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 paper, 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 and females are exactly the same. 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 paper, we focus on the problem of finding fairness-aware cliques in attributed graphs where each vertex in the graph has one attribute. We propose two new models to characterize the fairness of a clique, called weak fair clique and strong fair clique respectively. A weak fair clique is a maximal subgraph which 1) satisfies 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 [11], [15], [44]. 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., women, racial and other biases) [5], [36], [50]. Therefore, many different definitions of fairness, such as individual fairness, group fairness [44], and related algorithms were proposed to generate a fairness ranking. Some other studies focus on the fairness in classification models, such as demographic parity [11] and equality of opportunity [15]. 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 and females are exactly the same. 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 paper, we focus on the problem of finding fairness-aware cliques in attributed graphs where each vertex in the graph has one attribute. We propose two new models to characterize the fairness of a clique, called weak fair clique and strong fair clique respectively. A weak fair clique is a maximal subgraph which 1) satisfies 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 [11], [15], [44]. 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., women, racial and other biases) [5], [36], [50]. Therefore, many different definitions of fairness, such as individual fairness, group fairness [44], and related algorithms were proposed to generate a fairness ranking. Some other studies focus on the fairness in classification models, such as demographic parity [11] and equality of opportunity [15]. 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 and females are exactly the same. 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 paper, we focus on the problem of finding fairness-aware cliques in attributed graphs where each vertex in the graph has one attribute. We propose two new models to characterize the fairness of a clique, called weak fair clique and strong fair clique respectively. A weak fair clique is a maximal subgraph which 1) satisfies some attribute constraints. None of them takes into account the fairness of attributes in the community.
the colorful $k$-core. We show that both weak fair cliques and strong fair cliques must be contained in the colorful $k$-core, thus we can use it for pruning unpromising vertices in enumerating weak or strong fair cliques. Then, we propose a backtracking algorithm WFCEnum to enumerate 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 propose 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.

Extensive experiments. We conduct extensive experiments to evaluate the efficiency and effectiveness of our algorithms using four real-world networks. The results show that the colorful $k$-core based pruning technique is very powerful which can significantly prune the original graph. The results also show that the WFCEnum and SFCEnum algorithms are efficient in practice. Both of them 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 show that both WFCEnum and SFCEnum can find fair communities with different research areas, and SFCEnum can further keep balance of attribute values in the subgraph.

Reproducibility. The source code of this paper is released at Github: [https://github.com/honnameiko22/fairnessclique](https://github.com/honnameiko22/fairnessclique) for reproducibility purpose.

II. 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 $A_{val} = \{a_{i}|0 \leq i < A_{n}\}$. 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. Based on the concept of clique, we present two fairness-aware clique models as follows.

**Definition 1:** (Weak fair clique) Given an attributed graph $G$ and 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 $C' \subset C$ satisfying (1).

**Example 1:** Consider a graph $G = (V,E,A)$ with $A_{val} = \{a,b\}$ in Fig. 1(a). Suppose that $k = 3$. By Definition 1, we can see that the subgraph $C$ induced by the vertex set $\{v_1,v_2,v_3,v_4,v_5,v_6\}$ is a weak fair clique. This is because the number of vertices with attribute value $a$ in $C$ is 4 ($\geq k = 3$), and with attribute $b$ is 3 ($\geq k = 3$). Moreover, there does not exist a subgraph $C'$ that contains $C$ and also satisfies the condition (1) in Definition 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 the fairness for all attributes. Below, we propose a strong fair clique definition which strictly requires the subgraph has the same number of vertices for each attribute.

**Definition 2:** (Strong fair clique) Given an attributed graph $G$ and 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 $C' \subset C$ satisfying (1) and (2).

**Example 2:** Reconsider the attributed graph $G$ in Fig. 1(a). Again, we assume that $k = 3$. By definition, we can easily check that the subgraph induced by $\{v_1,v_2,v_3,v_4,v_5,v_6\}$ is a strong fair clique. Note that the subgraph induced by $\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$ is a weak fair clique, but it is not a strong fair clique, as it violates the condition (2) in Definition 2.

**Remark.** According to Definition 1 and Definition 2, the parameter $k$ in our fair clique models provides a lower bound on the size of a clique. There are at least $k \times A_v$ vertices in both a weak fair clique and a strong 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 model, 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 addition, another potential definition of fairness-aware clique is to consider the difference of the number of each attribute in the clique. Such a definition, however, has a limitation. If we only guarantee that the difference of the number of each attribute is below a given threshold, we may miss fairness in some cases. For example, suppose that we have three attributes A, B and C, and the given threshold is 5. Then, we may find a 5-clique that has 5 A vertices, 0 B vertices, and 0 C vertices which is clearly unfair for the attributes B and C. However, our definitions of fair cliques can guarantee that each attribute has at least $k$ vertices.

**Problem statement.** Given an attributed graph $G$ and an integer $k$, our goal is to enumerate all weak fair cliques and strong fair cliques in $G$ respectively.

**Example 3:** Reconsider the attributed graph $G$ in Fig. 1(a). Suppose that $k$ equals 2. We aim to find all 2-weak fair cliques and 2-strong fair cliques in $G$. The answer of 2-weak fair clique enumeration is $C = \{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$ because it is the maximal clique satisfying Definition 1. We can also find that there are three 2-strong fair cliques in $G$, i.e., $C_1 = \{v_1,v_2,v_3,v_4,v_5\}$, $C_2 = \{v_1,v_2,v_7,v_4,v_5,v_6\}$, and $C_3 = \{v_2,v_3,v_4,v_5,v_6\}$.
and $C_4 = \{v_2, v_3, v_7, v_4, v_5, v_6\}$, thus they are the answers for 2-strong fair clique search. Clearly, all 2-strong fair cliques are subgraphs of the 2-weak fair clique. □

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 (see Example 2), which means that it is difficult to check the maximality of strong fair cliques.

Unlike traditional maximal cliques, both weak fair cliques and strong fair cliques 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; and propose the SFCEnum algorithm with a novel attribute-alternatively-selection strategy for enumerating all strong fair cliques. Both of our algorithms are able to correctly find all fair cliques and significantly improve the efficiency compared to the baseline enumeration algorithm.

III. 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.

A. 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 [18], [26]. 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 3: (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 4: (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 $u \in V_H$, $D_{\min}(u, G) \geq k$; (2) there is no subgraph $H' \subseteq G$ that satisfies (1) and $H \subseteq H'$.

Based on Definition 4, 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 $G$.

Proof: Assume that $C$ is a weak fair clique and consider a vertex $u \in C$. Based on Definition 1, for each $a_i \in A_{val}$, $u$ has at least $k-1$ neighbors in $C$ whose attribute value is $a_i$. Since the vertices with the same color must not be adjacent, we have $D_{a_i}(u, G) \geq D_{\min}(u, G) \geq k-1$ for each $a_i \in A_{val}$. Thus, if a subgraph $g \subseteq G$ satisfies $D_{\min}(u, g) < k-1$, $C$ must not be included in $g$. □

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 [4], [25] (lines 8-20). Specifically, it first performs greedy coloring on $G$ which colors vertices based on the order of degree [16], [27] (line 1). Note that finding the optimal coloring is an NP-hard problem [18], [26], 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 G as the colorful k-core. Below, we analyze the complexity of Algorithm 1.

**Example 4:** Consider the graph G = (V, E, A) in Fig. 1(a). Assume that we want to search all 2-weak fair cliques. By Lemma 1, we invoke ColorfulCore to calculate the colorful-1-core of G. Specifically, we first color the vertices of G using the greedy method. Then, we obtain a colored graph which is illustrated in Fig. 1(b) with seven different colors. Take the vertex v8 as an example. v8 connects to v1 and v7 in G and both of them have attribute value a, thus Dv(v8) = 2 and Dh(v8) = 0 hold. Due to Dmin(v8) = Dh(v8) = 0 < 1, v8 is not contained in any 2-weak fair clique. Thus, ColorfulCore removes v8 from G. The removal of v8 subsequently updates the colorful-degrees of v1 and v7. ColorfulCore repeatedly removes vertices until all the remaining vertices satisfying Dmin ≥ 1. Finally, we can obtain a subgraph induced by the vertex set V - {v8} which is a colorful-1-core with Dmin = 2.

**Theorem 1:** Algorithm 1 consumes O(E + V) time using O(V × A × color) space, where color denotes the total number of colors.

**Proof:** In line 1, the greedy coloring procedure takes O(E + V) time [16]. In lines 2-7, we can easily derive that the algorithm takes O(E + V) time. In lines 11-20, the algorithm can update Mv for each v ∈ N(u) in O(1) time. For each edge (u, v), the update operator only performs once, thus the total time complexity is bounded by O(E + V). For the space complexity, the algorithm needs to maintain the structure Mv for each vertex which takes at most O(V × A × color) space in total.

**B. 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 Dmin(u, G) ≥ (k - 1) > Dmin(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 whose Dmin values are no less than Dmin(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 Dmin from the remaining graph. The vertices-removal ordering by this procedure is denoted as ColorOD.

Algorithm 2 outlines the pseudo-code of CalColorOD. For each vertex u, we use O(u) to indicate the rank of u in our order O. A heap-based structure H is employed to maintain the vertices with their Dmin values, which always pops out the pair (u, Dmin(u)) with minimum Dmin. CalColorOD first calculates Dmin(u) for every vertex u and pushes (u, Dmin(u)) into H (lines 3-5). Then, CalColorOD iteratively pops out the vertex with minimum Dmin from H and records its rank in O (lines 6-15). As a vertex is removed, we maintain the Dmin 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.

**C. 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.

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 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 CC containing u and sets B as true for all vertices within CC to denote that CC will not be searched again (line 6). WFCEnum then calls CalColorOD to derive the search order ColorOD of vertices in CC, and performs
the BackTrack procedure on CC 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 \( C \) and \( R \), if \( |C| + |R| < k \times A_n \), BackTrack terminates as the sets cannot reach the minimum size of a weak fair clique (line 18). On the other hand, we use \( \hat{R}_{cnt} \) and \( \hat{C}_{cnt} \) 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 \( X \) 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).

IV. 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 (\(|A_n| > 2 \)).

A. 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 5:** (Fairness degree) Given a colored attributed graph \( G = (V, E, A) \) with \( A_{val} = \{a_1, a_2\} \), the fairness degree of a vertex \( u \), denoted by \( FD(u) \), is the largest number of vertices from which we select vertices so that the number of vertices with attribute \( a_1 \) is the same as the number of vertices with attribute \( a_2 \).

By Definition 5, 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 both \( a_1 \) and \( a_2 \). Let \( c_1, c_2 \), and \( c_m \) be the number of the OA1Group, the OA2Group, 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 \( FD(u) = 2 \times ((c_m − (c_2 − c_1))/2) + 2 \).

Based on these results, we can easily derive the fairness degree for each vertex by using the three quantities \( c_1, c_2 \), and \( c_m \). The pseudo-code of our algorithm to compute the fairness is given in lines 17-29 of Algorithm 4.

Based on 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 6:** (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 \( u \in V_H \), \( FD(u) \geq 2k \); (2) there is no subgraph \( H' \subseteq G \) that satisfies (1) and \( H \subset H' \).

By Definition 6, we can show that any strong fair clique must be contained in the fairness \( k \)-core.

**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 \( G \).
Algorithm 4: FairnessCore

| Input: $G = (V, E, A)$, an integer $k$ |
| Output: The reduced graph $\hat{G}$ |
| $\hat{G} = (V, \hat{E}, \hat{A})$ ← ColorfulCore($G, k$); |
| 1. Let $FD$ be an array of size $|V|$; Let $Q$ be a queue; |
| 2. for $u \in V$ do |
| 3. for $v \in N(u)$ do |
| 4. $[\text{Group}(u, \text{color}(v), v.val)++; |
| 5. $FD(u) ← \text{FairDegCal}(u, \text{Group})$; |
| 6. if $FD(u) < 2 \times k$ then |
| 7. $\text{Remove } u \text{ from } Q$. $Q.push(u)$; |
| 8. while $Q \neq \emptyset$ do |
| 9. $u ← Q.pop(); |
| 10. for $v \in N(u)$ do |
| 11. $\text{if } v \text{ is removed then continue;}$ |
| 12. $\text{Group}(u, \text{color}(u), u.val) ← -1; |
| 13. $\text{Calculate } FD(v)$ and update $Q$ as lines 6-8; |
| 14. $\hat{G} ← \text{the remaining graph of } \hat{G}; |
| 15. $\text{return } \hat{G}; |
| 16. $\text{Procedure FairDegCal}(u, \text{Group})$; |
| 17. $c_u ← 0; c_{v, cr, a} ← 0; |
| 18. for each color $cr$ do |
| 19. if $\text{Group}(u, cr, a_{cr}) ≥ 1$ and $\text{Group}(u, cr, a_2) = 0$ then |
| 20. $c_1 ← c_1 + 1; |
| 21. if $\text{Group}(u, cr, a_2) ≥ 1$ and $\text{Group}(u, cr, a_1) = 0$ then |
| 22. $c_2 ← c_2 + 1; |
| 23. if $\text{Group}(u, cr, a_1) ≥ 1$ and $\text{Group}(u, cr_1) ≥ 1$ then |
| 24. $c_m ← c_m + 1; |
| 25. if $c_1 ≤ c_2$ then |
| 26. if $a_u ≥ (c_2 - c_1)$ then $FD(u) ← 2 \times ((c_m - (c_2 - c_1)) / 2 + c_2)$; |
| 27. else $FD(u) ← 2 \times (c_m + c_1); |
| 28. else $FD(u) ← 2 \times (c_m + c_2); |
| 29. return $FD(u);$ |

**Proof:** Consider a strong fair clique $C$. According to Definition 2, assume there are $k$ vertices of attribute $a_1$ and $k$ vertices of attribute $a_2$ in $C$. For an arbitrary vertex $u$ in $C$, we suppose that $u.val = a_1$. There are $k-1$ vertices of attribute $a_1$ and $k$ vertices of attribute $a_2$ in $u$’s neighbors. Therefore, after performing FairDegCal for $u$, we have $c_1 = k-1, c_2 = k$ and $c_m = 0$. Further, $FD(u) = 2(k-1)$. Due to the arbitrariness of $u$, the fairness degree of each vertex in $C$ must reach $2(k-1)$, too. Hence, $C$ must be contained in the fairness-$(k-1)$-core of $G$. $\square$

**Example 5:** Reconsider the attributed graph in Fig. 1(b). Suppose that $k = 3$. By Lemma 2, we consider the fairness 2-core of $G$. For vertex $v_8$, $v_9$ has two neighbors $v_1$ and $v_7$, and both of them have attribute value $a_1$. Clearly, we have $FD(v_9) = 0 < 2 \times 2$, thus $v_9$ is not contained in the fairness 2-core. For vertex $v_1$, the initial value of $c_1$ and $c_2$ and $c_m$ are 2, 3, 1. Obviously, $c_m + c_1 = c_2$, thus we have $FD(v_1) ≥ 4 > 4$. Similarly, the fairness degrees of other attributes are all equal to 6. Therefore, the subgraph induced by $V \setminus \{v_8\}$ is a fairness 2-core. Clearly, such a subgraph contains the strong fair clique as illustrated in Example 2. $\square$

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 $O((E + V) \times V)$ time using $O(V \times color)$ space.

**Proof:** In line 1, Algorithm 4 invokes Algorithm 1 which takes $O(V + E)$ time and $O(V \times color)$ space (since $A_0 = 2$). The FairDegCal procedure takes at most $O(color)$ time for each vertex. Therefore, the total time overhead taken in lines 3-8 is $O(V \times color + E)$. In lines 9-14, for each edge $(u, v)$, the update cost is bounded by $O(color)$, thus the total time complexity is $O((E + V) \times color)$. For the space complexity, the algorithm takes $O(V \times color)$ space to maintain the Group structure. $\square$

**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.

**B. 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 $a_{\phi}$, 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 $A_\phi$ sets, where the attribute values of vertices in each set are the same, i.e., $C_A(a_{\phi}) = \{u | u \in C, u.val = a_{\phi}\}$ (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 $R$ and $C$ to determine whether to call StrongBackTrack for a more in-depth search (lines 16-27). Specifically, we classify the candidates in $C$ 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 satisfying $|C_A(a_{\phi})| = c_{\min}$, we pick $d$ with the largest $i$ as $c_{\min}$. Clearly, $c_{\min}$ determines how large a strong fair clique can be. We use $R_e$ to denote the largest size of possible strong fair cliques. If $|R| \% A_n = 0$, the numbers of vertices with various attribute values are the same in the current set $R$, thus there are at most $c_{\min} \times A_n$ vertices can be added into $R$, and further we have $R_e = c_{\min} \times A_n + |R|$. 

264

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23, 2024 at 06:04:59 UTC from IEEE Xplore. Restrictions apply.
Algorithm 5: SFCEnum

```
Input: \( G = (V, E, A) \), an integer \( k \)
Output: The set of all the strong fair cliques \( Res \)
1. \( Res = \emptyset \); \( R = \emptyset \); \( C = \emptyset \);
2. \( G = (V, E) \leftarrow \text{FairnessCore}(G, k - 1) \);
3. Initialize an array \( B \) with \( B(i) = \text{false}, 1 \leq i \leq |V| \);
4. for \( u \in V \) do
5.  \( B(u) = \text{false} \) then
6.   \( CC \leftarrow \text{ConnectedGraph}(u, B) \);
7.   \( C \leftarrow \text{FairOD}(CC) \);
8.   \( R \leftarrow C \cup \emptyset \); \( \text{StrongBackTrack} (R, CC, a_0, O) \);
9. return \( Res \);

Procedure \( \text{StrongBackTrack} (R, C, a_0, O) \)
10. if \( |R| \cap A_n = \emptyset \) and \( |R| \geq k \times A_n \) then
11.  if \( \text{IsMaximal}(C) \) then
12.      \( Res \leftarrow Res \cup R; \) return;
13. for \( u \in C \) then
14.   \( C_A(u.val) \leftarrow C_A(u.val) \cup u \);
15. for \( u \in C_A(a_0) \) do
16.   \( R \leftarrow R \cup u \);
17. for \( v \in C \) do
18.   if \( v = N(u) \) and \( C(v) > C(u) \) then
19.      \( C \leftarrow C \cup u; \) \( C_A(u.val) \leftarrow C_A(v.val) \cup v \);
20. \( c_{\min} \leftarrow \min(|C_A(a_i)|); \) \( a_{\min} \leftarrow \arg \min_{a_i} |C_A(a_i)| \);
21. if \( |R| \cap A_n = \emptyset \) then \( R \leftarrow R \cup \min \times A_n + |R| \);
22. else
23.    if \( c_{\min} \in \{a_0, a_{n,1}, \ldots, a_n\} \) then
24.       \( R \leftarrow R \cup \min \times A_n + (|R|/A_n + 1) \times A_n \);
25.    else \( R \leftarrow (c_{\min} - 1) \times A_n + (|R|/A_n + 1) \times A_n \);
26. if \( R \leq k \times A_n \) then continue;
27. \( \text{StrongBackTrack} (R, C, a_{n+1}, O) \);
```

Algorithm 6: IsMaximal(C)

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

(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 \( a_0 \), the number of vertices with attribute value \( a_f \) (\( a_f \in \{a_0, \ldots, a_n\} \)) is always one more than that of vertices with \( a_b \) (\( a_b \in \{a_{n+1}, \ldots, a_{n-1}\} \)) in \( R \). If \( a_{\min} = a_f \), we can add one vertex, for each \( a_b \), into \( R \) to obtain a clique with size \( (|R|/A_n + 1) \times A_n \), 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 \( a_{\min} = a_b \), 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 \( \text{StrongBackTrack} \) with the attribute value \( a_{n+1} \) (line 27).

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, 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.

Once the StrongBackTrack procedure finds a clique whose size is equal to \( k' \times A_n \) with \( k' \geq k \), 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 not destroy the fairness property of \( R \), \( R \) is a strong fair clique 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 represent the vertices in \( C \) with the attribute value \( a_i \). Clearly, if \( |C_i| = 0 \) 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 at least \( A_n + |R| \). In such case, \( R \) is not a strong fair clique and the StrongBackTrack procedure returns false (lines 6-14).

C. 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 cases, because there may be \( 2^{A_n} - 1 \) \( A_m \) 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 Group\((cr)\). For a color group Group\((cr)\), we let Group\((cr)\) be the set of attributes of the vertices in Group\((cr)\). For an attribute \( a_i \), if \( a_i \in \text{Group}(cr) \) and \( |\text{Group}(cr)| = 1 \) 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 \( \text{cnt}(a_i) \) to record the number of color groups that only contain vertices with \( a_i \). Clearly, \(|\text{Group}(cr)| > 1 \) 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...
Algorithm 7: CalHeurOrd

Input: A connected graph $G = (V, E)$
Output: The HeurOD ordering $O$
1 $O ← ∅; \ Q ← ∅$
2 Let $B$ be an array with $B(i) = \text{false}, 1 \leq i \leq |V|$
3 for $u \in V$ do
   4   for $v \in N(u)$ do
      5       $S_u(color(v), v.val) ← S_u(color(v), v.val) + 1$;
   6   Let cnt be an array with $cnt(i) = 0, 0 \leq i < A_n$;
7   for each color $cr$ do
8      for $a_i \in A_{val}$ do
9         if $S_u(cr, a_i) \geq 1$ then
10            $a_{\text{min}} = \arg \min_{a_i \in S_u(cr, a_i)} cnt(a_i)$;
11            $cnt(a_{\text{min}}) ← cnt(a_{\text{min}}) + 1$;
12            $GD(u) = \min\{cnt(a_i), a_i \in A_{val}\}$;
13            $Q.push(u, GD(U))$;
14   while $Q \neq ∅$ do
15      $u ← Q.pop(); Q.push(u); B(u) = \text{true}$;
16   for $v \in N(u)$ do
17      if $B(v) = \text{false}$ then
18         $S_u(color(u), u.val) ← S_u(color(u), u.val) - 1$;
19         Calculate $GD(v)$ and update $Q$ as lines 6-13;
20 return $O$;

1 where $a_{\text{min}} = \arg \min_{a_i \in S_u(cr)} cnt(a_i)$. Finally, $GD(u)$ is obtained by taking the minimum counter over all attributes, i.e., $GD(u) = \min\{cnt(a_i), a_i \in A_{val}\}$.

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.

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

Proof: It is easy to derive that the time complexity to compute $GD$ for all vertices is $O((V + E) \times A_n)$ (lines 3-13). The total cost to update the $GD$ in line 19 is $O(E \times \text{color} \times A_n)$. Therefore, the total time complexity is $O((V + E) \times A_n \times \text{color})$. For the space complexity, the algorithm takes $O(V \times \text{color} \times A_n)$ space to maintain all $S_u$, and $O(V)$ to maintain all $GD$. Thus, the total space overhead of the algorithm is $O(V \times A_n \times \text{color})$. □

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 $A_n \geq 2$. 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).

V. EXPERIMENTS

A. 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 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. Since there is no existing algorithm that can be directly used to enumerate fairness-aware cliques, we implement two baseline algorithms, called BaseWeak and BaseStrong. For the weak fair clique enumeration, BaseWeak first finds all maximal cliques using the state-of-the-art Bron-Kerbosch algorithm with pivoting technique [8], [41], and then filters them based on attribute constraint to identify weak 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 Bs5OD, 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., Bs5OD, 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 four real-world graphs to evaluate the efficiency of the proposed algorithms. Table I summarizes the statistics of the datasets in our experiments. WikiTalk is a communication network. Themarker, Slashdot and Flixster are social networks. All datasets can be downloaded from networkrepository.com/ and snap.stanford.edu. Note that all these datasets are non-attributed graphs, thus we randomly assign an attribute to each vertex to generate attributed graphs which will be used to evaluate the efficiency of all algorithms.

Parameters. There are two parameters in our algorithms: $k$ and $d = A_n$. The parameter $k$ is the threshold for fair cliques and $d$ is the number of attribute values (i.e., the attribute dimension). Since different datasets have various scales, the parameter $k$ is set within different integers. For Themarker,
$k$ is chosen from the interval $[7, 11]$ with a default value of $k = 4$. 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 $d = 2$. Unless otherwise specified, the values of the other parameters are set to their default values when varying a parameter.

### B. Efficiency testing

#### Evaluation of the pruning techniques

For the 2D case ($d = 2$), both ColorfulCore and FairnessCore can be used to reduce the graph size in the SFCEnum algorithm. In this experiment, we first evaluate these two pruning techniques by comparing the number of remaining vertices after pruning with varying $k$. The results are depicted in Fig. 2 (a)-(d). As expected, both ColorfulCore and FairnessCore can significantly reduce the number of vertices compared to the original graph. For example, in Slashdot with $k = 9$, ColorfulCore reduces the number of vertices from 82,169 to 3,985; and FairnessCore further reduces the number of vertices to 1,335. In general, FairnessCore consistently outperforms ColorfulCore in terms of the pruning performance, especially for relatively small $k$ values. As expected, when $k$ goes larger, the number of remaining vertices becomes smaller. Additionally, we can also observe that the pruning performance of ColorfulCore is slightly worse than that of FairnessCore for a large $k$. This is because FairnessCore first invokes ColorfulCore to prune unpromising vertices. Since ColorfulCore is already able to prunes 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 to reduce the graph size.

Note that for the high-dimensional case ($d \geq 3$), only the ColorfulCore algorithm can be used to prune the unpromising vertices in both WFCEnum and SFCEnum. Therefore, we further study how the dimension $d$ affects the pruning performance of ColorfulCore. Fig. 2 (e)-(h) 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. 3. As can be seen, BaseWeak can only output the results on Slashdot 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. 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 effectiveness 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 $d = 1$, we only evaluate SFCEnum with FairOD by varying $k$. The experimental results of SFCEnum are illustrated in Fig. 4. 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 SFCEnum algorithms with FairOD and HeurOD are faster than those with BfsOD and VidOD. 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 effectiveness of the proposed ordering techniques.

Additionally, by comparing BaseStrong and SFCEnum, we find that the running time of BaseStrong on all datasets exceeds the time limit, thus we do not show them in Fig. 4. 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 $k \times A$, first. The number of such cliques are often extremely large, thus the runtime of BaseStrong is significantly higher than SFCEnum.

**The number of fairness-aware cliques.** Fig. 5 (a)-(d) shows the numbers of weak fair cliques and strong fair cliques with different $d$. Clearly, there are significant numbers of fair cliques in each dataset. In general, the number of strong fair cliques is larger than that of weak fair cliques. This finding is consistent with our analysis in Section II, since a weak fair clique often contains 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 less number of fair cliques. Similar results can also be observed when varying $d$ from Fig. 5 (e)-(h).

**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. 6 illustrates the results on Flixster. The results on the other datasets are consistent. In Fig. 6(a), 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. However, for SFCEnum with FairOD and HeurOD, the runtime increases smoothly with $m$ increases. These results demonstrate the high scalability of the proposed algorithms.

**Memory overhead.** Fig. 7 shows the memory overheads of the enumeration algorithms with different orderings on all datasets. Note that the memory costs of different algorithms do not include the size of the graph. From Fig. 7, we can see that the memory usages of different algorithms 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, we can see that the memory usages are robust w.r.t. (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 w.r.t. different orderings.

**C. Case study**

We conduct a case study on a collaboration network DBLP to evaluate the effectiveness of our algorithms. The DBLP 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 WFCEnum and SFCEnum to find all weak fair cliques and strong fair cliques on DBCS with $k = 2$. Both algorithms apply ColorfulCore to prune the unpromising vertices. The remaining graph after pruning by ColorfulCore only has 61 vertices and 516 edges. Fig. 8(a) illustrates 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, green = $DB$, pink = $DM$, 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 relatively-fair communities with diverse research areas. However, in Fig. 8(a), the weak fair clique is imbalanced (w.r.t. different attributes) due to the high percentage of authors with $DB$. Fig. 8(b) and Fig. 8(c) show two strong fair cliques which
are also subgraphs of the clique in Fig. 8(a). 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. These results demonstrate that both WFCEnum and SFCEnum can be used to find fair communities with diverse attributes; and SFCEnum can further keep balance over different attributes in the community. Moreover, this case study also indicates that the weak fair cliques and strong fair 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 at cross-cutting areas.

D. 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-of-the-art maximum clique search algorithms [9], [37]. Suppose the size of a maximum clique is $C_{\text{max}}$. Then, the parameter $k$ in our...
fair clique models satisfies \( k \leq \left\lfloor \frac{C_{\text{max}}}{A_n} \right\rfloor \). Note that when the maximum clique size is hard to compute for some instances, an alternative solution is to compute an approximation of \( C_{\text{max}} \) by using a linear-time greedy algorithm [34]. Therefore, for a particular application, we can use a binary search method to find an appropriate \( k \) from the interval \( [1, \left\lfloor \frac{C_{\text{max}}}{A_n} \right\rfloor] \) by invoking the proposed algorithms to compute the fair cliques.

VI. 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 [13], [19], [24], [33], [43], [45]. For example, Li et al. [24] propose an embedding-based model to discover communities in attributed graphs. Tong et al. [43] studied the problem of finding subgraphs for given query patterns in attributed graphs. Fang et al. [13] investigated the attributed community search problem and developed an index structure, called CL-tree, to efficiently support attributed community search. Khan et al. [19] 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. [33] 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 [44]. Zehlike et al. [50] 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. [36] investigated a problem of fairness in package-to-group recommendation, and proposed a greedy algorithm to find approximate solutions. Beutel et al. [5] 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 including demographic parity [11] and equality of opportunity [15]. For instance, Hardt et al. [15] proposed a framework that can optimally adjust any learned predictor to reduce bias. Compare 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 machine learning literature.

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 [30], [21]. Ostergard et al. [30] presented a branch-and-bound algorithm with the vertex order taken from a coloring of the vertices. Konc et al. [21] 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 [39], MCR [38], MCS [40] and MCT [37], [42], 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-attribute 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 attribute graphs.

Another researching problem of clique mining is to enumerate maximal cliques. The well-known algorithm for enumerating all maximal cliques is the classic Bron-Kerbosch (BK) algorithm [8]. Tomita et al. [41] proposed an algorithm, using a greedy pivoting technique, to find all maximal cliques. Eppsten et al. [12] further improved the BK algorithm based on a heuristic degeneracy ordering. In addition, some relaxed definitions of clique were also proposed, such as \( n \)-clique [2], \( n \)-clan, \( n \)-club [28], \( n \)-plex [3], [35], quasi-clique [1], [32], \( k \)-core [10], [20], [29], and so on [7]. However, the solutions mentioned above are not tailored for attributed graphs, thus cannot be directly used to solve our problems. In this work, we develop novel algorithms to compute maximal fair cliques in attributed graphs with several non-trivial pruning techniques.

VII. CONCLUSION

In this paper, we study a problem of enumerating fairness-aware cliques in an attributed graph. To this end, we propose a weak fair clique model and a strong 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 case. We conduct extensive experiments using four large real-life graphs, and the results demonstrate the efficiency and effectiveness of the proposed algorithms.

There are several future directions deserved to further investigate. 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) to define the fairness-aware cohesive subgraphs. 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.

Acknowledgement. This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants 62072034, U1809206, and 61772346. Rong-Hua Li is the corresponding author of this paper.
REFERENCES

[1] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In Latin American symposium on theoretical informatics, pages 598–612, 2002.
[2] Richard D Alba. A graph-theoretic definition of a sociometric clique. Journal of Mathematical Sociology, 3(1):113–126, 1973.
[3] Balabhaskar Balasundaram, Sergiy Butenko, and Ilyia V Hcks. Clique relaxations in social network analysis: The maximum k-plex problem. Operations Research, 59(1):133–142, 2011.
[4] Vladimir Batagelj and Matjaz Zaversnik. An o(m) algorithm for cores decomposition of networks. CoRR, cs.DS/0310049, 2003.
[5] Alex Beutel, Jiin Chen, Tulsee Doshi, et al. Fairness in recommendation ranking through pairwise comparisons. In SIGKDD, 2019.
[6] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. Mining market data: a network approach. Computers & Operations Research, 33(11):3171–3184, 2006.
[7] Stephen P Borgatti, Martin G Everett, and Paul R Shirey. Ls sets, lambda sets and other cohesive subsets. Social networks, 12(4):337–357, 1990.
[8] Cono Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM, 16(9):575–577, 1973.
[9] Lijun Chang. Efficient maximum clique computation over large sparse graphs. In Ankur Teredesai, Vinip Kumar, Ying Li, Römer Rosales, Evimaria Terzi, and George Karypis, editors, KDD, 2019.
[10] Sergey N Dorogovtsev, Alexander V Goltsev, and Jose Ferreira F Mendes. K-core organization of complex networks. Physical review letters, 96(4):040601, 2006.
[11] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In J. Glob. Optim., 44(2):311, 2009.
[12] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Ploutarchos Spyridonos. Community detection in social media. Data Mining and Knowledge Discovery, 24(3):515–554, 2012.
[13] J Pardalos and M Resende. On maximum clique problems in very large graphs. DIMACS series, 50:119–130, 1999.
[14] Clara Pizzuti and Annalisa Socievole. A genetic algorithm for community detection in attributed graphs. In EvooApplications, pages 159–170, 2018.
[15] Ryan A Rossi, David F Gleich, and Assefaw H Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. SIAM Journal on Scientific Computing, 37(5):C589–C616, 2015.
[16] Dimitris Serpos, Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsiparas. Fairness in package-to-group recommendations. In WWW, 2017.
[17] Etsuji Tomita. Efficient algorithms for finding maximum and maximal cliques and their applications. In Moti意-Hung Poon, Md. Saidur Rahman, and Hou-Chuan Yen, editors, WALCOM, 2017.
[18] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim., 44(2):311, 2009.
[19] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique. In Cristian Calude, Michael J. Dinneen, and Vincent Vajnovszki, editors, DMTCs, 2003.
[20] Etsuji Tomita, Yoichi Sutani, and Mitsuo Wakatsuki. A much faster branch-and-bound algorithm for finding a maximum clique. In Md. Saidur Rahman and Satoshi Fujita, editors, WALCOM, 2010.
[21] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical computer science, 363(1):28–42, 2006.
[22] Etsuji Tomita, Kohei Yoshida, Takuro Hatta, Atsuki Nagao, Hiro Ito, and Mitsuo Watsuki. A much faster branch-and-bound algorithm for finding a maximum clique. In Daming Zhu and Sergey Bereg, editors, FAW, 2016.
[23] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In SIGKDD, 2007.
[24] Clara Pizzuti and Annalisa Socievole. A genetic algorithm for community detection in attributed graphs. In SIGMOD, 2007.
[25] Clara Pizzuti and Annalisa Socievole. A genetic algorithm for community detection in attributed graphs. In SIGMOD, 2007.
[26] Clara Pizzuti and Annalisa Socievole. A genetic algorithm for community detection in attributed graphs. In SIGMOD, 2007.
[27] John Mitchell. On various algorithms for estimating the chromatic number of a graph. The Computer Journal, 19(2):182–183, 1976.