FairAD: Computationally Efficient Fair Graph Clustering via Algebraic Distance

Minh Phu Vuong
Texas State University
San Marcos, TX, USA
cty13@txstate.edu

Young-Ju Lee
Texas State University
San Marcos, TX, USA
yjlee@txstate.edu

Iván Ojeda-Ruiz
Lamar University
Beaumont, TX, USA
iojedaruiz@lamar.edu

Chul-Ho Lee
Texas State University
San Marcos, TX, USA
chulho.lee@txstate.edu

Abstract

Due to the growing concern about unsavory behaviors of machine learning models toward certain demographic groups, the notion of 'fairness' has recently drawn much attention from the community, thereby motivating the study of fairness in graph clustering. Fair graph clustering aims to partition the set of nodes in a graph into k disjoint clusters such that the proportion of each protected group within each cluster is consistent with the proportion of that group in the entire dataset. It is, however, computationally challenging to incorporate fairness constraints into existing graph clustering algorithms, particularly for large graphs. To address this problem, we propose FairAD, a computationally efficient fair graph clustering method. It first constructs a new affinity matrix based on the notion of algebraic distance such that fairness constraints are imposed. A graph coarsening process is then performed on this affinity matrix to find representative nodes that correspond to k clusters. Finally, a constrained minimization problem is solved to obtain the solution of fair clustering. Experiment results on the modified stochastic block model and six public datasets show that FairAD can achieve fair clustering while being up to 40 times faster compared to state-of-the-art fair graph clustering algorithms.

CCS Concepts

• **Mathematics of computing → Graph algorithms**; • **Theory of computation → Unsupervised learning and clustering**; • **Information systems → Clustering**.

Keywords

Graph Clustering, Spectral Clustering, Fairness

ACM Reference Format:

Minh Phu Vuong, Young-Ju Lee, Iván Ojeda-Ruiz, and Chul-Ho Lee. 2025. FairAD: Computationally Efficient Fair Graph Clustering via Algebraic Distance. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746252.3761320

CIKM '25, Seoul, Republic of Korea

ACM ISBN 979-8-4007-2040-6/2025/11
https://doi.org/10.1145/3746252.3761320

1 Introduction

Recent advancements in machine learning (ML) have enabled its integration into decision-critical applications across various domains, including finance, healthcare, education, and law enforcement. Despite their significant capabilities, ML algorithms are susceptible to biases present in datasets, resulting in potentially *unfair* outcomes for certain demographic groups [14]. Thus, fairness criteria have been introduced into ML problems, ranging from supervised and unsupervised learning settings [4, 7, 10, 11, 13, 18, 30, 31, 34] to semi-supervised and self-supervised settings [6, 39, 40], to eliminate unwanted algorithmic bias and develop fair ML models.

Fairness refers to the unbiased treatment of individuals or groups across various demographic categories, such as race, gender, age, and socioeconomic status. In general, fairness can be incorporated in an ML problem by introducing a fairness regularizer term to its objective function; formulating an optimization problem with explicit fairness constraints; or post-processing the output of a model to account for the fairness. As a result, it leads to an unbiased outcome in the target ML task or the representations that are invariant to protected attributes or have feature distributions that are statistically indistinguishable across demographic groups.

Fairness has been characterized by several concepts, including individual fairness [17, 27], group fairness [12], and counterfactual fairness [19]. Individual fairness requires similar individuals to receive similar outcomes, while counterfactual fairness demands that an individual's outcome remains unchanged if only their protected attribute were hypothetically modified, with all the other features being held constant. Group fairness ensures a fairly proportional representation across demographic groups. Given the prevalence of demographic groups in real-world datasets, ensuring group fairness has become a critical requirement, especially when it comes to clustering applications.

Chierichetti et al. [7] pioneered the integration of fairness into k-center and k-median clustering algorithms. They introduced the concept of fairness by ensuring that the proportion of each demographic group within each cluster is consistent with its overall proportion in the dataset. Backurs et al. [4] extended their approach to efficiently handle larger datasets with near-linear running time.

More recently, several works have addressed fair k-clustering in the presence of outliers [1, 2]. They first identify a subset of points as outliers to remove and partition the remaining data so that each cluster preserves the overall demographic proportions. These methods, however, have been predominantly applied to clustering tasks in the Euclidean space, but clustering problems also often occur in the context of graph data.

Graph clustering is a fundamental problem and has been extensively studied in the literature. Among others, spectral clustering [26] has been the most popular unsupervised graph-clustering algorithm as it is developed in a principled way to find the optimal solution to a well-defined graph cut problem. It has also been actively extended to variants of the graph cut problem for various reasons, e.g., improving the quality of clustering with assistive input from the user [8, 20, 32, 35, 37, 38]. For example, Xu et al. [35] incorporate linear constraints to the objective of spectral clustering, while Wang et al. [32] enforce prior knowledge via must-link and cannot-link constraints. These constraint-driven methods are especially effective for image segmentation applications, where a small set of annotated pixels helps the algorithm produce more accurate segmentation boundaries.

Kleindessner et al. [18] introduced a mathematical framework that imposes the notion of fairness as additional linear constraints into the problem of spectral clustering. Their algorithm, which we name as FairSC, however, faces scalability issues for larger graphs due to its high computational cost. Wang et al. [30] recently proposed a scalable fair spectral clustering algorithm named sFairSC by reformulating the problem as a projected eigenvalue problem and effectively improving its scalability. While their algorithms improve the balance performance compared to the standard spectral clustering, their frameworks still rely on solving constrained or projected eigenvalue problems due to the fairness constraints. They generally take much longer than solving unconstrained eigenvalue problems as they require computing the nullspace of a fairness matrix or employing the nullspace projection.

We propose FairAD, a computationally efficient **fair** graph clustering method via **A**lgebraic **D**istance. In FairAD, we first construct a new affinity matrix based on the notion of algebraic distance such that the fairness constraints are imposed. We then employ a recursive graph coarsening process on the affinity matrix to find representative nodes that correspond to a given number of clusters. They eventually lead to a simple constrained minimization problem, which can be solved efficiently. We further optimize the implementation of FairAD through several techniques.

Our contributions can be summarized as follows:

- We introduce a novel framework to integrate fairness constraints into the affinity matrix for graph clustering, when constructed based on the algebraic distance.
- We demonstrate how graph coarsening can be effectively leveraged to convert the problem into a simpler minimization problem, which can be solved efficiently.
- We develop a series of implementation optimizations to further improve the efficiency of FairAD.
- We evaluate the effectiveness and efficiency of FairAD through extensive experiments on the modified stochastic block model and six real-world datasets. The results show that FairAD not only delivers fair clustering but also runs up to 40× faster than state-of-the-art fair graph clustering algorithms.

2 Preliminaries

Consider an undirected, weighted graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes and E is the set of edges. Each edge between nodes i and j is associated with a positive weight $W_{i,j} > 0$. $W_{i,i} = 0$ for all i, and $W_{i,j} = 0$ if nodes i and j are not neighbors. Let $\mathbf{W} = (W_{i,j})$ be the $n \times n$ weight matrix, which is also called affinity matrix. Let d_i be the degree of node i, which is defined as $d_i := \sum_{j \in V} W_{i,j}$, and let $\mathbf{D} := \mathrm{diag}(d_1, d_2, \ldots, d_n)$ be the degree matrix. For a subset of nodes $A \subset V$, we define $\mathrm{vol}(A) := \sum_{i \in A} d_i$ to be a volume of A. Also, for two subsets $A, B \subset V$, we define $W(A, B) := \sum_{i \in A, j \in B} W_{i,j}$. The Laplacian and normalized Laplacian matrices of G are defined as $\mathbf{L} := \mathbf{D} - \mathbf{W}$ and $\overline{\mathbf{L}} := \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$, respectively.

Notations. For an integer $n \geq 1$, let $[n] := \{1, 2, \ldots, n\}$. Let $\mathbf{1}$ and $\mathbf{0}$ denote the n-dimensional all-one and all-zero column vectors, respectively. Let $\mathbf{I}_k$ be the $k \times k$ identity matrix. For a node subset $A \subset V$, let $\overline{A}$ denote its complement $V \setminus A$.

2.1 Spectral Clustering

The basic problem of graph clustering is the minimum cut problem, which is to partition V into k disjoint subsets (clusters), i.e., $V = C_1 \cup C_2 \cup \cdots \cup C_k$, such that the sum of the weights of the edges across different clusters is minimized. That is, it is to find k disjoint subsets to minimize

$$\mathrm{Cut}(C_1, C_2, \ldots, C_k) := \frac{1}{2} \sum_{l=1}^{k} W(C_l, \overline{C}_l).$$

While this problem can be solved easily, it is widely known that its solution does not lead to satisfactory partitions, and it often separates an individual node from the rest of the graph. Thus, its *properly normalized* versions have been introduced and extensively studied in the literature [20, 28]. Among others, the normalized cut (NCut) problem is the most popular problem, and its corresponding 'spectral clustering' algorithm is widely used as an effective graph clustering algorithm [26].

Specifically, the NCut problem is to minimize

$$\mathrm{NCut}(C_1, C_2, \ldots, C_k) := \frac{1}{2} \sum_{l=1}^{k} \frac{W(C_l, \overline{C}_l)}{\mathrm{vol}(C_l)}. \tag{1}$$

Consider $k = 2$. Letting $\boldsymbol{h}$ be an indicator vector with entries $h_i = 1$ if $i \in C_1$ and $h_i = -1$ otherwise, the NCut problem in (1) becomes $\min_{h_i \in \{1,-1\}} \mathrm{NCut}(\boldsymbol{h})$, which can also be written as

$$\begin{aligned} &\min_{u_i \in \{\sigma, -1/\sigma\}} \boldsymbol{u}^\top \mathbf{L} \boldsymbol{u} \\ &\text{subject to } \boldsymbol{u}^\top \mathbf{D} \boldsymbol{u} = \mathrm{vol}(V) \text{ and } \boldsymbol{u}^\top \mathbf{D} \mathbf{1} = 0, \end{aligned} \tag{2}$$

where σ is some positive constant. Since this problem is NP-hard [29], by relaxing $\boldsymbol{u}$ to take arbitrary real values and substituting $\boldsymbol{v} := \mathbf{D}^{1/2}\boldsymbol{u}$, we have the following relaxed problem:

$$\begin{aligned} &\min_{\boldsymbol{v} \in \mathbb{R}^n} \boldsymbol{v}^\top \overline{\mathbf{L}} \boldsymbol{v} \\ &\text{subject to } \|\boldsymbol{v}\|^2 = \mathrm{vol}(V) \text{ and } \boldsymbol{v}^\top \mathbf{D}^{1/2} \mathbf{1} = 0. \end{aligned} \tag{3}$$

This boils down to finding the eigenvector corresponding to the smallest non-zero eigenvalue of the normalized Laplacian $\overline{\mathbf{L}}$ and evaluating the "sign" of each component of the eigenvector to partition the graph into two clusters.

For $k > 2$, by repeating the similar arguments as above, we can relax the k-way NCut problem as the following standard trace minimization problem, which is to find the $n \times k$ partition matrix $\mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$ to minimize the trace:

$$\min_{\mathbf{V} \in \mathbb{R}^{n \times k}} \operatorname{Tr}(\mathbf{V}^\top \overline{\mathbf{L}} \mathbf{V}) \text{ subject to } \mathbf{V}^\top \mathbf{V} = \mathbf{I}_k, \tag{4}$$

where $\operatorname{Tr}(\mathbf{V}^\top \overline{\mathbf{L}} \mathbf{V}) = \sum_{i=1}^{k} \boldsymbol{v}_i^\top \overline{\mathbf{L}} \boldsymbol{v}_i$. It is also known that the solution to the relaxed k-way NCut problem in (4) is to find the eigenvectors that correspond to the k smallest eigenvalues of $\overline{\mathbf{L}}$ [26].

2.2 Fairness Constraints

The notion of *fairness* can now be introduced in the context of graph clustering. The goal of fair graph clustering is to ensure that the proportion of nodes in each cluster is identical to the proportion of the population as a whole. Specifically, suppose that nodes are originally divided into h distinct groups, i.e., $V = V_1 \cup V_2 \cup \cdots \cup V_h$. Then, it aims to ensure that, for $s = 1, 2, \ldots, h$ and $l = 1, 2, \ldots, k$,

$$\frac{|V_s \cap C_l|}{|C_l|} = \frac{|V_s|}{|V|}. \tag{5}$$

Let $\boldsymbol{f}^{(s)} := [f_1^{(s)}, f_2^{(s)}, \ldots, f_n^{(s)}]$ be the group indicator vector for V_s, which has elements $f_i^{(s)} = 1$ if $i \in V_s$ and $f_i^{(s)} = 0$ otherwise. Also, let $\mathbf{F} = (F_{i,s})$ be an $n \times (h-1)$ matrix with elements $F_{i,s} := f_i^{(s)} - |V_s|/|V|$ for $s \in [h-1]$ and $i \in [n]$. Then, as shown in [18, 30], a partition $V = C_1 \cup C_2 \cup \cdots \cup C_k$ is *fair* if and only if its corresponding $n \times k$ partition matrix $\mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$ satisfies

$$\mathbf{F}^\top \mathbf{V} = \mathbf{0}_{(h-1) \times k}, \tag{6}$$

where $\mathbf{0}_{(h-1)\times k}$ is the all-zero matrix of dimension $(h-1) \times k$. In other words, the fairness constraints in (5) are equivalent to the linear constraints in (6). Therefore, the problem of *fair* spectral clustering now becomes

$$\min_{\mathbf{V}^\top \mathbf{V} = \mathbf{I}_k,\ \mathbf{F}^\top \mathbf{V} = \mathbf{0}_{(h-1)\times k}} \operatorname{Tr}(\mathbf{V}^\top \overline{\mathbf{L}} \mathbf{V}), \tag{7}$$

which is imposing the linear constraints in (6) into the problem of spectral clustering in (4).

To solve this problem efficiently, novel fair spectral clustering algorithms, i.e. FairSC and sFairSC, have been developed [18, 30]. While they improve the balance performance compared to spectral clustering, their frameworks still rely on solving constrained or projected eigenvalue problems due to the fairness constraints. They generally take much longer than solving unconstrained eigenvalue problems as they require computing the nullspace of $\mathbf{F}$ or employing the nullspace projection. As shall be demonstrated through the experiments, their computational time grows quickly with increasing size of the graph. Therefore, there is a need for an efficient and scalable approach for fair graph clustering.

3 Proposed Method: FairAD

In this section, we introduce FairAD, a computationally efficient fair graph clustering method. We first construct a new affinity matrix based on the notion of algebraic distance, where the fairness constraints are imposed. We then perform a recursive graph coarsening process on this affinity matrix to find representative (or anchor) nodes that correspond to k clusters. We finally determine which cluster each node in the original graph belongs to by solving a relaxed k-way graph cut problem where the representative nodes are used as additional linear constraints. In addition to the operations of FairAD, we also explain a set of implementation optimizations made to speed up FairAD in practice. Figure 1 illustrates an overview of FairAD.

3.1 Imposing Fairness Constraints

For a given graph with the affinity (weight) matrix $\mathbf{W}$, as the first step of FairAD, we construct a new affinity matrix such that the fairness constraints are imposed. To this end, we propose to use the algebraic distance, which was originally developed in [22] to measure the strength of a connection between each node pair in the graph and to construct a new affinity matrix to achieve better solutions to partitioning problems, albeit not under fairness constraints. We below explain how the process of computing the algebraic distance can be modified to impose the fairness constraints into a new affinity matrix.

We begin with the definition of algebraic distance. Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_R$ be the n-dimensional *test* vectors. Each test vector $\boldsymbol{x}_r$ is obtained by running τ Jacobi relaxation iterations [24] on $\mathbf{L}\boldsymbol{x}_r = \mathbf{0}$, where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the (unnormalized) Laplacian matrix. Starting from a random vector $\boldsymbol{x}_r^0$, each Jacobi relaxation iteration on $\mathbf{L}\boldsymbol{x}_r = \mathbf{0}$ leads to

$$\boldsymbol{x}_r^t = \boldsymbol{x}_r^{t-1} + \mathbf{D}^{-1}(\mathbf{0} - \mathbf{L}\boldsymbol{x}_r^{t-1}) = \mathbf{D}^{-1}\mathbf{W}\boldsymbol{x}_r^{t-1},\ t = 1, 2, \ldots, \tau - 1,$$

and we finally have a test vector $\boldsymbol{x}_r$ at $t = \tau$. Intuitively, in each iteration, the value of each node is updated based on the weighted average of the values of its neighbors. This iterative process effectively smooths out the values of the nodes that are strongly connected, while preserving the differences across the values of the weakly connected nodes. Then, the algebraic distance between nodes i and j is defined as

$$s(i, j) = \max_{r=1,2,\ldots,R} |x_{r,i} - x_{r,j}|, \tag{8}$$

where $x_{r,i}$ is the i-th element of test vector $\boldsymbol{x}_r$. Next, a new affinity matrix, say, $\mathbf{W}_{\text{alg}} := (W_{i,j}^{\text{alg}})$, is constructed based on the algebraic distance in (8) as follows [22]: For all i, j,

$$W_{i,j}^{\text{alg}} = \exp(-s(i, j)). \tag{9}$$

We here aim to incorporate the fairness constraints in (6) into the process of computing the algebraic distance in (8). Specifically, our approach is to impose the fairness constraints in (6) at *each* Jacobi relaxation iteration so that its resulting test vector, say $\boldsymbol{x}_r^t$, at iteration t satisfies the fairness constraints, i.e., $\mathbf{F}^\top \boldsymbol{x}_r^t = \mathbf{0}$. For brevity, we drop the subscript r as the test vectors are obtained in the same way.

First, observe that the vector $\boldsymbol{x}^t$ at the t-th Jacobi relaxation iteration is given by

$$\boldsymbol{x}^t = \mathbf{D}^{-1}\mathbf{W}\boldsymbol{x}^{t-1},$$

which leads to

$$\mathbf{D}\boldsymbol{x}^t = \mathbf{W}\boldsymbol{x}^{t-1}. \tag{10}$$

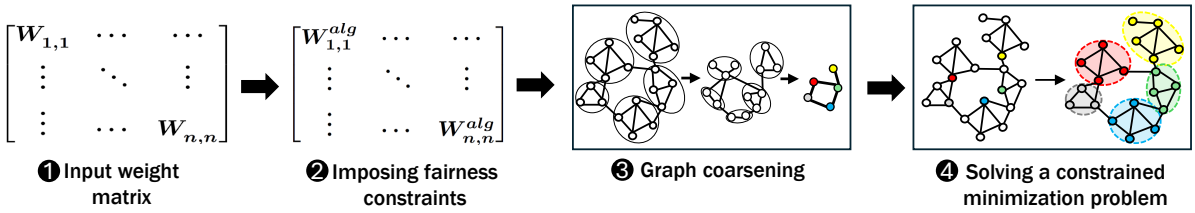


Figure 1: An overview of FairAD.

Algorithm 1: Constrained Jacobi (cJacobi)

Input : $\mathbf{W}$, $\mathbf{D}$, $\mathbf{F}$
Output: $\boldsymbol{x}^{\tau}$.
Initialize $\boldsymbol{x}^0$.
for $t = 1$ **to** τ **do**
  $\boldsymbol{x}^t := (\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1}\mathbf{W}\boldsymbol{x}^{t-1}$.
end for

We then incorporate the fairness constraints $\mathbf{F}^\top \boldsymbol{x}^t = \mathbf{0}$ into (10), which leads to

$$\mathbf{D}\boldsymbol{x}^t = \mathbf{W}\boldsymbol{x}^{t-1} \text{ subject to } \mathbf{F}^\top \boldsymbol{x}^t = \mathbf{0}. \tag{11}$$

Let $\boldsymbol{b} := \mathbf{W}\boldsymbol{x}^{t-1}$. Since we solve this system for each t, for ease of exposition, we also drop the superscript t. Next, we observe that the system in (11) is equivalent to the following quadratic optimization problem:

$$\min_{\mathbf{F}^\top \boldsymbol{x}=\mathbf{0}} \frac{1}{2}\boldsymbol{x}^\top \mathbf{D}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}. \tag{12}$$

We write its Lagrangian function, which is given by

$$\mathcal{L}(\boldsymbol{x}, \lambda) = \frac{1}{2}\boldsymbol{x}^\top \mathbf{D}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x} + \lambda \mathbf{F}^\top \boldsymbol{x},$$

where λ is the Lagrange multiplier. The KKT conditions applied to this Lagrangian function yield

$$\begin{pmatrix} \mathbf{D} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \mathbf{0} \end{pmatrix}, \tag{13}$$

which is a system of linear equations.

We leverage the augmented Lagrangian Uzawa method [15] to solve the indefinite system in (13) as it has a fast rate of convergence, implying that just one iteration provides a good approximate solution. Specifically, given $(\boldsymbol{x}^\ell, \lambda^\ell)$, a new iterate $(\boldsymbol{x}^{\ell+1}, \lambda^{\ell+1})$ is obtained by solving the following equations:

$$\begin{aligned} (\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)\boldsymbol{x}^{\ell+1} &= \boldsymbol{b} - \mathbf{F}\lambda^\ell, \\ \lambda^{\ell+1} &= \lambda^0 + \mu\mathbf{F}^\top \boldsymbol{x}^\ell, \end{aligned}$$

where μ is a penalty parameter. It is known that if μ is sufficiently large, the iterates converge exponentially fast to the solution of (13). More formally, the following result is known for the convergence of the Uzawa method from [15, 21]:

Lemma 1. *Let (x^0, λ^0) be a given initial guess, and for $\ell \geq 1$ let (x^ℓ, λ^ℓ) be the iterates produced by the augmented Lagrangian Uzawa method. Denote by γ_0 the smallest eigenvalue of $\mathbf{F}^\top\mathbf{D}^{-1}\mathbf{F}$. Then the following holds:*

$$\|\lambda - \lambda^\ell\| \leq \left(\frac{1}{1+\gamma_0\mu}\right)^\ell \|\lambda - \lambda^0\|,$$

$$\|x - x^\ell\| \leq \sqrt{1/\mu}\,\|\lambda - \lambda^{\ell-1}\| \leq \sqrt{1/\mu}\left(\frac{1}{1+\gamma_0\mu}\right)^\ell \|\lambda - \lambda^0\|.$$

Lemma 1 implies that the Uzawa method converges exponentially fast for a sufficiently large value of μ. Since the factor $\sqrt{1/\mu}(1+\gamma_0\mu)^{-1}$ decreases monotonically with μ, even a single iteration with $\mu \gg 1$ can yield a good approximate solution. In other words, by applying the Uzawa method to (13), for a given $(\boldsymbol{x}^0, \lambda^0)$, we can obtain $(\boldsymbol{x}^1, \lambda^1)$ in the first iteration as follows:

$$\begin{aligned} (\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)\boldsymbol{x}^1 &= \boldsymbol{b} - \mathbf{F}\lambda^0, \\ \lambda^1 &= \lambda^0 + \mu\mathbf{F}^\top \boldsymbol{x}^1, \end{aligned}$$

Setting $\lambda^0 = 0$ yields

$$(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)\boldsymbol{x}^1 = \boldsymbol{b}. \tag{14}$$

By Lemma 1, we can safely use $\boldsymbol{x}^1$ in (14) as an approximate solution to (13). That is, we have the following solution to (13):

$$\boldsymbol{x} \approx (\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1}\boldsymbol{b}. \tag{15}$$

Thus, by noting that we have dropped the superscript t, and since $\boldsymbol{b} = \mathbf{W}\boldsymbol{x}^{t-1}$, the test vector $\boldsymbol{x}^t$ at iteration t is now obtained by

$$\boldsymbol{x}^t = (\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1}\mathbf{W}\boldsymbol{x}^{t-1}. \tag{16}$$

The process of imposing the fairness constraints into every Jacobi relaxation iteration to obtain each test vector $\boldsymbol{x}_r$ is summarized in Algorithm 1. Once we obtain R test vectors, we compute the algebraic distance $s(i, j)$ for each pair of nodes i and j as in (8) and construct a new affinity matrix $\mathbf{W}_{\text{alg}}$ as in (9).

3.2 Fair Graph Clustering via Algebraic Distance

From the new affinity matrix $\mathbf{W}_{\text{alg}}$, which now reflects the fairness constraints, the next step of FairAD is to partition the nodes V into k clusters. To this end, we leverage 'graph coarsening' to coarsen the (updated) graph with $\mathbf{W}_{\text{alg}}$ in order to identify a small number of representative corresponding to k clusters. They are then used as anchor nodes to guide the final clustering process. Specifically, we finally solve a constrained minimization problem, which is a relaxed k-way graph cut problem with having the representative nodes as additional linear constraints.

Graph coarsening. For graph coarsening, we use a coarsening algorithm introduced in [25]. It is a recursive algorithm, which starts from the finest level and moves towards increasingly coarser levels. Let $G^{(\ell)} = (V^{(\ell)}, E^{(\ell)})$ be the coarse graph at level ℓ, and let $\mathbf{W}_\ell = (W^{(\ell)}_{i,j})$ be its corresponding affinity (weight) matrix, where $\ell = 0, 1, \ldots, \kappa$. We set $G_0 := G$ and $\mathbf{W}_0 := \mathbf{W}_{\text{alg}}$. That is, the finest graph is the graph G with $\mathbf{W}_{\text{alg}}$. Also, $G^{(\kappa)}$ and $\mathbf{W}_\kappa$ are the coarsest graph and its affinity matrix, respectively. We below explain how $G^{(\ell)}$ and $\mathbf{W}_\ell$ are updated at each level ℓ.

Algorithm 2: Coarsening

Input : $\mathbf{W}_{\text{alg}}$, # of coarse levels κ
Output: Coarse graphs $\{G_\ell\}_{\ell=1}^{\kappa}$ with $\{\mathbf{W}_\ell\}_{\ell=1}^{\kappa}$

$\mathbf{W}_0 \coloneqq \mathbf{W}_{\text{alg}}$, $V^{(0)} \coloneqq V$.
for *each coarse level* $\ell = 1, 2, \ldots, \kappa$ **do**
  $\eta \coloneqq |V^{(\ell-1)}|$.
  $V^{(\ell)} \coloneqq \{n_1\}$.
  for $i = 2, 3, \ldots, \eta$ **do**
    if $\max_{j \in V^{(\ell)}} W_{n_i,j}^{(\ell-1)} \leq \alpha \sum_{j' \in V^{(\ell-1)}} W_{n_i,j'}^{(\ell-1)}$ **then**
      $V^{(\ell)} \coloneqq V^{(\ell)} \cup \{n_i\}$.
    end if
  end for
  Compute $\mathbf{P}_\ell$ by (18).
  $\mathbf{W}_\ell \coloneqq \mathbf{P}_\ell^\top \mathbf{W}_{\ell-1} \mathbf{P}_\ell$.
end for

The coarsening algorithm begins by initializing $V^{(\ell)}$ as a singleton set containing only the first node, say, $n_1 \in V^{(\ell-1)}$. That is, $V^{(\ell)} \coloneqq \{n_1\}$. We then repeatedly check if the next node $n_i \in V^{(\ell-1)}$ is 'weakly' connected to the ones that have been added in $V^{(\ell)}$ by evaluating the following inequality:

$$\max_{j \in V^{(\ell)}} W_{n_i,j}^{(\ell-1)} \leq \alpha \sum_{j' \in V^{(\ell-1)}} W_{n_i,j'}^{(\ell-1)}, \tag{17}$$

where α is the coarsening parameter. The value of α is generally chosen to be much smaller than one, i.e., $\alpha \ll 1$, and our choice of the value shall be explained later in the experiments. If the inequality holds, it implies that node n_i does not have a strong connection with any of the previously added nodes in $V^{(\ell)}$. As a result, node n_i is treated as a 'sufficiently independent' node and added to $V^{(\ell)}$, i.e., $V^{(\ell)} \coloneqq V^{(\ell)} \cup \{n_i\}$. Otherwise, $V^{(\ell)}$ remains unchanged. This coarsening process is repeated until all the nodes $V^{(\ell-1)}$ are evaluated. Note that the intuition behind this process is to eliminate *mutually strongly connected* nodes at each level.

Once the coarsening process is complete at level ℓ, we construct a $|V^{(\ell-1)}| \times |V^{(\ell)}|$ interpolation matrix $\mathbf{P}_\ell \coloneqq (P_{i,j}^\ell)$, with elements $P_{i,j}^\ell$ given by

$$P_{i,j}^\ell \coloneqq \begin{cases} \dfrac{W_{i,j}^{(\ell-1)}}{\sum_{j' \in V^{(\ell)}} W_{i,j'}^{(\ell-1)}}, & \text{for } i \in V^{(\ell-1)}, j \in V^{(\ell)}, \\ 1, & \text{for } i \in V^{(\ell)}, i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

This interpolation matrix is then used to obtain the affinity matrix $\mathbf{W}_\ell$ at level ℓ as follows:

$$\mathbf{W}_\ell = \mathbf{P}_\ell^T \mathbf{W}_{\ell-1} \mathbf{P}_\ell.$$

The entire graph coarsening process is repeated recursively, level by level. It is summarized in Algorithm 2.

Constrained minimization. The primary objective of graph coarsening is to identify k representative nodes corresponding to k distinct clusters. Intuitively, they are *most weakly* connected to each other, so they could serve as anchor nodes (i.e., each node represents each separate cluster) for the final clustering process. However, a drawback of the graph coarsening algorithm is that we cannot control the exact number of nodes generated at the coarsest level ($\ell = \kappa$). Thus, we instead identify at least $m > k$ representative nodes that correspond to k clusters, where multiple nodes can correspond to the same cluster. Once we obtain coarse graphs $\{G_\ell\}_{\ell=1}^{\kappa}$ from the coarsening algorithm, we find the smallest coarse graph containing at least m nodes. Note that for a given value of k, we set the value of m to be a bit greater than the value of k. For example, we use the value of m between 15 and 50 for $k < 10$.

Algorithm 3: FairAD

Input : $\mathbf{W}$, $\mathbf{D}$, $\mathbf{F}$, m
Output: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$

1 **for** $r = 1$ **to** R **do**
2 $\boldsymbol{x}_r \leftarrow$ cJacobi($\mathbf{W}, \mathbf{D}, \mathbf{F}$).
3 **end for**
4 Compute $\mathbf{W}_{\text{alg}}$ using (9) with $\{\boldsymbol{x}_r\}_{r=1}^{R}$.
5 $\{G_\ell\}_{\ell=0}^{\kappa} \leftarrow$ Coarsening($\mathbf{W}_{\text{alg}}$).
6 **for** *each coarse level* $\ell = \kappa, \kappa-1, \ldots, 0$ **do**
7 **if** $|V^{(\ell)}| \geq m$ **then**
8 $\mathbf{B}, \mathbf{c} \leftarrow$ SpectralClustering(G_ℓ, k).
9 break;
10 **end if**
11 **end for**
12 $\overline{\mathbf{L}}_{\text{alg}} \coloneqq \mathbf{D}_{\text{alg}}^{-1/2}(\mathbf{D}_{\text{alg}} - \mathbf{W}_{\text{alg}})\mathbf{D}_{\text{alg}}^{-1/2}$.
13 $\mathbf{A}_{\text{alg}} \coloneqq \overline{\mathbf{L}}_{\text{alg}} + \mu \mathbf{B}^\top \mathbf{B}$.
14 **for** *each cluster* $i = 1, 2, \ldots, k$ **do**
15 $\boldsymbol{v}_i \coloneqq \mu \mathbf{A}_{\text{alg}}^{-1} \mathbf{B}^\top \mathbf{c}_i$.
16 **end for**

Specifically, we move towards the finest graph, starting from the coarsest one, and find the first coarse graph containing at least m nodes (Lines 6-7 of Algorithm 3). We then apply spectral clustering to this coarse graph to obtain k groups of representative nodes (Line 8 of Algorithm 3). In other words, we first compute the first k eigenvectors of the $m \times m$ normalized Laplacian matrix of the coarse graph, which form a $m \times k$ matrix, and then run k-means on its rows to produce k groups of representative nodes. Note that since the coarse graph (with m nodes) is significantly smaller than the original graph (with n nodes), it is not a computational burden to use the spectral clustering at this stage.

Let m^* be the number of identified representative nodes. Our next step is to find the clustering solution such that m^* representative nodes belong to their corresponding clusters. In other words, we impose the representative nodes with their corresponding groups as linear constraints to a relaxed graph cut problem. This problem is given by, for $i = 1, 2, \ldots, k$,

$$\min_{\mathbf{B}\boldsymbol{v}_i = \mathbf{c}_i} \frac{1}{2} \boldsymbol{v}_i^\top \overline{\mathbf{L}}_{\text{alg}} \boldsymbol{v}_i, \tag{19}$$

where $\overline{\mathbf{L}}_{\text{alg}}$ is the normalized Laplacian corresponding to $\mathbf{W}_{\text{alg}}$. From m^* representative nodes, say, $r_1, r_2, \ldots, r_{m^*}$, we define $\mathbf{B}$ and $\mathbf{c}_i$ to be

$$\mathbf{B} = \begin{pmatrix} \mathbf{e}_{r_1}^\top \\ \mathbf{e}_{r_2}^\top \\ \vdots \\ \mathbf{e}_{r_{m^*}}^\top \end{pmatrix} \quad \text{and} \quad \mathbf{c}_i = \begin{pmatrix} \delta_{r_1,i} \\ \delta_{r_2,i} \\ \vdots \\ \delta_{r_{m^*},i} \end{pmatrix}, \tag{20}$$

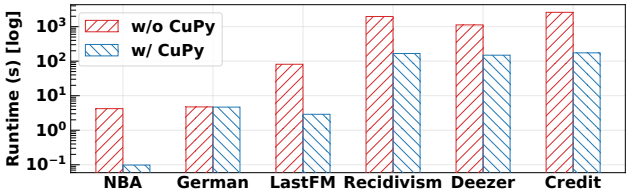


Figure 2: Running time of FairAD with and without CuPy.

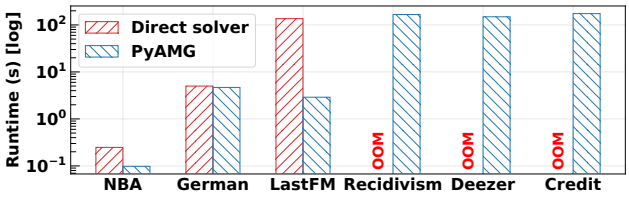


Figure 3: Running time of FairAD with and without PyAMG.

respectively, where $\mathbf{e}_i$ is an n-dimensional vector with the i-th entry being one and the others being zero, and $\delta_{r,i} = 1$ if node r belongs to cluster i and 0 otherwise. That is, each row of $\mathbf{B}$ is a one-hot vector that represents the location of its corresponding representative node, and $\mathbf{c}_i$ is a vector to indicate which representative nodes belong to cluster i.

We can employ the same technique as used in solving (12) to solve (19). First, we write its equivalent indefinite system as

$$\begin{pmatrix} \bar{\mathbf{L}}_{\text{alg}} & \mathbf{B}^\top \\ \mathbf{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_i \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{c}_i \end{pmatrix}. \tag{21}$$

Then, as was done in Section 3.1, we leverage the Uzawa method to solve the system in (21). For a given $(\boldsymbol{v}_i^0, \lambda^0)$, we have $(\boldsymbol{v}_i^1, \lambda^1)$ in the first iteration as follows:

$$(\bar{\mathbf{L}}_{\text{alg}} + \mu\mathbf{B}^\top\mathbf{B})\boldsymbol{v}_i^1 + \mathbf{B}\lambda^0 = \mu\mathbf{B}^\top\mathbf{c}_i$$
$$\lambda^1 = \lambda^0 + \mu(\mathbf{B}\boldsymbol{v}_i^1 - \mathbf{c}_i).$$

Setting $\lambda^0 = 0$ yields

$$(\bar{\mathbf{L}}_{\text{alg}} + \mu\mathbf{B}^\top\mathbf{B})\boldsymbol{v}_i^1 = \mu\mathbf{B}^\top\mathbf{c}_i.$$

By Lemma 1, we obtain the following approximate solution to (21):

$$\boldsymbol{v}_i \approx \mu\mathbf{A}_{\text{alg}}^{-1}\mathbf{B}^\top\mathbf{c}_i, \tag{22}$$

where $\mathbf{A}_{\text{alg}} := \bar{\mathbf{L}}_{\text{alg}} + \mu\mathbf{B}^\top\mathbf{B}$. Finally, once we have the solutions $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, the cluster label of node j is determined by identifying which $\boldsymbol{v}_i$ has the maximum value in its j-th entry, i.e., $\arg\max_i v_{i,j}$ for $j \in [n]$.

3.3 Optimized Implementation

We here explain a set of implementation optimizations used for the implementation of FairAD, which is summarized in Algorithm 3. First, recall that obtaining the test vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_R$ to construct a new affinity matrix $\mathbf{W}_{\text{alg}}$ requires computing (16) iteratively (or running Algorithm 1), which involves inverting the matrix $(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)$. To efficiently compute $(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1}$, we employ the Woodbury matrix identity [33]. For given matrices $\mathbf{A}, \mathbf{C}, \mathbf{U}$, and $\mathbf{V}$ with the shapes $n \times n, k \times k, n \times k$ and $k \times n$, respectively, the Woodbury matrix identity to compute $(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1}$ is given by

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1}.$$

Using this identity, we can efficiently compute $(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1}$ as follows:

$$(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{F}(\mathbf{I}_\mu^{-1} + \mathbf{F}^\top\mathbf{D}^{-1}\mathbf{F})^{-1}\mathbf{F}^\top\mathbf{D}^{-1},$$

where $\mathbf{I}_\mu := \mu\mathbf{I}$. Here, since $\mathbf{D}$ is a diagonal matrix, it is straightforward to compute $\mathbf{D}^{-1}$. In addition, $\mathbf{F}$ is a tall matrix, meaning that the inversion $(\mathbf{I}_\mu^{-1} + \mathbf{F}^\top\mathbf{D}^{-1}\mathbf{F})^{-1}$ is also easier to compute. This is because it only involves inverting a $k \times k$ matrix, where k is typically much smaller than n. That is, the inversion costs $O(k^3)$ instead of $O(n^3)$.

Second, we optimize the implementation of FairAD by leveraging CuPy, which is an open source library for GPU-accelerated computing with Python.[1] In particular, the matrix operations are done much more efficiently since CuPy allows us to fully exploit GPU parallel processing. As a result, the computational efficiency of FairAD can be greatly enhanced. In Figure 2, we compare the running times of our GPU-accelerated implementation of FairAD with CuPy and its CPU-based counterpart. The GPU-accelerated implementation achieves at least an order-of-magnitude speed-up on five of the six datasets. Specifically, we observe a nearly two orders-of-magnitude improvement on NBA and LastFM graphs and roughly an order-of-magnitude improvement on the larger graphs such as Recidivism, Deezer, and Credit.

Third, we optimize the efficiency of the graph coarsening algorithm, which is an integral part of FairAD, by judiciously choosing the order of nodes in $V^{(\ell-1)}$ to be evaluated, as in Line 5 of Algorithm 2. Instead of simply evaluating the nodes in increasing order of their node IDs, we prioritize the nodes that are strongly connected with others. To this end, we maintain a set of 'volumes', denoted by $\boldsymbol{\nu}$. It is initially $\boldsymbol{\nu} := \mathbf{1}$ and updated by $\boldsymbol{\nu} := \boldsymbol{\nu}\mathbf{P}_\ell$. The nodes in $V^{(\ell-1)}$ are then evaluated in descending order of their volumes. This way, we can first evaluate more important nodes from a network connectivity perspective at each level.

Finally, to identify the cluster membership of each node in the end, i.e., which cluster each node belongs to, we need to compute the solution $\boldsymbol{v}_i$ as in (22), which is equivalent to solving the following linear system: For $i = 1, 2, \ldots, k$,

$$\mathbf{A}_{\text{alg}}\boldsymbol{v}_i = \mu\mathbf{B}^\top\mathbf{c}_i.$$

Note that unlike computing $(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)^{-1}$, it is impractical to directly compute $\mathbf{A}_{\text{alg}}^{-1}$, since $\mathbf{A}_{\text{alg}}$ does not possess the nice property of $(\mathbf{D} + \mu\mathbf{F}\mathbf{F}^\top)$ that allows us to leverage the Woodbury matrix identity. One could solve the linear system using the standard solvers from NumPy or SciPy's linear algebra packages, but they become inefficient for large graphs. Thus, we utilize PyAMG's classical AMG solver [5], which is well-suited for large, sparse systems.[2] In Figure 3, we compare the running times of the implementation of FairAD when using the PyAMG solver and a direct SciPy solver. For the small graphs such as NBA and LastFM, the implementation of FairAD with PyAMG outperforms the one with the direct solver by up to an order of magnitude. For the larger graphs such as Recidivism, Deezer, and Credit, the implementation with PyAMG completes in roughly 150 seconds, while the one with the direct solver fails with out-of-memory errors ("OOM"). Thus, we can see

[1] https://docs.cupy.dev/en/v13.2.0/index.html
[2] https://pyamg.readthedocs.io/en/latest/

Table 1: Statistics of datasets used in the experiments

Dataset	|V|	|E|	Sensitive Attribute	h
NBA	403	10,621	Country	2
German	1,000	21,742	Gender	2
LastFM	7,624	27,806	Country	4
Recidivism	18,876	311,870	Race	2
Deezer	28,281	92,752	Gender	2
Credit	29,460	136,196	Education	3

that the implementation of FairAD based on the PyAMG solver is more computationally efficient and scalable to large graphs.

4 Experiments

In this section, we provide extensive experiment results to demonstrate the superior performance of FairAD to baseline algorithms, which are state-of-the-art fair clustering algorithms such as FairSC and sFairSC as well as the plain spectral clustering (SC).

Datasets. We consider both synthetic and public real-world datasets for performance evaluation. The synthetic dataset is generated based on a modified stochastic block model (mSBM) [18], which has been widely used to generate synthetic networks for clustering and community detection. Suppose that the set of nodes V consists of h groups, i.e., $V = V_1 \cup V_2 \cup \cdots \cup V_h$, and is also partitioned into k ground-truth clusters, i.e., $V = C_1 \cup C_2 \cup \cdots \cup C_k$. The synthetic dataset is generated such that the 'fairness' condition is satisfied, meaning that the almost same proportion of nodes from each group V_s appears in each cluster C_l. In other words, we ensure the proportion $\eta_s := |V_s \cap C_l|/|C_l|$ to be more or less the same for all $s \in [h]$, i.e., $\eta_1 \approx \eta_2 \approx \ldots \approx \eta_h$, for each $l \in [k]$.

For mSBM, the probability of having an edge between two nodes depends on their membership in the groups and clusters. Thus, we define the probability of having an edge between nodes i and j, say, $Q_{i,j}$, to be given by

$$Q_{i,j} := \begin{cases} a, & \text{if } i \text{ and } j \text{ are in the same group and the same cluster,} \\ b, & \text{if } i \text{ and } j \text{ are in the same group and different clusters,} \\ c, & \text{if } i \text{ and } j \text{ are in different groups and the same cluster,} \\ d, & \text{if } i \text{ and } j \text{ are in different groups and different clusters,} \end{cases}$$

with $a > b > c > d$, which is to reflect stronger connections within groups and clusters [18, 30]. Then, the affinity matrix of mSBM is obtained by

$$W_{i,j} = \begin{cases} \text{Bernoulli}(Q_{i,j}), & \text{if } i \neq j \\ 0, & \text{otherwise,} \end{cases}$$

where $\text{Bernoulli}(Q_{i,j})$ is a Bernoulli random variable with probability $Q_{i,j}$.

In addition, we consider six real-world datasets, namely, NBA [9], German [3], Recidivism [16], LastFM [23], Deezer [23], and Credit [36], which are all from social networks and contain sensitive attributes. The statistics of the datasets are provided in Table 1. We use the largest connected component of each graph.

Parameter settings. In our experiments, we set the parameters of mSBM as follows: $a = 10(\log n/n)^{2/3}$, $b = 7(\log n/n)^{2/3}$, $c = 4(\log n/n)^{2/3}$, and $d = (\log n/n)^{2/3}$. For FairAD, we set $\mu = 10^9$ and $\alpha = 10^{-4}$. Both the number of test vectors, R, and the number of Jacobi iterations, τ, are set to 10. We observed that the test vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_R$ often become indistinguishable due to floating point precision limitations. To address this issue, we introduce a scaling parameter β to the definition of $W_{i,j}^{\text{alg}}$ in (9), which is now given by $W_{i,j}^{\text{alg}} = \exp(-\beta s(i,j))$, where the value of β is chosen to be sufficiently large. This way, we ensure that the difference between the scaled algebraic distances of different node pairs becomes much bigger, thereby leading to informative test vectors. Specifically, we set $\beta = n/\log(n)$ in the experiments.

4.1 Experiment Setup

Evaluation metrics. We use the error rate introduced in [18] to quantify the performance of FairAD and baselines on the synthetic networks generated by mSBM. Let $\mathbf{V} = (V_{i,j})$ be the indicator matrix of the predicted clustering labels, where $V_{i,j} = 1$, if node i belongs to cluster j, and 0 otherwise. Similarly, let $\mathbf{V}^*$ be the indicator matrix for the ground-truth labels generated by the mSBM. Note that the numeric labels $1, 2, \ldots, k$ are arbitrary, so the same partitions may differ by a permutation of columns. In other words, there exists a permutation matrix $\mathbf{U}$ such that $\mathbf{VU} = \mathbf{V}^*$. To account for this nature, letting $\mathbf{\Pi}_k$ be the set of all $k \times k$ permutation matrices, the error rate is defined as the smallest difference between the permuted prediction $\mathbf{VU}$ and the ground truth $\mathbf{V}^*$, which is given by

$$E(\mathbf{V} - \mathbf{V}^*) = \frac{1}{k} \min_{\mathbf{U} \in \mathbf{\Pi}_k} \|\mathbf{VU} - \mathbf{V}^*\|. \tag{23}$$

For real-world datasets, we use the average balance introduced in [7] as the performance metric. Specifically, for a given group partition $V = V_1 \cup \cdots \cup V_h$, having a partition of clusters $V = C_1 \cup \cdots \cup C_k$, we define the balance of cluster C_l as follows: For $l = 1, 2, \ldots, k$,

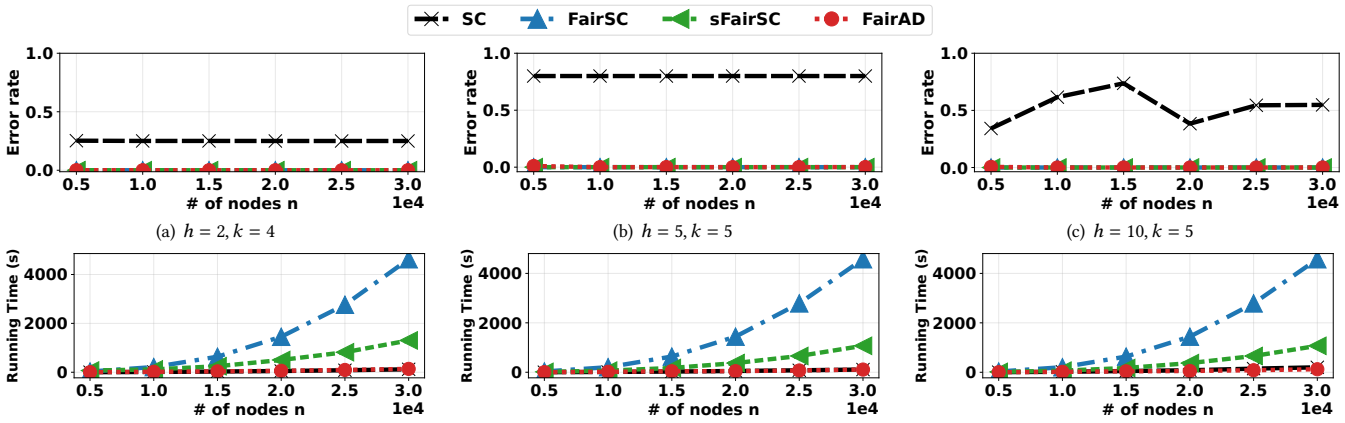
$$\text{balance}(C_l) = \min_{s,s' \in [h], s \neq s'} \frac{|V_s \cap C_l|}{|V_{s'} \cap C_l|}, \tag{24}$$

where $|V_s \cap C_l|$ is the number of the members of group V_s that also appear in cluster C_l. This metric measures the degree of the discrepancy in the proportional representation of each group V_s in cluster C_l by taking the minimum ratio across all pairs of groups. In other words, the balance of C_l is determined by the largest difference in the proportional representation of each group V_s in cluster C_l, i.e., the largest difference between $|V_s \cap C_l|$ and $|V_{s'} \cap C_l|$ among all pairs of groups. A cluster achieves perfect balance (value 1) only if all groups are equally represented in the cluster, while a lower balance value indicates that at least one group pair is unbalanced. The average balance is then obtained by taking the average of the balance values over all k clusters.

Hardware and software configuration. For hardware, all experiments are conducted on a Linux server equipped with an Intel Xeon Gold 5218R CPU, 95GB RAM, and an NVIDIA Quadro RTX 8000 48GB GPU. For software, we use Python 3.12, SciPy v1.11, CuPy v13.2, and CUDA 12.3. Each experiment is repeated ten times, and the average results are reported for performance evaluation.

4.2 Experiment Results

Synthetic dataset. We first consider the mSBM dataset to demonstrate the effectiveness and scalability of FairAD in comparison



(a) $h = 2, k = 4$ (b) $h = 5, k = 5$ (c) $h = 10, k = 5$

(d) $h = 2, k = 4$ (e) $h = 5, k = 5$ (f) $h = 10, k = 5$

Figure 4: Error rate in (23), shown in the top row, and running time (in seconds), shown in the bottom row, under synthetic networks generated by mSBM with varying values of h and k.

Table 2: Running times (in seconds) of SC, FairSC, sFairSC, and FairAD on mSBM with $n = 20000$

	$k=3$					$k=4$					$k=5$					$k=6$				
	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$
SC	44	34	55	53	53	32	22	26	46	47	53	35	46	55	58	79	58	44	39	53
FairSC	1337	1215	1249	1329	1357	1304	1325	1361	1213	1196	1137	1253	1143	1156	1243	1248	1313	1337	1189	1179
sFairSC	546	512	535	636	457	444	499	466	450	526	578	534	698	437	567	642	603	654	433	603
FairAD	57	52	51	52	52	61	60	56	58	55	65	63	64	62	63	81	66	69	63	71

Table 3: Running times (in seconds) of SC, FairSC, sFairSC, and FairAD on mSBM with $n = 30000$

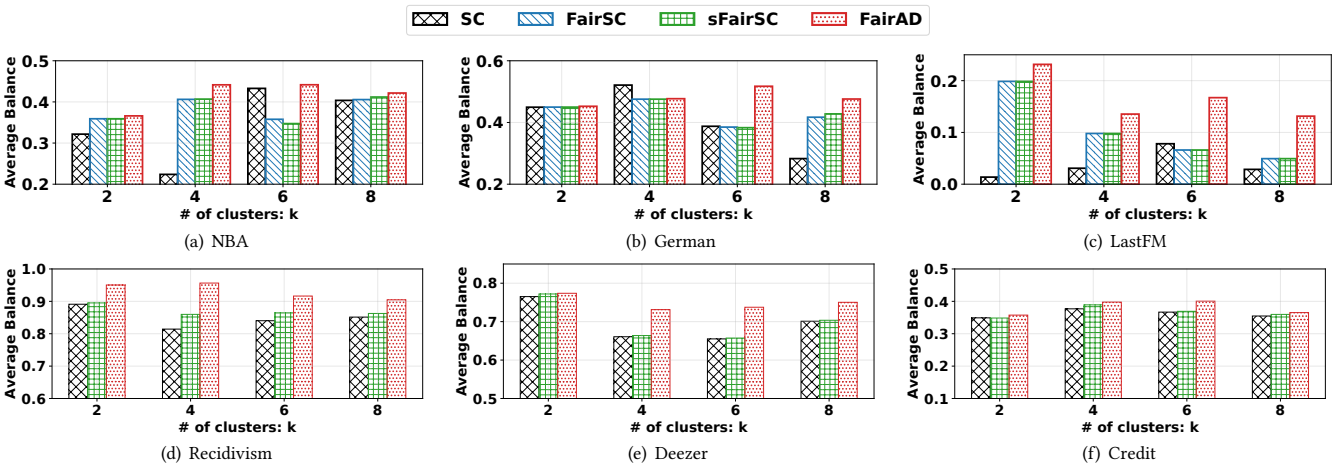
	$k=3$					$k=4$					$k=5$					$k=6$				
	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$	$h=2$	$h=4$	$h=6$	$h=8$	$h=10$
SC	73	78	86	111	97	111	72	84	111	87	89	89	78	107	122	82	82	94	107	122
FairSC	4502	4287	4383	4285	4472	4574	4523	4567	4502	4421	4439	4553	4494	4529	4519	4510	4332	4453	4339	4379
sFairSC	1008	1036	1003	963	1001	1032	1006	1007	962	1044	1099	1157	1063	1076	981	890	862	804	883	789
FairAD	122	116	114	112	114	149	129	133	132	140	140	149	134	130	131	161	149	146	148	158

with the baseline algorithms in terms of error rate and running time. As illustrative results, in Figure 4, we present the results for the cases with $h = 2$ and $k = 4$, $h = 5$ and $k = 5$, and $h = 10$ and $k = 5$. Here, the graph sizes vary from $n = 5 \times 10^3$ to $n = 3 \times 10^4$. As shown in Figure 4, we observe that FairSC, sFairSC, and FairAD accurately obtain the ground truth clustering labels, whereas SC fails with a high error rate. While FairSC, sFairSC, and FairAD do the job correctly, FairAD is significantly faster than FairSC and sFairSC in terms of running time, with the speed-ups of up to 42× and 12×, respectively.

We further evaluate the impact of varying values of h and k on the performance of FairAD and baselines when the graph sizes are $n = 20000$ and $n = 30000$. We observe that the error rates of FairSC, sFairSC, and FairAD are all zero for most cases, whereas SC's error rate ranges from 0.17 to 0.83. We omit the results for brevity. We here mainly evaluate the running time of each method. As shown in Tables 2 and 3, the running time of FairAD is just under 200 seconds, achieving over 10× speed-up compared to sFairSC and 30× to 40× speed-up compared to FairSC for all test cases. While SC achieves the similar running time as FairAD, it has a much higher error rate. These results demonstrate the superior performance of FairAD across different settings.

Real-world datasets. We next evaluate the effectiveness and efficiency of FairAD on six real-world datasets in terms of the average balance and running time and demonstrate its superiority to the baselines. As shown in Figure 5, we observe that FairAD outperforms all baselines consistently in terms of average balance. Specifically, for the small graphs such as NBA, German, and LastFM, FairAD achieves an average improvement of approximately 20%, with up to about 100% improvement, when compared to FairSC and sFairSC. Note that FairSC and sFairSC exhibit nearly identical balance performance for all graphs. We also observe that FairSC takes an excessive amount of time, i.e., several hours to days, for the larger graphs such as Recidivism, Deezer, and Credit, for which we do not report its results. For the larger graphs, FairAD outperforms sFairSC by 10% to 15%. Overall, the results demonstrate the effectiveness of FairAD, which consistently achieves the highest average-balance score for all test cases on all datasets. In other words, FairAD produces the most balanced clusters for all cases.

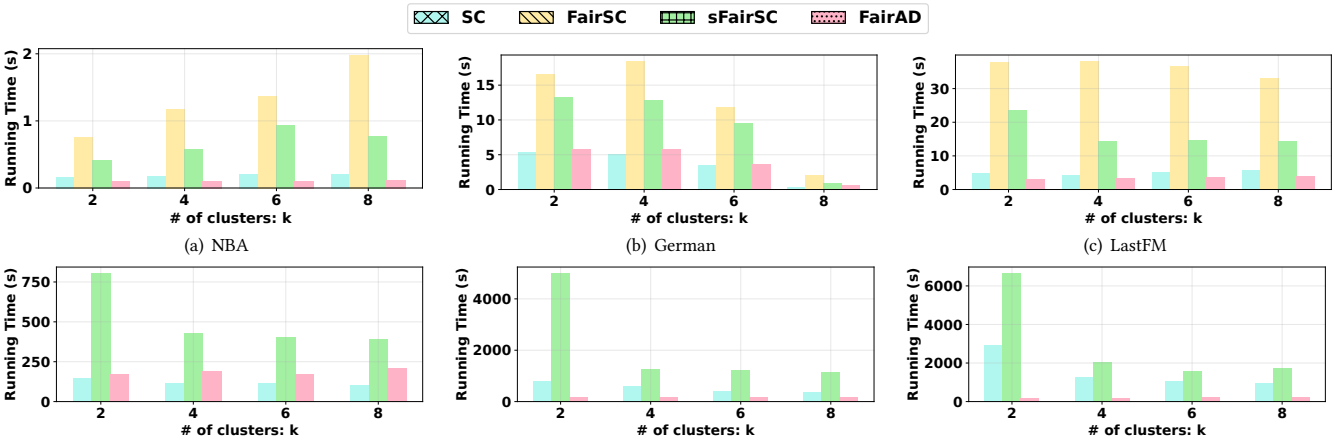
To evaluate the efficiency of FairAD, we compare its running time with that of each baseline and report the results in Figure 6. FairAD is at least twice as fast as sFairSC and more than 3× faster than FairSC for all values of k on the small graphs such as NBA, German, and LastFM. The performance gap in running time becomes wider



(a) NBA (b) German (c) LastFM

(d) Recidivism (e) Deezer (f) Credit

Figure 5: Average balance for NBA, German, and LastFM datasets (top row) and for Recidivism, Deezer, and Credit datasets (bottom row), when changing the number of clusters.



(a) NBA (b) German (c) LastFM

(d) Recidivism (e) Deezer (f) Credit

Figure 6: Running time for NBA, German, and LastFM datasets (top row) and for Recidivism, Deezer, and Credit datasets (bottom row), when changing the number of clusters.

on the larger graphs, as shown in Figures 6(d)–6(f). FairAD is more than an order of magnitude faster than sFairSC, achieving up to 40× speed-up compared to sFairSC. This is because sFairSC, especially its eigensolver, requires a much larger number of iterations to converge for such large graphs. We also observe that FairAD is even faster than SC for Deezer and Credit datasets, where SC already fails to produce balanced clusters.

To summarize, the experiments on both synthetic and real-world datasets confirm the effectiveness and efficiency of FairAD. It produces highly balanced clusters while taking only a fraction of the running times of its competing methods, namely FairSC and sFairSC, making it a practical solution for fair graph clustering.

5 Conclusion

We have introduced FairAD, a novel fair graph clustering method via algebraic distance. The main enabler of FairAD is its framework to impose fairness constraints into the affinity matrix when it is constructed based on the algebraic distance. FairAD then effectively leveraged graph coarsening to convert the optimization problem into a simpler graph cut problem, which is solved efficiently. Its implementation was further optimized through several techniques. Experiment results demonstrated the superior performance of FairAD to state-of-the-art fair graph clustering algorithms in terms of both the quality of fairness and computational efficiency.

Acknowledgments

This work was supported by the National Science Foundation under grants IIS-2209921, CNS-2209922, and DMS-2208499, the International Energy Joint R&D Program of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade, Industry & Energy, Republic of Korea (No. 20228530050030), and an equipment donation from NVIDIA Corporation. C. Lee is the corresponding author.

GenAI Usage Disclosure

No GenAI tools were used in any stage of the research, nor in the writing.

References

[1] Matteo Almanza, Alessandro Epasto, Alessandro Panconesi, and Giuseppe Re. 2022. k-Clustering with fair outliers. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 5–15.

[2] Daichi Amagata. 2024. Fair k-center clustering with outliers. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*. PMLR, 10–18.

[3] Arthur Asuncion, David Newman, et al. 2007. UCI machine learning repository.

[4] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. 2019. Scalable fair clustering. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 405–413.

[5] Nathan Bell, Luke N. Olson, Jacob Schroder, and Ben Southworth. 2023. PyAMG: Algebraic Multigrid Solvers in Python. *Journal of Open Source Software* 8, 87 (2023), 5495.

[6] Brian Brubach, Darshan Chakrabarti, John P. Dickerson, Aravind Srinivasan, and Leonidas Tsepenekas. 2021. Fairness, semi-supervised learning, and more: A general framework for clustering with stochastic pairwise constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 6822–6830.

[7] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. 2017. Fair clustering through fairlets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 5036–5044.

[8] Mihai Cucuringu, Ioannis Koutis, Sanjay Chawla, Gary Miller, and Richard Peng. 2016. Simple and scalable constrained clustering: a generalized spectral method. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. PMLR, 445–454.

[9] Enyan Dai and Suhang Wang. 2021. Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 680–688.

[10] Yushun Dong, Ninghao Liu, Brian Jalaian, and Jundong Li. 2022. Edits: Modeling and mitigating data bias for graph neural networks. In *Proceedings of the ACM Web Conference 2022*. 1259–1269.

[11] Xin Du, Yulong Pei, Wouter Duivesteijn, and Mykola Pechenizkiy. 2020. Fairness in network representation by latent structural heterogeneity in observational data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3809–3816.

[12] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 214–226.

[13] Wei Fan, Kunpeng Liu, Rui Xie, Hao Liu, Hui Xiong, and Yanjie Fu. 2021. Fair Graph Auto-Encoder for Unbiased Graph Representations with Wasserstein Distance. In *2021 IEEE International Conference on Data Mining (ICDM)*. 1054–1059.

[14] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 259–268.

[15] Michel Fortin and Roland Glowinski. 2000. *Augmented Lagrangian methods: Applications to the numerical solution of boundary-value problems*. Elsevier.

[16] Kareem L Jordan and Tina L Freiburger. 2015. The effect of race/ethnicity on sentencing: Examining sentence type, jail length, and prison length. *Journal of Ethnicity in Criminal Justice* 13, 3 (2015), 179–196.

[17] Christopher Jung, Sampath Kannan, and Neil Lutz. 2020. Service in Your Neighborhood: Fairness in Center Location. In *1st Symposium on Foundations of Responsible Computing*, Vol. 156. 5:1–5:15.

[18] Matthäus Kleindessner, Samira Samadi, Pranjal Awasthi, and Jamie Morgenstern. 2019. Guarantees for spectral clustering with fairness constraints. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 3458–3467.

[19] Matt Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 4069–4079.

[20] Mariá C.V. Nascimento and André C.P.L.F. de Carvalho. 2011. Spectral methods for graph clustering–a survey. *European Journal of Operational Research* 211, 2 (2011), 221–231.

[21] Iván Ojeda-Ruiz and Young-Ju Lee. 2020. A fast constrained image segmentation algorithm. *Results in Applied Mathematics* 8 (2020), 100103.

[22] Dorit Ron, Ilya Safro, and Achi Brandt. 2011. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation* 9, 1 (2011), 407–423.

[23] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. 1325–1334.

[24] Yousef Saad. 2003. *Iterative methods for sparse linear systems*. SIAM.

[25] Eitan Sharon, Meirav Galun, Dahlia Sharon, Ronen Basri, and Achi Brandt. 2006. Hierarchy and adaptivity in segmenting visual scenes. *Nature* 442, 7104 (2006), 810–813.

[26] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 8 (2000), 888–905.

[27] Ali Vakilian and Mustafa Yalciner. 2022. Improved approximation algorithms for individually fair clustering. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. PMLR, 8758–8779.

[28] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17 (2007), 395–416.

[29] Dorothea Wagner and Frank Wagner. 1993. Between min cut and graph bisection. In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*. Springer, 744–750.

[30] Ji Wang, Ding Lu, Ian Davidson, and Zhaojun Bai. 2023. Scalable spectral clustering with group fairness constraints. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*. PMLR, 6613–6629.

[31] Nan Wang, Lu Lin, Jundong Li, and Hongning Wang. 2022. Unbiased graph embedding with biased graph observations. In *Proceedings of the ACM Web Conference 2022*. 1423–1433.

[32] Xiang Wang, Buyue Qian, and Ian Davidson. 2014. On constrained spectral clustering and its applications. *Data Mining and Knowledge Discovery* 28 (2014), 1–30.

[33] M.A. Woodbury. 1950. *Inverting Modified Matrices*. Department of Statistics, Princeton University.

[34] Ruicheng Xian, Lang Yin, and Han Zhao. 2023. Fair and optimal classification via post-processing. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 37977–38012.

[35] Linli Xu, Wenye Li, and Dale Schuurmans. 2009. Fast normalized cut with linear constraints. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2866–2873.

[36] I-Cheng Yeh and Che-hui Lien. 2009. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications* 36, 2 (2009), 2473–2480.

[37] Stella X Yu and Jianbo Shi. 2001. Grouping with bias. *Advances in Neural Information Processing Systems* 14 (2001), 1327–1334.

[38] Stella X Yu and Jianbo Shi. 2004. Segmentation given partial grouping constraints. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 2 (2004), 173–183.

[39] Fengda Zhang, Kun Kuang, Long Chen, Yuxuan Liu, Chao Wu, and Jun Xiao. 2022. Fairness-aware contrastive learning with partially annotated sensitive attributes. In *The Eleventh International Conference on Learning Representations*.

[40] Tao Zhang, Tianqing Zhu, Jing Li, Mengde Han, Wanlei Zhou, and Philip S. Yu. 2020. Fairness in semi-supervised learning: Unlabeled data help to reduce discrimination. *IEEE Transactions on Knowledge and Data Engineering* 34, 4 (2020), 1763–1774.