaded from dblp.uni-trier.de/xml/. We extract a subgraph DBCS from DBLP which contains the authors who had published at least one paper in the database (*DB*), data mining (*DM*), and artificial intelligence (*AI*) related conferences. The DBCS subgraph contains 52,106 vertices (authors) and 341,382 undirected edges. The attribute *A* represents the author's main research area with $A_{val} = \{DB, DM, AI\}$. Each vertex has one attribute value selected from the set A_{val} . We set the attribute value for each vertex based on the maximum number of papers that the author published in the related conferences. For example, if an author has published 20 papers in *DB* related conferences and 5 papers in *DM* related conferences, we choose *DB* as the author's attribute value.

We perform the WFCEnum, SFCEnum and RFCRefine *Enum* (RFCAlterEnum) algorithms to find all weak fair cliques, strong fair cliques and relative fair cliques on DBCS with k = 2 and $\delta = 2, 3$. All algorithms apply ColorfulCore to prune the unpromising vertices. The remaining graph after pruning by ColorfulCore only has 61 vertices and 516 edges. Fig. 11a shows a weak fair clique with size 10, which involves 6 authors of *DB*, 2 authors of *DM* and 2 authors of *AI*. We use different colors to represent the main research area of these authors, namely, pink = DB, green = DM, and blue = AI. Clearly, the number of vertices with different attribute values is no less than k = 2. These results indicate that WFCEnum can find fair communities with diverse research areas. However, in Fig. 11a, the weak fair



Fig. 11. Results of WFCEnum and SFCEnum on DBCS with $A_{val} = \{DB, DM, AI\}$.



Fig. 12. Results of RFCRefineEnum/RFCAlterEnum on DBCS with $A_{val} = \{DB, DM, AI\}.$

clique is imbalanced (w.r.t. different attributes) due to the high percentage of authors with *DB*. Figs. 11b and 11c show two strong fair cliques which are also subgraphs of the clique in Fig. 11a. This is consistent with the finding that a strong fair clique must be contained in a weak fair clique. As expected, the number of authors with different attribute values is exactly equal to 2, thus it can avoid the *attribute imbalance* problem in the weak fair clique.

We also depict four relative fair cliques in Fig. 12, which are related to the weak fair clique and strong fair cliques in Fig. 11. Figs. 12a and 12b and Figs. 12c and 12d are the cliques for $\delta =$ 2 and $\delta = 3$, respectively. As can be seen from Figs. 12a and 12b, the number of vertices with different attribute values is no less than k = 2 and the maximum difference in the number of vertices of those attributes is $2 \le \delta = 2$. Moreover, these two relative cliques are also subgraphs of the clique in Fig. 11a and they both contains the strong fair cliques shown in Figs. 11b and 11c. Similar results can also be found in Figs. 12c and 12d. By comparing the cliques with $\delta = 2$ and $\delta = 3$, we can find that the difference threshold δ does measure the balance between the attributes in a relative fair clique. A larger δ leads to finding a clique in which the number of nodes of each attribute varies greatly, and thus the result is closer to a weak fair clique. While for a smaller δ , the enumerated relative fair cliques are closer to the model of strong fair clique. This finding reveals that our relative fair clique model is a good compromise between the weak fair clique and the strong fair clique models as described in Section 2.

All the results demonstrate that the WFCEnum, SFCEnum and RFCAlterEnum/RFCRefineEnum algorithms can be used to find fair communities with diverse attributes; SFCEnum can further keep a balance over different attributes in the community; and RFCAlterEnum and RFCRefineEnum provide a more flexible way to find fair communities as a compromise by specifying the difference threshold δ . In addition, this case study also indicates that the fairness-aware cliques show the scholars of different research areas who cooperate with each other, and further reflect the closeness of different research areas. That is, the closer these areas are, the larger fair cliques will be. If no fair clique can be found, then it means that at least one research area has no obvious connection to others. The fairness-aware clique models aim to find balance among different attributes, which are suitable to be used in cross-cutting areas.

6.4 Discussions

As shown in our experiments, seeking a suitable k for our fair clique model is important for practical applications. Here we introduce a heuristic method to find an appropriate k. Since the sizes of fair cliques are clearly no larger than the maximum clique size of the graph, we can first compute the maximum clique size of a graph by using the state-ofthe-art maximum clique search algorithms [31], [32]. Suppose that the size of a maximum clique is C_{max} . Then, the parameter k in our fair clique models satisfies $k \leq \lfloor \frac{C_{max}}{A_{ra}} \rfloor$. Note that when the maximum clique size is hard to compute for some instances, an alternative solution is to compute an approximation of C_{max} by using a linear-time greedy algorithm [33]. Therefore, for a particular application, we can use a binary search method to find an appropriate k from the interval $\left[1, \left\lfloor \frac{C_{max}}{A_n} \right\rfloor\right]$ by invoking the proposed algorithms to compute the fairness-aware cliques.

7 RELATED WORK

Attributed Graph Mining. Our work is related to attributed graph mining which has attracted much attention in data mining due to the diverse applications [9], [10], [11], [12], [13], [14]. For example, Li et al. [9] proposed an embeddingbased model to discover communities in attributed graphs. Tong et al. [10] studied the problem of finding subgraphs for given query patterns in attributed graphs. Fang et al. [11] investigated the attributed community search problem and developed an index structure, called CL-tree, to efficiently support attributed community search. Khan et al. [12] proposed an algorithm to mine subgraphs such that the vertices in the subgraph are closely connected and each vertex contains as many query keywords as possible. Pizzuti et al. [13] introduced a community mining algorithm for attributed graphs that considers both node similarity and structural connectivity. In this paper, we study a problem of mining fair communities (fair cohesive subgraph) in attributed graphs. To the best of our knowledge, our work is the first to study the fair community search problem in attributed networks.

Fairness-Aware Data Mining. Our work is related to fairness-aware data mining which has been recognized as an important issue in data mining and machine learning. To measure fairness, many concepts have been proposed in the literature [17]. Zehlike et al. [20] proposed a method to generate a ranking with a guaranteed group fairness, which can ensure the proportion of protected elements in the rank is no less than a given threshold. Serbos et al. [21] investigated a problem of fairness in package-to-group recommendation, and proposed a greedy algorithm to find approximate solutions. Beutel et al. [22] also studied fairness in recommendation systems and presented a set of metrics to evaluate algorithmic fairness. Another line of research on fairness was studied in classification algorithms. Some notable work includes demographic parity [19] and equality of opportunity [18]. For instance, Hardt et al. [18] proposed a framework that can optimally adjust any learned predictor to reduce bias. Compared to the existing studies, our definition of fairness which requires the equality of different attribute values in a group is different from those in the

Cohesive Subgraph Mining. Our work is also related to cohesive subgraph mining. Clique is an important cohesive subgraph model and there are numerous studies that focus on clique mining. Finding maximum cliques, aiming to discover the cliques with the largest size, has attracted much attention. The algorithms for maximum clique search are mainly based on the branch-and-bound framework [34], [35]. Ostergard et al. [34] presented a branch-and-bound algorithm with the vertex order taken from a coloring of the vertices. Konc et al. [35] proposed an approximate coloring algorithm and used it to provide bounds of the size of the maximum clique. Tomita et al. proposed a series of maximum clique algorithms, called MCQ [36], MCR [37], MCS [38] and MCT [31], [39], based on the coloring technique. All these algorithms either use the coloring technique to obtain an upper bound of the maximum clique or apply the coloring heuristics to design a branching strategy. Moreover, all these algorithms are mainly tailored to non-attributed graphs. Different from these works, we use the coloring technique to develop a k-core based graph reduction approach; and our work aims to find fairness-aware cliques in attributed graphs.

Another research problem of clique mining is to enumerate maximal cliques [29], [30], [40], [41]. The well-known algorithm for enumerating all maximal cliques is the classic Bron-Kerbosch (BK) algorithm [29]. Tomita et al. [30] proposed an algorithm, using a greedy pivoting technique, to find all maximal cliques. Eppsten et al. [40] further improved the BK algorithm based on a heuristic degeneracy ordering. Yuan et al. [41], [42] studies the diversified top-k clique search problem which aims to find top-k maximal cliques that can cover most number of nodes in the graph. In addition, some relaxed definitions of clique were also proposed, such as n-clique [43], n-clan, n-club [44], k-plex [45], [46], quasi-clique [47], [48], k-core [49], [50], [51], and so on [52]. However, the solutions mentioned above are not tailored for attributed graphs, and thus cannot be directly used to solve our problems.

The clique model has also been extended to various networks, such as signed graphs and bipartite graphs, and has been widely investigated in [53], [54], [55], [56], [57], [58]. Li et al. [53], [57] introduced a signed clique model by constraining the number of positive and negative edges, and studied the problems of finding all and top-r maximal signed cliques in signed networks. Chen et al. [54] proposed a balanced clique model based on the structural balance theory, and developed a new algorithm with two optimization strategies to compute all the maximal balanced cliques in signed networks. However, the techniques in the above studies are tailored for signed networks, and thus cannot be directly used to solve fair clique enumeration problems for graphs where the vertices have attributes. For bipartite graphs, Lyu et al. [53] defined a size-constrained maximum edge biclique and studied the problem of finding a maximum biclique for bipartite graphs with billion scale. Chen et al. [56] investigated the maximal bicliques enumeration problem and proposed a novel unilateral order and a batchpivots technique. Recently, Chen et al. [56] presented a balanced biclique model that requires the number of vertices on two sides of a biclique to be the same, and developed exact algorithms to find a maximum balanced biclique. The

which are clearly different from our work and thus the related solutions cannot be used to address the fair clique enumeration problems. In this paper, we develop novel algorithms to compute maximal fair cliques in attributed graphs with several non-trivial pruning techniques.

CONCLUSION 8

In this paper, we study a problem of enumerating fairnessaware cliques in attributed graphs. To this end, we propose a weak fair clique model, a strong fair clique model and a relative fair clique model. To enumerate all weak fair cliques, we first present a novel colorful k-core based pruning technique to prune unpromising vertices. And then we develop a backtracking algorithm with a carefully-designed ordering technique to enumerate all weak fair cliques in the pruned graph. To enumerate all strong fair cliques, we propose a new fairness k-core based pruning algorithm for the 2D case, and then develop a backtracking algorithm with a fairness k-core based ordering technique to enumerate all strong fair cliques. We also present a strong fair clique enumeration algorithm with a heuristic ordering for handling high-dimensional cases. To enumerate all relative fair cliques, we present two efficient algorithms based on a weak fair clique refinement strategy and an attribute-alternatively-selection strategy, respectively. We also design an enhanced colorful k-core based pruning technique for 2D attributes, which can also be applied to reduce the graph for weak fair clique enumeration. Extensive experiments are conducted using four large real-life graphs, and the results demonstrate the efficiency and effectiveness of the proposed algorithms.

There are several future directions that are deserved further investigation. First, the proposed models are based on the concept of clique which may be strict for some real-life applications. A promising direction is to relax the clique model used in our definitions, and apply other models (e.g., k-truss and k-edge connected subgraph) to define the fairness-aware cohesive subgraphs, which clearly requires new algorithms with different techniques. Second, the proposed pruning technique is mainly based on the colorful *k*-core. An interesting question is that can we develop a colorful k-truss based pruning technique? Since *k*-truss is often much denser than *k*-core, such a pruning technique may be more powerful than our colorful k-core based technique. Finally, it is also interesting to develop more efficient branching and ordering techniques to further speed up the backtracking enumeration procedure. We leave these problems as interesting future works.

REFERENCES

- [1] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," Proc. VLDB Endowment, vol. 8, no. 5, pp. 509–520, ž015.
- S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, [2] "Community detection in social media - performance and application considerations," Data Mining Knowl. Discov., vol. 24, no. 3, pp. 515–554, 2012.
- X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-[3] truss community in large and dynamic graphs," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2014, pp. 1311–1322. T. Friedrich and A. Krohmer, "Cliques in hyperbolic random
- [4] graphs," in Proc. IEEE Conf. Comput. Commun., 2015, pp. 1544-1552.

- R. Li, Q. Dai, G. Wang, Z. Ming, L. Qin, and J. X. Yu, "Improved [5] algorithms for maximal clique search in uncertain networks," in *Proc. Int. Conf. Data Eng.*, 2019, pp. 1178–1189.[6] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, "Index-based
- densest clique percolation community search in networks," IEEE Trans. Knowl. Data Eng., vol. 30, no. 5, pp. 922-935, May 2018.
- H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein, "Predicting [7] interactions in protein networks by completing defective cliques, Bioinf., vol. 22, no. 7, pp. 823-829, 2006.
- V. Boginski, S. Butenko, and P. M. Pardalos, "Mining market data: [8] A network approach," Comput. Operations Res., vol. 33, no. 11,
- pp. 3171–3184, 2006. Y. Li, C. Sha, X. Huang, and Y. Zhang, "Community detection in [9] attributed graphs: An embedding approach," in Proc. Conf. Assoc. Advance. Artif. Intell., 2018, pp. 338-345.
- [10] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, "Fast besteffort pattern matching in large attributed graphs," in Proc. Int. *Conf. Knowl. Discov. Data Mining*, 2007, pp. 737–746. [11] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search
- for large attributed graphs," Proc. VLDB Endowment, vol. 9, no. 12, pp. 1233-1244, 2016.
- [12] A. Khan et al., "Compact group discovery in attributed graphs and social networks," *Inf. Process. Manage.*, vol. 57, no. 2, 2020, Art. no. 102054.
- [13] C. Pizzuti and A. Socievole, "A genetic algorithm for community detection in attributed graphs," in Proc. Int. Conf. Appl. Evol. Com*put.*, 2018, pp. 159–170. [14] Y. Wu, Z. Zhong, W. Xiong, and N. Jing, "Graph summarization
- for attributed graphs," in Proc. Int. Conf. Inf. Sci., Electron. Elect. Eng., 2014, pp. 503-507.
- [15] J. Yang, J. McAuley, and J. Leskovec, "Community detection in networks with node attributes," in Proc. Int. Conf. Data Mining, 2013, pp. 1151-1156.
- [16] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *Proc. ACM SIGMOD* Int. Conf. Manage. Data, 2012, pp. 505-516.
- [17] S. Verma and J. Rubin, "Fairness definitions explained," in Proc. IEEE/ACM Int. Workshop Softw. Fairness, 2018, pp. 1-7.
- [18] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in Proc. Int. Conf. Neural Inf. Process. Syst., 2016, pp. 3315-3323.
- [19] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. S. Zemel, "Fairness through awareness," in Proc. Innov. Theor. Comput. Sci. Conf., 2012, pp. 214-226.
- [20] M. Zehlike, F. Bonchi, C. Castillo, S. Hajian, M. Megahed, and R. Baeza-Yates, "Fa*ir: A fair top-k ranking algorithm," in Proc. Conf. Inf. Knowl. Manage., 2017, pp. 1569-1578.
- [21] D. Serbos, S. Qi, N. Mamoulis, E. Pitoura, and P. Tsaparas, "Fairness in package-to-group recommendations," in Proc. Int. World Wide Web Conf., 2017, pp. 371–379.
- [22] A. Beutel et al., "Fairness in recommendation ranking through pairwise comparisons," in Proc. Int. Conf. Knowl. Discov. Data Mining, 2019, pp. 2212-2220.
- [23] D. W. Matula, G. Marble, and J. D. Isaacson, "Graph coloring algorithms," in Graph Theory and Computing. New York, NY, USA: Academic Press, 1972, pp. 109-122.
- [24] T. R. Jensen and B. Toft, Graph Coloring Problems, vol. 39. Hoboken, NJ, USA: Wiley, 2011.
- [25] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," 2003, arXiv:cs/0310049.
- [26] D. W. Matula and L. L. Beck, "Smallest-last ordering and cluster-ing and graph coloring algorithms," J. ACM, vol. 30, no. 3, pp. 417–427, 1983.
- [27] J. Mitchem, "On various algorithms for estimating the chromatic number of a graph," *Comput. J.*, vol. 19, no. 2, pp. 182–183, 1976. [28] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson,
- "Ordering heuristics for parallel graph coloring," in Proc. ACM Symp. Parallelism Algorithms Archit., 2014, pp. 166–177.
- [29] Č. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," Commun. ACM, vol. 16, no. 9, pp. 575-577, 1973.
- [30] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," Theor. Comput. Sci., vol. 363, no. 1, pp. 28-42, 2006.
- [31] E. Tomita, "Efficient algorithms for finding maximum and maximal cliques and their applications," in Proc. Int. Workshop Algorithms Comput., 2017, pp. 3–15.

- [32] L. Chang, "Efficient maximum clique computation over large sparse graphs," in Proc. Int. Conf. Knowl. Discov. Data Mining, 2019, pp. 529-538
- [33] R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin, "Parallel maximum clique algorithms with applications to network analysis," SIAM J. Sci. Comput., vol. 37, no. 5, pp. C589-C616, 2015.
- [34] P. R. Östergård, "A fast algorithm for the maximum clique problem," Discrete Appl. Math., vol. 120, no. 1-3, pp. 197-207, 2002.
- [35] J. Konc and D. Janezic, "An improved branch and bound algorithm for the maximum clique problem," Proteins, vol. 4, no. 5, pp. 590–596, 2007.
- [36] E. Tomita and T. Seki, "An efficient branch-and-bound algorithm for finding a maximum clique," in Proc. Int. Conf. Discrete Math. Theor. Comput. Sci., 2003, pp. 278-289.
- [37] E. Tomita and T. Kameda, "An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments," J. Glob. Optim., 2009, Art. no. 311.
- [38] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki, "A simple and faster branch-and-bound algorithm for finding a maximum clique," in Proc. Int. Workshop Algorithms Comput., 2010, pp. 191-203.
- [39] E. Tomita, K. Yoshida, T. Hatta, A. Nagao, H. Ito, and M. Wakatsuki, "A much faster branch-and-bound algorithm for finding a maximum clique," in Proc. Int. Workshop Front. Algorithmics, 2016, pp. 215-226.
- [40] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," in *Proc. Int. Symp. Exp. Algorithms*, 2011, pp. 364–375
- [41] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified topk clique search," in Proc. Int. Conf. Data Eng., 2015, pp. 387-398.
- [42] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified topk clique search," VLDB J., vol. 25, no. 2, pp. 171–196, 2016.
- [43] R. D. Alba, "A graph-theoretic definition of a sociometric clique," J. Math. Sociol., vol. 3, no. 1, pp. 113–126, 1973.
- [44] R. J. Mokken et al., "Cliques, clubs and clans," Qual. Quantity, vol. 13, no. 2, pp. 161-173, 1979.
- [45] S. B. Seidman and B. L. Foster, "A graph-theoretic generalization of the clique concept," J. Math. Sociol., vol. 6, no. 1, pp. 139-154, 1978.
- [46] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum k-plex problem," Operations Res., vol. 59, no. 1, pp. 133-142, 2011.
- [47] J. Pardalos and M. Resende, "On maximum clique problems in very large graphs," *DIMACS Ser.*, vol. 50, pp. 119–130, 1999. J. Abello, M. G. Resende, and S. Sudarsky, "Massive quasi-clique
- [48] detection," in Proc. Latin Amer. Symp. Theor. Inform., 2002, pp. 598-612.
- [49] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, "K-core organization of complex networks," Phys. Rev. Lett., vol. 96, no. 4, 2006, Art. no. 040601.
- [50] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single PC," VLDB J., vol. 9, no. 1, pp. 13-23, 2015.
- [51] A. Montresor, F. De Pellegrini, and D. Miorandi, "Distributed kcore decomposition," IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 2, pp. 288–300, Feb. 2013.
- [52] S. P. Borgatti, M. G. Everett, and P. R. Shirey, "LS sets, lambda sets and other cohesive subsets," Social Netw., vol. 12, no. 4, pp. 337-357, 1990.
- [53] R. Li et al., "Efficient signed clique search in signed networks," in Proc. Int. Conf. Data Eng., 2018, pp. 245-256.
- [54] Z. Chen, L. Yuan, X. Lin, L. Qin, and J. Yang, "Efficient maximal balanced clique enumeration in signed networks," in Proc. Int. World Wide Web Conf., 2020, pp. 339-349.
- [55] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum biclique search at billion scale," Proc. VLDB Endowment, vol. 13, no. 9, pp. 1359–1372, 2020.
- [56] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2021, pp. 248–260.
- [57] R. Li et al., "Signed clique search in signed networks: Concepts and algorithms," IEEE Trans. Knowl. Data Eng., vol. 33, no. 2, pp. 710–727, Feb. 2021.
- L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient maximal bicli-[58] que enumeration for large sparse bipartite graphs," Proc. VLDB Endowment, vol. 15, no. 8, pp. 1559-1571, 2022.



Qi Zhang is currently working toward the PhD degree with the Beijing Institute of Technology, China. Her current research interests include social network analysis and data-driven graph mining.



Rong-Hua Li received the PhD degree from the Chinese University of Hong Kong in 2013. He is currently a professor with the Beijing Institute of Technology, Beijing, China. His research interests include graph data management and mining, social network analysis, graph computation systems, and graph-based machine learning.



Minjia Pan received the BS degree in computer science from Northeastern University, China in 2019. She is currently working toward the under graduation degree with the Beijing Institute of Technology, China. Her current research interests include social network analysis and graph mining.



Yongheng Dai received the PhD degree from The Chinese University of Hong Kong in 2011. Now he works as an R&D engineer in the areas of domain modeling, knowledge formalization, and knowledge-driven machine learning in the China Academy of Electronics and Information Technology (CAEIT). From 2011 to 2013, he worked on optical fiber communication and digital signal processing in Huawei. After that, he worked on OFDM-based visible light communication and OCC-based indoor positioning until 2017 in CAEIT. He has published

33 papers in international journals and conferences, and holds 10 pending patents. He is also the recipient of China postdoctoral science foundation grant 2015, Beijing Science and Technology Award 2008, and IEEE Photonics Society (HK chapter) Best Paper Award 2008.



Qun Tian received the master's degree from the Harbin Institute of Technology, is a senior engineer in artificial intelligence with Diankeyun Technologies Ltd, Beijing, China. His interests include complex network, graph neural network and knowledge inference.



Guoren Wang received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991 and 1996, respectively. Currently, he is a professor with the Department of Computer Science, Beijing Institute of Technology, Beijing, China. His research interests include XML data management, query processing and optimization, bioinformatics, high dimensional indexing, parallel database systems, and cloud data management. He has published more than 100 research papers.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.