D3.2 Algorithms for Mitigating Bias

Christos Gkartzios[1], **Christos Karanikolopoulos**[1], **Panagiotis Papadakos**[1, 2], **Glykeria Toulina**[1], **Spyridon Tzimas**[1], **Panayiotis Tsaparas**[1]

[1] Department of Computer Science and Engineering, University of Ioannina, Greece
[2]Institute of Computer Science (ICS) - Foundation for Research and Technology - Hellas (FORTH), Greece

1. Introduction

The AI revolution has ushered in a world where algorithms increasingly make, or assist in making, decisions that affect our lives in diverse ways. These decisions can be relatively trivial (e.g., suggesting the next song to play) or highly consequential (e.g., recommending a career path, determining a medical treatment, or influencing judicial sentencing). Given this growing influence, there are serious concerns about whether algorithmic decisions are fair and unbiased. Such concerns are supported by mounting empirical evidence: multiple cases have revealed that automated systems can exhibit bias against specific population subgroups, often with harmful consequences [3].

The demand for fairness guarantees in algorithmic outputs has given rise to the research areas of *Responsible AI* and *Algorithmic Fairness*. Considerable effort has been invested in understanding and quantifying algorithmic bias, and in designing methods to mitigate it. Bias mitigation aims to produce algorithms with formal fairness guarantees, typically defined under specific fairness criteria.

In Deliverable D3.1, we outlined general approaches to bias mitigation and presented key algorithms from the literature relevant to the Themis project. In this report, we present our current contributions to bias mitigation and the design of novel fair algorithms.

The report is structured as follows:

- Section 2 provides an overview of our current work on bias mitigation and fair algorithm design.
- Section 3 presents our fair algorithms for community detection and outlines ongoing efforts to extend these methods to numerical data.
- Section 4 presents techniques for bias mitigation in opinion formation processes.
- Section 5 presents an approach for achieving Pagerank fairness at minimum cost.
- Section 6 presents a methodology for enforcing fairness in the k-NN classification algorithm.
- Section 7 concludes the paper.

2. Current Work Overview

Following the categorization in Deliverable D3.1, all our algorithms adopt the *in-processing* bias mitigation approach. That is, we design or modify algorithms to directly incorporate fairness during computation, rather than before or after the process.

Our work focuses on fair algorithm design for the following core problems: clustering and community detection, PageRank, opinion formation, and classification. The main contributions and our ongoing work are summarized below:

- **Fair Modularity-based Community Detection.** We propose a novel fair community detection algorithm that incorporates a community modularity fairness metric we introduce. It extends the popular Louvain algorithm and was published in the 2025 ACM International Web Conference [16]. We have also adapted our approach for spectral clustering, with a submission currently under review for the 2025 IEEE ICDM Conference.

- **Fair Label-Propagation.** We present a methodology for integrating balance into the popular Label Propagation community detection algorithm using physics-inspired principles. This approach was presented in the Algorithmic Fairness in Network Science workshop at NetSci 2025, and published at ASONAM 2025 [23]. We are currently exploring the application of the general methodology to clustering problems involving numerical data.

- **Fair Opinion Formation.** We introduce a new fairness metric for opinion dynamics and propose an algorithm that achieves fairness in such processes. This work has been submitted for publication to the 2025 IEEE ICDM Conference.

- **Fair Pagerank.** We propose a fair version of the PageRank algorithm, building on the framework in [25]. Our approach aims to minimize the cost of the local interventions required for achieving fairness. This work has been submitted for publication to the 2026 ACM KDD Conference.

- **Fair k-NN Classification.** We are developing a novel method for achieving fairness in the k-Nearest-Neighbors (k-NN) classification algorithm. A submission to the ACM SIGMOD 2026 Conference is currently in preparation.

In the following sections, we elaborate on each of these contributions in detail.

3. Fair Algorithms for Community Detection

Given as input a network $G = (V, E)$, the output of a community detection algorithm is a partition of the nodes into k disjoint subsets (communities), $\mathcal{C} = \{C_1, C_2, ..C_k\}$, $C_i \subseteq V$, $C_i \cap C_j = \emptyset$, $\cup_{i=1}^{k} C_i = V$. The number of communities k may be given as input, or it may be decided by the algorithm. The goal is to discover communities where the nodes are densely intra-connected, while sparsely inter-connected.

To define fairness, we assume that the nodes of the graph are associated with some sensitive attribute A, such as gender, religion or race, that takes t values $\{a_1, ..., a_t\}$, which partition the nodes of the graph into t *groups* $G = \{G_1, G_2, .., G_t\}$, $G_i = \{v \in V : A(v) = a_i\}$. In the following, we will often refer to the attribute values, and the corresponding groups, as *colors*. For simplicity, we assume two colors Red (R) and Blue (B). We will assume that the red group is the *protected* or minority group, for which we want to mitigate bias.

There is a variety of community detection algorithms that use different criteria to produce communities [13]. In our work we consider two approaches: modularity-based algorithms [20, 5, 24], and label propagation algorithms [22].

3.1. Modularity-based Community Detection

Given that network processes, including opinion formation, information propagation, and diffusion, primarily occur through interactions along the edges of the network [30, 12], in [16], we look into group fairness from the edge perspective, and we introduce a modularity-based metric of fairness.

3.1.1. Group Modularity fairness

Modularity measures the divergence between the number of intra-community edges and the expected such number assuming a null model [18, 10]. The most commonly used null model is a random graph where the expected degree of each node within the graph is equal to the actual degree of the corresponding node in the real network. Specifically, the modularity of community $\mathcal{C}_i$, $Q(\mathcal{C}_i)$, is defined as [18]:

$$Q(C_i) = \frac{1}{2m} \left(\sum_{u \in C_i} \sum_{v \in C_i} A_{uv} - \frac{k_u\, k_v}{2m} \right) \tag{1}$$

where A is the adjacency matrix of G, m the number of edges in G and k_u, k_v the degree of node u, and v respectively. Modularity provides a measure of how well nodes in a community are connected with each other. Negative values indicate less connections than expected, while positive values indicate more connections.

Our goal is to ensure that red nodes are well connected within each community. Thus, for each red node u in C_i we take the difference between the actual number of its intra-community edges and the expected such number. We call this measure *red modularity.*

As before, the expected number of connections is estimated assuming as null model a random graph that preserves the degrees of nodes in G. Using this null model, red modularity, $Q^R(C_i)$ is defined as:

$$Q^R(C_i) = \frac{1}{2m} \sum_{u \in C_i^R} \sum_{v \in C_i} \left(A_{uv} - \frac{k_u\, k_v}{2m} \right). \tag{2}$$

We define similarly the *blue modularity* $Q^B(C_i)$. We refer to red and blue modularity collectively as *group modularity.*

Note that if we consider the whole graph as a single community both the red and the blue modularity are zero. In general, positive values in a community mean that the nodes with the corresponding color are more connected in the community than expected.

We define *(group) modularity unfairness* by comparing the red and blue modularity.

Definition 1 *For a community $C_i \in \mathcal{C}$, the modularity unfairness of C_i, $u(C_i)$, is defined as:*

$$u(C_i) = Q^R(C_i) - Q^B(C_i).$$

Negative values of $u(C_i)$ indicate unfairness towards the red group meaning that the red nodes are less well-connected within the community than the blue ones. Positive values indicate the opposite, while a zero value indicates lack of unfairness towards any of the groups.

We also consider diversity within each community by looking at the edges that connect nodes of different color. Let us call these edges *diverse edges.* Note that the expected number of diverse edges cannot be estimated using the same null model, since we need to know the color of both endpoints of each edge. Instead, in this case, we estimate the expected number of diverse edges using as null model a random bipartite graph, with edges only between nodes of different color, that preserves the degrees of the nodes in the original graph G.

For a community C_i, the *diversity modularity*, or simply *diversity*, is defined as:

$$D^{RB}(C_i) = \frac{1}{2m} \sum_{u \in C_i^R} \sum_{v \in C_i^B} \left(A_{uv} - \frac{k_u\, k_v}{m} \right). \tag{3}$$

If we consider the whole graph as a single community, then diversity takes a non positive value. The larger the value of D^{RB} the more diverse the network.

We also consider a null model which is not agnostic of the color of edge endpoints. For a node u, let k_u^R be the number of edges of u to red nodes and k_u^B be the number of edges of u to blue nodes, $k_u^R + k_u^B = k_u$. In the following, k_u^R and k_u^B are respectively called the *red degree* and *blue degree* of node u.

We consider as null model a random graph where the expected red degree and the expected blue degree of each node is equal to the actual red degree and blue degree of the corresponding node in the real graph G. Formally, let P_{uv} be the probability of creating an edge between nodes u and v. Let m_{RR} be the number of red-red edges, m_{RB} the number of red-blue edges and m_{BB} the number of blue-blue edges in the graph. We have that $P_{uv} = k_u^R\, k_v^R / 2m_{RR}$, for red nodes $u, v \in R$, $P_{uv} = k_u^B\, k_v^B / 2m_{BB}$ for blue nodes $u, v \in B$, and $P_{uv} = k_u^B\, k_v^R / m_{RB}$ for red-blue nodes $u \in R$ and $v \in B$. For any node u, it holds that $\sum_{v \in R} P_{uv} = k_u^R$ and $\sum_{v \in B} P_{uv} = k_u^B$.

We define the *labeled red modularity* $Q_L^R(C_i)$ by taking again the difference between the actual number of intra-community edges involving red nodes, and the expected such number, but now considering the color (or, in general, label) of both endpoints.

$$\begin{aligned} Q_L^R(C_i) = \frac{1}{2m}\Big(& \sum_{u \in C_i^R} \sum_{v \in C_i^B} (A_{uv} - \frac{k_u^B\, k_v^R}{m_{RB}}) \\ & + \sum_{u \in C_i^R} \sum_{v \in C_i^R} (A_{uv} - \frac{k_u^R\, k_v^R}{2m_{RR}})\Big). \end{aligned} \tag{4}$$

We define similarly the *labeled blue modularity* $Q_L^B(C_i)$. We refer to labeled red and labeled blue modularity collectively as *labeled group modularity.* Again, if we consider the whole graph as a single community both the labeled red and the labeled blue modularity are zero.

We define the *labeled modularity unfairness* by comparing the red and blue labeled modularity.

Definition 2 *For a community $C_i \in \mathcal{C}$, the labeled modularity unfairness of C_i, $u_L(C_i)$, is defined as:*

$$u_L(C_i) = Q_L^R(C_i) - Q_L^B(C_i).$$

Negative values of $u_L(C_i)$ indicate unfairness towards the red group, positive values indicate unfairness towards the blue group, and a zero value lack of unfairness.

We define *labeled diversity modularity*, or simply *labeled diversity*, as follows:

$$D_L^{RB}(C_i) = \frac{1}{2m} \left(\sum_{u \in C_i^R} \sum_{v \in C_i^B} \left(A_{uv} - \frac{k_u^B\, k_v^R}{m_{RB}} \right) \right). \tag{5}$$

The labeled diversity of the whole graph is zero, while positive diversity values in a community indicate that the community contains more diverse edges than expected.

3.1.2. Fairness-Aware Louvain Community Detection

Given the definitions of modularity fairness, we propose a novel fairness-aware community detection algorithm. Our algorithm is based on the well-known Louvain algorithm that identifies communities in networks by optimizing modularity [10, 5, 24]. The algorithm was published in [16].

The fairness-aware Louvain algorithm (Algorithm 1) follows a hierarchical agglomerative approach, starting with each node forming its own community. The original Louvain algorithm joins together two communities whose merge produces the largest increase in modularity Q (Eq. 1). The fairness-aware algorithm uses two-criteria: two communities are joined if (1) modularity increases and (2) a group fairness criterion (FC) is met.

For FC, we consider different approaches using either the non-labeled and the labeled group modularity, namely:

(a) the *fairness-gain* approach where we ask that unfairness in absolute value decreases,

(b) the *group-increase* approach, where we ask that the group modularity of the group towards which the network is unfair increases

(c) the *diversity-increase* approach where we ask that diversity increases.

The algorithm operates in two phases that are repeated iteratively. In the first phase, the algorithm computes for each node u the gain in modularity and the fairness criterion FC when removing u from its current community and placing it to each of its neighboring communities. This process is applied repeatedly and sequentially and stops when a local maxima is reached, i.e., when no individual move can both increase modularity and satisfy the FC criterion.

In the second phase, the algorithm constructs a new graph whose nodes are now the communities found during the first phase. The algorithm operates on a weighted graph; each edge e is associated with a weight, $w(e)$, initially set to 1. Upon merging, the weights of the edges between the two nodes are set equal to the sum of the weights of the edges in the corresponding two communities. Edges between the nodes inside each community are modeled with a self-loop whose weight is the sum of the weights of these edges. For the node degrees, it holds $k(u) = \sum_{v,(u,v)\in E} w(u,v)$ and $k^X(u) = \sum_{(u,v)\in E, v\in X} w(u,v)$, for $X \in \{R, B\}$.

Once the graph is constructed, the first and second phase are repeated on the new graph. The iterations continue until there are no changes. The computational complexity of Algorithm 1 is $\mathcal{O}(L|E|)$, the same with the original Louvain, where $\mathcal{O}(|E|)$ is the complexity of the two phases, and L the number of iterations.

Algorithm 1 Fairness-Aware Louvain

```
Input: Graph $G(V, E, A)$ where $V$ is the set of nodes, $E$ is the set of edges, and $A$ are the node colors
Output: List of communities detected.
repeat
  Assign every node $v \in V$ to a singleton community
  Calculate modularity $Q$
  for each node $v \in V$ do
    for each $u$ in neighbors of $v$ do
      Calculate the modularity gain $\Delta Q$ and fairness criterion $FC$ from the removal of $v$ from its current community and placement in the community of each neighbor.
      if modularity increases and $FC$ is met then
        Move $v$ to neighboring community
      end if
    end for
  end for
  Create new "super-nodes" from the communities found in previous step. The new $V$ set consists of these "super-nodes".
  Recalculate the weight of the edges between these new "super-nodes".
until there is no improvement
```

3.1.3. Group-Aware Modularity Matrices

Building upon the work in [16], we now consider a similar definition of modularity fairness, and we propose modifications of the Spectral and Deep Community Detection algorithms that incorporate fairness.

Let $A \in \mathbb{R}^{n\times n}$ denote the adjacency matrix of the graph G. We partition A into four disjoint sub-matrices:

$$A = \begin{bmatrix} A_{RR} & A_{RB} \\ A_{BR} & A_{BB} \end{bmatrix}$$

where

- A_{RR}: the matrix with the edges between Red nodes,
- A_{RB}: the matrix with the edges from Red nodes to Blue nodes,
- A_{BR}: the matrix with the edges from Blues nodes to Red nodes, and
- A_{BB}: the matrix with the edges between Blue nodes.

Since the graph is undirected, it holds that $A_{RB} = A_{BR}^\top$.

We now define the sub-matrices A_R, A_B, and A_{div} as follows:

$$A_R = \begin{bmatrix} A_{RR} & A_{RB} \\ A_{BR} & \mathbf{0} \end{bmatrix} \quad A_B = \begin{bmatrix} \mathbf{0} & A_{RB} \\ A_{BR} & A_{BB} \end{bmatrix} \quad A_{\text{div}} = \begin{bmatrix} \mathbf{0} & A_{RB} \\ A_{BR} & \mathbf{0} \end{bmatrix}$$

where

- A_R: the matrix with all edges incident to Red nodes,
- A_B: the matrix with all edges incident to Blue nodes
- A_{div}: the inter-group adjacency matrix, capturing diversity across groups.

Given these matrices, we can define the corresponding subgraphs G_R, G_B and G_{div}. We will use these matrices to decompose the modularity matrix, and define the clustering objectives that we use throughout this work.

Classical modularity optimization builds upon the *modularity matrix*, introduced by Newman [19]:

$$B = A - \frac{dd^\top}{2m}$$

where A is the adjacency matrix, d is the degree vector with d_i being the degree of node i, and m is the number of edges in the graph.

The modularity score for the partition $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ can be computed using the modularity matrix B. Let $S \in \{0,1\}^{n\times k}$ be the binary community assignment matrix, where $S_{ij} = 1$ if node $i \in C_j$, and 0 otherwise. Then, the modularity of the partition $\mathcal{C}$ defined by the assignment matrix S is given by:

$$Q(\mathcal{S}) = \frac{1}{2m} \operatorname{Tr}(S^\top B S), \tag{6}$$

where $\operatorname{Tr}(\cdot)$ denotes the matrix trace [19].

We can now use the decomposition of the adjacency matrix to define the group-aware variants of modularity defined in [16], by decomposing the modularity matrix. Specifically, we define the *red modularity matrix* B_R using the red adjacency matrix A_R as follows:

$$B_R = A_R - \frac{d_R d_R^\top}{2m_R}$$

where d_R is the degree vector for the graph G_R and m_R is the number of edges in the graph G_R. The modularity score for red group connectivity, denoted Q_{R}, is then given by:

$$Q_{\text{R}}(S) = \frac{1}{2m} \operatorname{Tr}(S^\top B_R S) \tag{7}$$

Analogously, we define the *blue modularity matrix* B_B using the blue adjacency matrix A_B and the degree vector d_B of the graph G_B. We also define the *diversity modularity matrix* B_{div} using the matrix A_{div} and the corresponding degree vector d_{div}:

$$B_{\text{div}} = A_{\text{div}} - \frac{d_{\text{div}} d_{\text{div}}^\top}{2m_{\text{div}}}$$

$$Q_{\text{div}}(S) = \frac{1}{2m}\operatorname{Tr}(S^\top B_{\text{div}} S) \tag{8}$$

These formulations allow us to measure and optimize modularity with respect to both group-specific and inter-group connectivity patterns. By incorporating group constraints directly into the adjacency structure, our modularity matrix variants enable fairness-aware spectral optimization, and fairness-aware loss functions.

3.1.4. Spectral and Deep Modularity Fair Algorithms

We now describe our algorithms for optimizing group modularity. We consider two classes of algorithms. Spectral community detection algorithms that make use of the eigenvectors of the modularity matrices, and deep community detection algorithms that use the modularity matrices to redefine the loss function.

Input-Based Fair Spectral Community Detection

Spectral community detection using the modularity matrix B was first introduced in [19]. The approach is similar to that of spectral clustering using the Laplacian matrix. The algorithm computes the top-k eigenvectors of the matrix B, which are viewed as a continuous approximation of the discrete binary assignment matrix S. It then uses these vectors to obtain the communities, typically by applying k-means clustering on the extracted k-dimensional vectors of the nodes.

To enable fairness-aware spectral clustering, we define a modified modularity matrix:

$$B_X^{(\lambda)} = (1-\lambda)B + \lambda B_X,$$

where B is the standard modularity matrix, and $B_X \in \{B_R, B_{\text{div}}\}$ is a fairness-aware modularity matrix chosen according to the fairness criterion (e.g., group or diversity), with the parameter $\lambda \in [0,1]$ controlling the emphasis on structural versus fairness-aware connectivity.

Specifically, we propose two new spectral community detection algorithms that rely on the different modularity matrices we have defined:

GroupSpectral: The algorithm assumes a protected group, usually the minority one, for which we want to achieve strong internal connectivity. It uses the modified modularity matrix $B_X^{(\lambda)} = (1-\lambda)B + \lambda B_X$, where $B_X \in \{B_R, B_B\}$ is the group modularity matrix corresponding to the selected protected group (red or blue). Then it applies the spectral clustering process: It extracts the k largest eigenvectors of $B_X^{(\lambda)}$ and performs k-means clustering to obtain the communities. The goal is to discover communities that simultaneously preserve structural quality and enhance group connectivity for the protected group.

DiversitySpectral: This algorithm uses $B_{\text{div}}^{(\lambda)} = (1-\lambda)B + \lambda B_{\text{div}}$ matrix to extract the eigenvectors, and as before applies k-means to obtain the communities. The goal of the algorithm is to obtain communities with high diversity modularity.

All spectral models have complexity $\mathcal{O}(mk)$ for sparse graphs [29].

Input-Based Fair Deep Community Detection

For this class of algorithms, we extend the Deep Modularity Network (DMoN) framework introduced in [26] to incorporate fairness-aware objectives. DMoN uses a Graph Convolutional Network (GCN) on the normalized adjacency matrix A to obtain k-dimensional node embeddings. Then it applies soft-max on the embeddings to obtain a soft assignment matrix S of the nodes to clusters. This soft assignment matrix is used to define the loss function $\mathcal{L}_{\text{DMoN}}$, which is defined as follows:

$$\mathcal{L}_{\text{DMoN}} = -\frac{1}{2m}\operatorname{Tr}(S^\top B S) + \gamma \mathcal{R}_{\text{collapse}}$$

The first term corresponds to the modularity Q, while the second term is a regularization term that is defined as:

$$\mathcal{R}_{\text{collapse}} = \left(\frac{\sqrt{k}}{n} \|S^\top\|_F - 1 \right)$$

This term discourages degenerate clustering solutions, such as assigning all nodes to a single community.

Building on the fairness-aware modularity formulations used in the spectral setting, we extend this idea to the deep clustering framework by modifying the input adjacency matrix. Specifically, we define:

$$A_X^{(\lambda)} = (1-\lambda)A + \lambda A_X,$$

where $A_X \in \{A_R, A_{\text{div}}\}$ is selected based on the desired connectivity objectiveeither group or diversity based. This modification retains the global structure of the graph while increasing the influence of A_X. The parameter $\lambda \in [0,1]$ controls the trade-off between preserving global structure and increasing the influence of group-based connectivity.

GroupDMoN: This algorithm promotes strong connectivity within a protected group by using the modified adjacency matrix $A_X^{(\lambda)} = (1-\lambda)A + \lambda A_X$, where $A_X = A_R$ retains only edges involving nodes from the target group (e.g., red nodes). The model is trained on $A_X^{(\lambda)}$ and learns cluster assignments by optimizing a trade-off between structural and group modularity, with B_X contributing to the modularity objective for the protected group. The goal of the algorithm is to form communities with high group modularity for the protected group.

DiversityDMoN: This algorithm promotes diversity by using the modified adjacency matrix $A_{\text{div}}^{(\lambda)} = (1-\lambda)A + \lambda A_{\text{div}}$, which retains only red-blue edges. The model is trained on $A_{\text{div}}^{(\lambda)}$ and learns cluster assignments by optimizing a trade-off between structural and diversity-based modularity, with B_{div} contributing to the modularity objective. The goal of the algorithm is to form communities with high diversity.

Algorithm 2 Fairness-Aware Community Detection

```
1: Input: Graph $G = (V, E)$; number of clusters $k$; method type $\mathsf{M}$ (Spectral, DMoN, Deep);
   fairness objective $\mathcal{F}$
2: Output: Community assignments $\{c_1, \ldots, c_n\}$
3: Construct fairness-aware matrices based on $\mathcal{F}$
4: Compute node representations $H$ via $\mathsf{M}$ (spectral for Spectral, GNN for DMoN/Deep)
5: Cluster nodes into $k$ communities using $H$
6: return Community assignments $\{c_1, \ldots, c_n\}$
```

Loss-Based Fair Deep Community Detection

The above models enforce fairness by modifying the input adjacency matrix, which influences the learned node representations. We now present an alternative strategy that incorporates fairness directly into the loss function, without changing the input graph structure. We define three such fairness-aware deep community detection algorithms, each based on a distinct group modularity objective.

DeepDiversity: The algorithm promotes the formation of communities with high diversity within the communities, maximizing diversity modularity Q_{div} in addition to the overall modularity. Therefore, the loss function is defined as:

$$\mathcal{L}_{\text{DeepDiversity}} = -\lambda\, Q_{\text{div}} \;-\; (1-\lambda)Q \;+\; \gamma\, \mathcal{R}_{\text{collapse}}$$

where λ is a parameter that controls the tradeoff between modularity and diversity.

DeepGroup: The algorithm enhances the connectivity of a particular group (red or blue) by increasing its group modularity Q_{group} in addition to the overall modularity. Therefore, the loss

Algorithm 3 Fair Label Propagation (FLP)

```
Require: Graph $G = (V, E)$, group partition $\{G_b, G_r\}$, parameter $\lambda$.
 1: Assign a unique label $L(v) = v$ to every node $v \in V$.
 2: repeat
 3:   for all $v \in V$ do
 4:     for all labels $\ell$ do
 5:       $N_\ell(v) = \{u \in N(v) : L(u) = \ell\}$
 6:       $F_p(C_\ell, v) = |N_\ell(v)|$; $F_e(C_\ell, v) = \text{sign}(v, C_\ell)|N_\ell(v)|imb(C_\ell)$
 7:       $F(C_\ell, v) = (1 - \lambda)F_p(C_\ell, v) + \lambda F_e(C_\ell, v)$
 8:     end for
 9:     $L(v) = \arg\max_\ell F(C_\ell, v)$
10:   end for
11: until No label change
```

function is:

$$\mathcal{L}_{\text{DeepGroup}} = -\lambda\, Q_{\text{group}} \;-\; (1-\lambda)Q \;+\; \gamma\, \mathcal{R}_{\text{collapse}}$$

The parameter λ controls the tradeoff between modularity and group modularity.

In addition to enhancing intra-group connectivity or diversity, we propose a third approach that directly incorporates fairness into the loss function. According to [16], unfairness is defined as the difference between the group modularity scores, Unfairness $= Q_R - Q_B$. A network partition is considered fair when this difference is close to zero, indicating that both groups are equally well connected within the discovered communities.

DeepFairness: The algorithm enforces group-level modularity balance by minimizing the difference between red and blue group modularities, in addition to maximizing overall modularity. Therefore, the loss function is defined as:

$$\mathcal{L}_{\text{DeepFairness}} = -Q \;+\; \phi\, |Q_{\text{R}} - Q_{\text{B}}| \;+\; \gamma\, \mathcal{R}_{\text{collapse}}$$

where ϕ is a parameter that controls the importance of the fairness term in the loss function.

Algorithm 2 gives the outline of our fairness-aware deep clustering framework, parameterized by the loss function. All models retain DMoNs per-epoch complexity, $\mathcal{O}(k^2 n + m)$ [26].

3.2. Fair Label Propagation

We now consider a different metric of clustering and community fairness: *balance*. The balance fairness metric was first defined in the work of Chierichetti et al. [7] for fair clustering. The definition can be directly applied to community detection, by simply substituting clusters with communities. For the following, we will assume that we have two groups (colors) of nodes. We will refer to them as the blue group G_b and the red group G_r. For a community C, let C^b and C^r denote the subset of blue and red of nodes in C respectively. We define the balance of community C as

$$bal(C) = \min\left\{\frac{|C^b|}{|C^r|}, \frac{|C^r|}{|C^b|}\right\} \in [0, 1] \tag{9}$$

A perfectly balanced community has equal number of red and blue nodes, resulting in a balance value of 1.

Given the definition of the balance of a community, the balance of a collection of communities $\mathcal{C} = \{C_1, \ldots, C_k\}$ is defined as the average balance of the communities in $\mathcal{C}$, that is, $bal(\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{C_i \in \mathcal{C}} bal(C_i)$.

We propose a fair community detection algorithm that produces balanced communities, and extends the popular *Label Propagation* (LP) algorithm [22].The algorithm starts by assigning each node v a unique label $L(v)$, usually the id of the node. Then it iteratively updates the label of each node, assigning the most frequent label in their neighborhood. If there are ties, the label with the largest id is selected. The algorithm terminates when no node changes label. A community $C_\ell = \{v \in V : L(v) = \ell\}$ is the set of nodes with the same label ℓ.

The Fair Label Propagation algorithm (FLP) builds upon the vanilla LP algorithm, taking a physics-inspired view of the LP algorithm. We assume that adjacent nodes (u, v) in the graph pull each other with force $F_p(u, v) = 1$. In an iteration of the LP algorithm, if $N_\ell(v)$ is the set of neighbors of v with label ℓ, then the community C_ℓ pulls node v with force $F_p(C_\ell, v) = |N_\ell(v)|$. Node v is assigned to the community C_ℓ that pulls node v the strongest, that is, the most popular community in the neighborhood of v.

To incorporate fairness (balance) in the LP algorithm, we introduce an additional *electrostatic* force between connected nodes. We assign to each node v a *charge* q_v, whose magnitude depends on the *imbalance* of the community node v belongs to. We define the imbalance of a community C as $imb(C) = 1 - bal(C)$, and set $|q_v| = imb(C)$, for the nodes in C.

The polarity of the charge depends on the majority color in the community C. Without loss of generality, we assume that if the majority color is red ($|C^r| > |C^b|$), then the charge is positive. In this case, we say that community C is a red community. If the majority color in C is blue ($|C^b| > |C^r|$, community C is a blue community), then the charge is negative. Note that a single red node has charge $+1$, a single blue node has charge -1, while a balanced community has charge 0.

The electrostatic force between two charged objects with charges q_1 and q_2 at distance d, is governed by Coulomb's law and it has magnitude $|F_e(q_1, q_2)| = K_c \frac{|q_1||q_2|}{d^2}$. The force is attractive if the charges have opposite sign, and repellent if the signs are the same. In our case, we assume that two connected nodes (u, v) exert electrostatic force to each other. The force is directionless, so we only care about the sign and the magnitude of the force. We assume that the nodes are at distance 1, and we set $K_c = 1$. Therefore, we have: $F_e(u, v) = -q_u q_v$.

When considering the label assignment of a node v, the FLP algorithm, computes the electrostatic force that the neighbors of v in the community C_ℓ exert to the node. The charge of the node v has magnitude $|q_v| = 1$, while the charge of a neighbor u in the C_ℓ community has magnitude $|q_u| = imb(C_\ell)$. Summing over $N_\ell(v)$, the neighbors of v in C_ℓ, the electrostatic force of community C_ℓ to node v is:

$$F_e(C_\ell, v) = \text{sign}(v, C_\ell)|N_\ell(v)|imb(C_\ell),$$

where $\text{sign}(v, C_\ell) = +1$ if v and C_ℓ are of the same color, and $\text{sign}(v, C_\ell) = -1$ if they have different color.

The electrostatic force captures the effect of node v on the balance of the community C_ℓ. An attractive (positive) force means that the community C_ℓ has a surplus of the opposite color of v, and adding v to the community will improve its balance. A repellent (negative) force means that the community has a surplus of the color of v, and adding v to the community will further increase the imbalance of the community. The strongest the force, the more unbalanced the community (positively or negatively).

The FLP algorithm computes the total force $F(C_\ell, v)$ that the community C_ℓ exerts on node v, by combining the pulling and electrostatic forces, that is:

$$F(C_\ell, v) = (1 - \lambda)F_p(C_\ell, v) + \lambda F_e(C_\ell, v),$$

where λ is a parameter of the algorithm. By combining the two forces, the algorithm aims to combine two objectives: The quality of the output communities, which is achieved by the pulling force, as in the vanilla LP algorithm, and the fairness of the output communities, which is achieved by the electrostatic force. The parameter λ controls the tradeoff between community quality and fairness. Higher values of λ place more emphasis on fairness. The value $\lambda = 0$ corresponds to the vanilla LP algorithm, while the case $\lambda = 1$ puts all the emphasis on fairness.

Putting everything together, the FLP algorithm operates exactly as the vanilla LP algorithm, iteratively updating the labels of the nodes, each time assigning a node v to the community C_ℓ

that exerts the strongest force $F(C_\ell, v)$. The outline of the algorithm is shown in Algorithm 3. The algorithm complexity remains $O(Im)$, with m edges, and I iterations.

3.3. Clustering fairness

The problem of community detection on graphs, is essentially a clustering problem on graph data. We can apply the physics-based idea for enforcing balance in communities for enforcing balance in communities of numerical vectors. In this case we have as input a set of vectors $X = \{x_1, ..., x_n\}$. Again we assume that the points in X are partitioned into two sets R, B, depending on their *color*, that is, the value of a sensitive attribute. The goal is to partition X into communities $\mathcal{C} = \{C_1, ..., C_k\}$ such that we maximize the balance of the partition, while retaining a high-quality clustering. We can define again the charge $q(C)$ of a community $q(C_i)$, and the charge $q(x_i)$ of a single point x_i.

We propose a modification of the popular k-means algorithm that incorporates balance fairness. Recall that the k-means algorithms proceeds iteratively: at each iteration we have k centers, and we assign each point to the closest center. Following the assignment, we find the new center of the cluster, and we repeat until there is no change in the centers.

Following the physics-based interpretation, we think of each center c of community C as exerting a gravitational attractive force to each point x_i,

$$F_g(c, x_i) = \frac{1}{d(c, x_i)^2}$$

The point is assigned to the community that attracts it the most.

Using the same idea as before, we define the charge of the center point c of community C, to have measure equal to the imbalance $imb(C)$ of the community C. For a point x_i the charge has measure 1, with sign depending on the color of the point. We then define the electrostatic force that the center c exerts to the point x_i, as

$$F_e(c, x_i) = -\frac{q(c)q(x_i)}{d(c, x_i)^2}$$

The total force exerted to the point by the community C is then defined as

$$F(C, x_i) = \lambda F_e(c, x_i) + (1 - \lambda) F_g(c, x_i)$$

The point x_i is assigned to the community that pulls it the strongest.

The algorithm then proceeds iteratively, similar to k-means. We start with some centers $\{c_1, ..., c_k\}$, which also have a charge $q(c_i)$. For example, c_i's may be points in X, or the centers of an initial partition (e.g., a random partition) of X. We then compute the force that each center exerts to each point, and assign each point to the center that attracts it the most (or repels it the least). We compute the new centers and the new charges and repeat until there is no change.

We are in the process of implementing and experimenting with this algorithm. We are also considering further extensions and applications of the physics-based clustering for numerical data.

4. Fair Opinion Formation

An opinion-formation model defines a dynamic process by which individuals in a network form opinions. We consider the popular Friedkin-Jonshen (FJ) opinion-formation model [14]. In this model, individuals are nodes in a social network, and each node holds an *inner* and an *expressed opinion*; the former is a fixed value that represents the ground beliefs of the individual. The latter is shaped through the interactions of the individual with their social circle and their own inner opinion. The relative effect of these two factors is governed by the *stubbornness* of the individual, which takes values in $[0, 1]$ and determines the degree to which the individuals stick to their own opinion or listen to their social circle.

The FJ model defines an iterative process, where at every step each node sets their expressed opinion to be the linear combination of their inner opinion, and the (weighted) average of their neighbors expressed opinion. At convergence, the expressed opinion of a node is influenced by the inner opinions of all the nodes in the graph. We compute the influence matrix $Q = [q_{ij}]$ that defines the influence q_{ij} that node j exerts on node i. For a node j we define the influence it has in the network as the average of the influence it has on the nodes in the graph and its average opinion.

As in the *group fairness* paradigm [11, 3], we assume that the nodes in the network are partitioned into groups, according to some sensitive attribute, such as gender, race, or religion, and we ask that influence is fairly distributed among the groups. For a group T, we define the influence Q_T of the group as the sum of the influences of its members. For a given protected group T (e.g. a minority), we say that the opinion-formation process is ϕ-fair if $Q_T = \phi$ for a given fairness level ϕ. That is, we require that the protected group has a sufficiently strong voice in the network.

Given this definition of fairness, we consider the problem of achieving ϕ-fairness, by performing interventions to the opinion-formation dynamics. The interventions we consider are adjustments to the stubbornness values of the nodes. We can think of these interventions as a campaign where we empower the individuals of the protected group to firmly stand by their opinions, while we ask the members of the other groups to listen more to the opinions in their neighborhood.

We assume that the interventions come at a cost, which captures the effort required to adjust the attitude (stubbornness) of the nodes. We measure this cost using the sum of squares error between the stubbornness vectors before and after the interventions. We formally define the Minimum Stubbornness Adjustment for Fairness (MSAF) problem as an optimization problem where we seek to achieve fairness while minimizing the cost of the adjustments. To the best of our knowledge, this problem definition is novel.

While our minimization objective (cost) is a convex function, the fairness constraints are non-linear, which is the main technical challenge in MSAF. We propose two classes of algorithms for solving this problem: one is based on iteratively linearizing the constraints; the other is changing the stubborness of one node at a time, making it either 1 or 0, depending on whether they belong to protected group. To make our algorithms efficient, we use perturbation theory to derive closed-form expressions for computing the effect of modifying the stubbornness of a single node, which we use to compute partial derivatives for the cost function.

4.1. Definitions

4.1.1. Opinion Formation Model

The model we consider is the popular Friedkin and Johnsen (FJ) model [14]. In this model, we are given a graph with a set of n nodes V and edges E. Each node $i \in V$ has a fixed *inner opinion* $s_i \in [0, 1]$ and an *expressed opinion* z_i. The former is fixed, and a characteristic of the node itself; the latter is the result of the opinion formation process that involves the inner opinion of the node and the interaction of the node with the expressed opinions in its social network. Each node i is also associated with a *stubbornness* value $a_i \in (0, 1)$, which determines how opinionated the node is about its inner opinion, and how resistant it is to the opinions of others – higher values of a_i indicate greater stubbornness, meaning that node i places a large weight on their inner opinion and less on the opinion of its social circle.

Each node interacts (iteratively) with its neighboring nodes in the network, adjusting its expressed opinion z_i. These interactions are determined by the *interaction matrix* $W \in [0, 1]^{n\times n}$, which defines a weight w_{ij} for each edge $(i, j) \in E$; w_{ij} determines the importance that node i places on the expressed opinion of neighboring node j. W is row stochastic (i.e., each entry $W[i, j] = w_{ij}$ is non-negative and every row sums to 1).

In FJ, the expressed opinions are updated iteratively. At iteration t the expressed opinion of node i becomes:

$$z_i^{(t)} = a_i s_i + \sum_{j\in N_i} w_{ij} z_j^{(t-1)} ,$$

where N_i is the neighborhood of node i in G. Let $\mathbf{a}$, $\mathbf{s}$ and $\mathbf{z}$ denote the n-dimensional vectors of all stubbornness values, inner and expressed opinions of the nodes in V. We can write the update equation for the FJ model in matrix-vector terms:

$$\mathbf{z}^{(t)} = A\mathbf{s} + (I - A)W\mathbf{z}^{(t-1)}, \tag{10}$$

where $A = \mathrm{Diag}(\mathbf{a})$ is the diagonal matrix with $A[i,i] = a_i$ and I is the $n \times n$ identity matrix. At steady state, the expressed opinions of the nodes in V are:

$$\mathbf{z} = (I - (I - A)W)^{-1} A\mathbf{s}. \tag{11}$$

A unique equilibrium vector $\mathbf{z}$ exists if: (i) matrix W is *irreducible* (i.e., the underlying graph is connected) and (ii) at least one node has stubbornness $a_i > 0$.

We now define the *influence matrix* Q, which is central in our setting:

$$Q := (I - (I - A)W)^{-1}A. \tag{12}$$

The entries of matrix Q, $Q[i,j] = q_{ij} \in (0,1)$ determine the influence that node j exerts to node i. Note that $\mathbf{z} = Q\mathbf{s}$. Therefore, $z_i = \sum_{j\in V} q_{ij}s_j$. The matrix Q is stochastic, that is, $\sum_{j\in V} q_{ij} = 1$. Therefore, the value q_{ij} determines the extent to which the opinion of node i is influenced by the inner opinion s_j of node j.

For a node j, we define the *influence of node* j in the network as the average influence node j exerts to all the nodes in the network, that is,

$$Q_j = \frac{1}{n}\sum_{i\in V} q_{ij}. \tag{13}$$

The value Q_j also defines the influence of node j to the average opinion $\bar{z} = \frac{1}{n}\sum_{i\in V} z_i$ in the network. This is an important quantity, as it captures the public opinion in the network, and it is often targeted for maximization [15, 1]. We have that $\bar{z} = \sum_{j\in V} Q_j s_j$, thus, Q_j is the influence of the inner opinion s_j to the public opinion.

Note that, $Q_i \in (0,1)$. Also, since matrix Q is row-stochastic, the total influence exerted by all nodes in the network satisfies: $\sum_{i=1}^{n} Q_i = 1$. This highlights the zero-sum nature of influence in the FJ model: the total amount of influence in the system remains constant and must always sum to one. Consequently, when one node loses influence, the influence of the remaining nodes increases by redistributing the vacated influence among themselves.

4.1.2. Opinion Fairness

Following the group fairness paradigm, we assume that the nodes in V are partitioned into two groups: *red* ($R \subseteq V$) and *blue* ($B \subseteq V$) with $R \cap B = \emptyset$ and $R \cup B = V$. These groups are defined according to some sensitive attribute such as gender. For the following we use $G = (V, E, W, R, B)$ to denote the weighted graph, where the weights are given by matrix W, and the partition of the nodes into red and blue is given by R and B respectively.

For a group $T \in \{R, B\}$, we define the *group influence* of T as:

$$Q_T = \sum_{i\in T} Q_i, \tag{14}$$

where Q_i is the influence of node i, defined in (13). As discussed, we have that $\sum_{i\in V} Q_i = 1$, and thus $Q_R + Q_B = 1$.

The Q_R and Q_B values can be thought of as the strength of the voice of each group within the network. To illustrate this point, consider the case that all nodes in each group hold the same inner opinion value, s_R and s_B for groups R and B, respectively. This could model the case where the nodes of each group are the followers of opposing political parties, or the advocates and opponents of a specific policy. Then $\bar{z} = Q_R s_R + Q_B s_B$, and the Q_R and Q_B values determine the contribution of each group to the public opinion in the network.

For the opinion formation process to be fair, we require that the Q_R and Q_B values are sufficiently balanced; otherwise, we say that the process is unfair.

Definition 3 (ϕ-Fairness) *Given the input graph $G = (V, E, W, R, B)$, and a parameter $\phi \in (0, 1)$, the FJ opinion-formation process on graph G is ϕ-**fair** if and only if: $Q_R = \phi$.*

Since $Q_R + Q_B = 1$, if $Q_R = \phi$, then $Q_B = 1 - \phi$. Increasing the influence of one of the groups to achieve fairness can only be achieved by reducing by equal amount the influence of the other group.

The definition of fairness is parameterized by the value $\phi \in (0, 1)$, which can be set to enforce different fairness policies. For instance, we can set $\phi = 0.5$ to enforce equal influence between the two groups. Alternatively, we can enforce demographic parity fairness by setting ϕ equal to the fraction of red nodes in the graph. Finally, if the red group is a minority or protected group, we can also set ϕ to enforce an affirmative action policy, empowering the red group's voice.

4.2. Minimum Stubbornness Adjustment for Fairness

We now consider the problem of achieving fairness in the opinion formation process. Consider a graph $G = (V, E, W, R, B)$, desired fairness ϕ and $Q_R \neq \phi$. Assume without loss of generality that the network is unfair towards the red group, i.e. $Q_R < \phi$. Our goal is to increase the influence of the red group by performing interventions. The interventions we consider are changes in the stubbornness of the nodes. Intuitively, we want to make the red nodes more assertive (increase their stubbornness) and the blue nodes more receptive (decrease their stubbornness), to achieve ϕ-fairness. However, our interventions come at a *cost*, which is the degree to which we change the original stubbornness values. This cost can be thought of as the effort required to change the attitude of the red and blue nodes. We want to achieve ϕ-fairness with *minimal* adjustments to the stubbornness values.

Formally, let $\mathbf{a} = (a_1, \ldots, a_n)$ denote the input vector of stubbornness values and $\mathbf{a}' = (a'_1, \ldots, a'_n)$ the stubbornness values after the interventions. The cost of the interventions is:

$$\text{Cost}(\mathbf{a}') = \sum_{i=1}^{n} (a_i - a'_i)^2. \tag{15}$$

Then, we can define our optimization problem as follows.

Problem 1 *[Minimum Stubbornness Adjustment for Fairness-MSAF] Assume an input graph $G = (V, E, W, R, B)$, and an initial stubbornness vector $\mathbf{a} = (a_1, \ldots, a_n)$. Given a fairness level $\phi \in (0, 1)$, find a modified stubbornness vector $\mathbf{a}' = (a'_1, \ldots, a'_n)$ such that:*

$$\begin{aligned} \min_{\mathbf{a}'} \quad & \mathit{Cost}(\mathbf{a}') \\ \textit{subject to:} \quad & Q_R(\mathbf{a}') = \phi \\ & a'_i \in (0, 1). \end{aligned}$$

MSAF models the trade-off between enforcing global fairness and preserving the behavioral tendencies of the nodes.

Although the objective function of this optimization problem is convex (15), the problem is challenging because of the non-linearity of the fairness constraint $Q_R(\mathbf{a}') = \phi$. This constraint depends on the influence matrix Q, which is the inverse of another matrix (12) and thus, nonlinear.

4.2.1. Tools for the MSAF problem

Assume that the stubbornness value a_i of a single node is altered. The analytical expressions for the resulting variation in the group influence Q_R are formalized in the following lemma. The lemma utilizes the Sherman-Morrison formula [17].

Lemma 1 *Given an input graph $G = (V, E, W, R, B)$ and an initial stubbornness vector $\mathbf{a} = (a_1, \ldots, a_n)$, altering the stubbornness of a single node i from a_i to a_i' results in the following group influence updates:*

$$Q_R' = Q_R + (a_i' - a_i)\frac{\left(\frac{Q_i}{a_i(1-a_i)}\right)\sum_{j\in B} q_{ij}}{1 + \left(\frac{a_i'-a_i}{a_i(1-a_i)}\right)(q_{ii} - a_i)}, \quad \textit{if } i \in R$$

$$Q_R' = Q_R - (a_i' - a_i)\frac{\left(\frac{Q_i}{a_i(1-a_i)}\right)\sum_{j\in R} q_{ij}}{1 + \left(\frac{a_i'-a_i}{a_i(1-a_i)}\right)(q_{ii} - a_i)}, \quad \textit{if } i \in B,$$

where Q_i is the initial influence of node i, and Q_R is the initial Red group influence; Q_R' is the updated Red group influence after modifying the stubbornness of node i from a_i to a_i'.

By utilizing the result of Lemma 1, we derive the partial derivative of the group influence (Q_R) with respect to the stubbornness (a_i) of an individual node as follows:

$$\frac{\partial Q_R}{\partial a_i} = \begin{cases} \frac{Q_i}{a_i(1-a_i)} \sum\limits_{j\in B} q_{ij}, & i \in R \\ -\frac{Q_i}{a_i(1-a_i)} \sum\limits_{j\in R} q_{ij}, & i \in B \end{cases}. \tag{16}$$

We can simplify (16) by approximating the partial derivatives using the first-order Neumann series expansion [17].

$$\frac{\partial Q_R}{\partial a_i} \approx \begin{cases} \sum\limits_{j\in B} a_j w_{ij}, & i \in R \\ \sum\limits_{j\in R} -a_j w_{ij}, & i \in B \end{cases} \tag{17}$$

4.2.2. Properties of the MSAF problem

Before proceeding into solving the MSAF problem, it is important to understand some of its properties.

Feasibility of fairness. First, we we show that level of fairness ϕ can be achieved solely by modifying the stubbornness values a_i of the nodes.

Lemma 2 (feasibility) *Given a graph $G = (V, E, W, R, B)$, then for any desired fairness level $\phi \in (0, 1)$, there always exists at least one stubbornness vector $\mathbf{a} = (a_1, \ldots, a_n)$ such that $Q_R(\mathbf{a}) = \phi$.*

Monotonicity of fairness. Then, we show that changing the stubbornness of a node in a group has a *monotonic* effect on the influence of the group, that is, as the stubbornness of a node increases, the influence of its group increases as well, and vice versa.

Lemma 3 (monotonicity) *Let $G = (V, E, W, R, B)$ be the input graph and assume node $i \in T$, where $T \in \{R, B\}$, with initial stubbornness a_i. Then increasing the stubbornness of node i to $a_i' > a_i$, the influence Q_T of group T monotonically increases, while the influence $Q_{\bar{T}}$ of the opposing group $\bar{T}$, monotonically decreases. Conversely, if the stubbornness of node i decreases to $a_i' < a_i$, then Q_T monotonically decreases, while $Q_{\bar{T}}$ monotonically increases.*

4.3. Algorithms for the MSAF Problem

In this section, we present two families of algorithms for MSAF: (i) the `Selective` algorithms, which iteratively adjust the stubbornness of individual selected nodes to achieve the desired level of fairness; (ii) the Global Adjustment-`GA` algorithms, which frame the problem as a sequence of convex optimization subproblems with linearized constraints.

Algorithm 4 `Selective` Algorithm

Input: Interaction matrix W, initial stubbornness vector $\mathbf{a} = (a_1, \dots, a_n)$, node group-assignments, fairness target ϕ.
Output: Adjusted stubbornness values $\mathbf{a}'$ s.t. $Q_R(\mathbf{a}') = \phi$.

```
 1: Compute initial fairness: $Q_R \leftarrow Q_R(\mathbf{a})$
 2: Initialize $\mathbf{a}' \leftarrow \mathbf{a}$
 3: Initialize available node list: $V_{\text{avail}} \leftarrow V$
 4: if $Q_R < \phi$ then
 5:     while $Q_R < \phi$ do
 6:         Select $i \in V_{\text{avail}}$ and remove $i$ from $V_{\text{avail}}$
 7:         if $i \in R$ then
 8:             $a'_i \leftarrow 1 - \epsilon$
 9:         else
10:             $a'_i \leftarrow 0 + \epsilon$
11:         end if
12:         Recompute $Q_R \leftarrow Q_R(\mathbf{a}')$
13:     end while
14:     Undo last change and solve for precise $a'_i$
15:     if $i \in B$ then
16:         $a'_i \leftarrow a_i \left( \frac{(Q_R-\phi)(1-a_i)}{Q_i \sum_{j \in R} q_{ij} - (Q_R-\phi)(q_{ii}-a_i)} + 1 \right)$
17:     else
18:         $a'_i \leftarrow a_i \left( \frac{(\phi-Q_R)(1-a_i)}{Q_i \sum_{j \in B} q_{ij} - (\phi-Q_R)(q_{ii}-a_i)} + 1 \right)$
19:     end if
20: else if $Q_R > \phi$ then
21:     Same steps, but swap $R$ with $B$ on while loop
22: end if
23: Return: Adjusted stubbornness values $\mathbf{a}'$.
```

4.3.1. `Selective` Algorithms

The general `Selective` algorithm incrementally modifies the stubbornness values of individual nodes in order to steer the Red group's influence, Q_R, toward the specified fairness target ϕ. At each iteration, the algorithm selects a single node and sets its stubbornness to one of the two extremal valueseither the minimum or the maximum $\{0+\epsilon, 1-\epsilon\}$depending on whether an increase or decrease in Q_R is needed. Specifically, when $Q_R < \phi$ (resp. $Q_R > \phi$), the algorithm increases (resp. decreases) Q_R by either maximizing (resp. minimizing) the stubbornness of a Red node or minimizing (resp. maximizing) that of a Blue node. This process continues until Q_R exceeds (resp. goes below) the fairness threshold ϕ. At this point, the algorithm performs a final refinement step: it revisits the last modified node and adjusts its stubbornness value to a specific intermediate level that ensures $Q_R = \phi$ exactly.

This algorithm terminates and outputs a valied solution because of two key properties: First, as shown in Lemma 3, altering the stubbornness of a single node results in a monotonic change in the influence of its group. Second, Lemma 2 guarantees that, due to the continuity of Q_R, an exact solution always exists within the feasible domain. The precise stubbornness value for the final adjustment is computed via Lemma 1. The full pseudocode of the generic `Selective` algorithm is shown in Alg. 4.

We consider four variants of `Selective`: `Se-Rand`, `Se-Greedy`, `Se-NMA`, and `Se-SM`, which differ in the strategy used to select the next node to modify at each iteration. `Se-Rand` selects nodes uniformly at random, `Se-Greedy` greedily selects, at each iteration, the node whose modification yields the greatest decrease in $|Q_R - \phi|$, using the formulas in Lemma 1. Finally, `Se-NMA` and `Se-SM` adopt a gradient-based selection criterion: at each step, `Se-NMA` and `Se-SM` select the node having the highest absolute value of the partial derivative $\partial_i = \frac{\partial Q_R}{\partial a_i}$. Depending on whether we use the exact expression (16) or the approximation (17), we obtain the `Se-SM` and `Se-NMA` variants, respectively.

Algorithm 5 Global Adjustment-GA Algorithm

Input: Interaction matrix W, initial stubbornness vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, node group-assignments, fairness target ϕ, tolerance τ.
Output: Adjusted $\mathbf{a}' = (a'_1, a'_2, \ldots, a'_n)$ such that $Q_R(\mathbf{a}') = \phi \pm \tau$.

```
1: Initialize: Set $\mathbf{a}'^{(0)} \leftarrow \mathbf{a}$, $t \leftarrow 0$
2: while $|Q_R(\mathbf{a}'^{(t)}) - \phi| > \tau$ do
3:     Compute influence matrix $Q$ and $Q_R(\mathbf{a}'^{(t)}) = Q_R^{(t)}$
4:     Compute partial derivatives using (16) or (17)
           $$\partial_i = \frac{\partial Q_R^{(t)}}{\partial a_i'^{(t)}}.$$
5:     Update stubbornness values:
           $$a_i'^{(t+1)} = a_i'^{(t)} - \frac{Q_R^{(t)} - \phi}{\sum_{k=1}^{n} \partial_k^2} \partial_i.$$
6:     Enforce box constraints: If any $a_i'^{(t+1)}$ violates $(0, 1)$:
           $$a_i'^{(t+1)} = \begin{cases} 1 - \epsilon, & \text{if } a_i'^{(t+1)} \geq 1, \\ 0 + \epsilon, & \text{if } a_i'^{(t+1)} \leq 0. \end{cases}$$
7:     Increment iteration counter: $t \leftarrow t + 1$.
8: end while
9: Return: Adjusted stubbornness values $\mathbf{a}'$.
```

4.3.2. Global Adjustment-GA Algorithms

A key observation is that MSAF is a convex optimization problem with a non-linear constraint. The non-linearity arises from the fairness constraint $Q_R(\mathbf{a}') = \phi$, which depends on matrix inversion in the computation of q_{ij} values. Key to the GA approach is the *linearization* of the constraint using the first-order Taylor approximation of $Q_R(\mathbf{a}')$. Using the first-order Taylor expansion, we can approximate $Q_R(\mathbf{a}') \approx Q_R(\mathbf{a}) + \nabla Q_R(\mathbf{a}) \cdot (\mathbf{a}' - \mathbf{a})$. The resulting optimization sub-problem has a convex objective and a linear constraint thus it is very easy to solve analytically (see Appendix **??**). The generic Global Adjustment-GA algorithm begins by solving this linearized subproblem using the initial stubbornness vector $\mathbf{a}$. It then proceeds iteratively: at each iteration t, it solves a new linearized subproblem using the current iterate $\mathbf{a}^{(t)}$, producing the updated solution $\mathbf{a}^{(t+1)}$. After each update, box constraints are enforced to ensure that all stubbornness values remain within the interval $(0, 1)$. From standard results in non-linear optimization [4],[21] (see also Appendix **??**) we can state that this iterative process converges to $Q_R(\mathbf{a}') = \phi$. The general GA algorithm is given in Alg. 5.

Depending on whether we use the exact (16) or approximate (17) expression for the partial derivative ∂_i, we refer to the corresponding instantiations of the algorithm as GA-SM and GA-NMA, respectively.

4.3.3. Computational Complexity

In both the Selective and GA algorithms, the dominant computational cost per iteration lies in computing the influence matrix $Q = B^{-1}A$, when the stubbornness values a_i are updated. In theory, the theoretical complexity of matrix inversion can be reduced to approximately $\mathcal{O}(n^{2.37})$ [28] In practice, we employ standard dense solvers provided by the CuPy library, which internally rely on LU decomposition to solve $BQ = A$ without explicitly forming B^{-1}. This yields a per-iteration computational cost of $\mathcal{O}(n^3)$, consistent with classical direct methods such as Gaussian elimination or LU factorization [17]. The Selective algorithms exploit the fact that only a single stubbornness value a_i is modified in each iteration. This rank-one change allows us to use the ShermanMorrison formula to efficiently update the inverse B^{-1} without recomputing it from scratch. This reduces the per-iteration cost to $\mathcal{O}(n^2)$.

The runtime of each algorithm also depends on the number of iterations required to achieve ϕ-fairness, which can vary considerably in practice.

5. Minimizing the cost of Pagerank fairness

Pagerank fairness was first studied by Tsioutsiouliklis et al. [25]. They assumed that the nodes in the network are partitioned into groups, and they asked for a fair allocation of Pagerank values between the different groups. They proposed the *Locally Fair Pagerank* class of algorithms, which achieves fairness by making the transitions out of each node in the random walk fair. This locally fair behavior of the nodes, results in overall fairness of the algorithm.

Local Pagerank Fairness is a simple and intuitive notion of fairness, but it is also particularly invasive, since it modifies the behavior of all nodes in the network, and alters significantly the resulting Pagerank vector. In this paper, we aim to find a set of nodes to make fair so as to achieve fairness at the minimum *cost*. We consider two notions of cost. The first is the size of the set, aiming to minimize the changes in the transition matrix of the random walk. The second is the change in the Pagerank values of the nodes, aiming for small *utility loss* in the resulting fair Pagerank vector.

We derive analytical expressions for estimating efficiently the gain in fairness, and the loss in utility, when making a node fair. Our formulas rely on the fact that making a node fair is a rank-1 perturbation of the transition matrix, and thus we can compute the change in Pagerank without recomputing the Pagerank vector. Using our formulas, we propose greedy algorithms for selecting the nodes to make fair.

5.1. Definitions

In this section, we present the necessary background for the Pagerank algorithm and the definition of Pagerank fairness from [25].

5.1.1. The Pagerank Algorithm

The Pagerank algorithm [6] pioneered the use of graph structure for assessing the importance of nodes in a graph. It was made popular through its use in the Google search engine, but it has found several applications in other areas as well.

The algorithm takes a (directed) graph $G = (V, E)$ as input, and produces a weight vector $\mathbf{p}$, where $p(i)$ is the weight assigned to node i. The weight vector is computed by performing a *random walk with restarts* on the graph. The random walk at each step follows one of the outgoing edges uniformly at random with probability $(1 - \gamma)$, while with probability γ it jumps to a node chosen uniformly at random according to the jump vector distribution $\mathbf{u}$ (usually, the uniform vector). The Pagerank vector is the stationary distribution of the random walk.

More formally, let P denote the transition probability matrix of the random walk, when following an outgoing link. Let d_i be the out-degree of node i. We have that $P[i, j] = 1/d_i$, if $(i, j) \in E$, and 0 otherwise. For the stationary distribution of the random walk it holds that

$$\mathbf{p}^T = (1 - \gamma)\mathbf{p}^T P + \gamma\mathbf{u}^T$$

Solving for $\mathbf{p}^T$, we have that

$$\mathbf{p}^T = \gamma\mathbf{u}^T \left(I - (1 - \gamma)P\right)^{-1},$$

where I is the identity matrix. Let $Q = \gamma \left(I - (1 - \gamma)P\right)^{-1}$. We have that $\mathbf{p}^T = \mathbf{u}^T Q$. The value $p(i)$ is the Pagerank value of node i.

5.1.2. Pagerank Fairness

The fairness of the Pagerank algorithm was defined and studied in [25]. The work considers a group fairness definition. Given a graph $G = (V, E)$, they assume that the nodes in the graph are partitioned into groups, based on a sensitive attribute such as gender or race. Following [25], for simplicity, we assume that there are two groups of nodes, Red (R) and Blue (B), $R, B \subseteq V$, $R \cap B = \emptyset$, $R \cup B = V$. We will refer to the partition (R, B) as the group partition. We also assume that the red group is the *protected* group, for which we want to guarantee fair treatment. Abusing the notation, let $p(R)$ denote the total Pagerank weight (probabiilty mass) allocated to the red group. Given a parameter ϕ, the Pagerank algorithm is ϕ-fair, if $p(R) = \phi$. We will refer to the $p(R)$ as the *red pagerank*. The *blue pagerank* is defined symmetrically.

To achieve fairness, Tsioutsiouliklis et al. [25], define the *Locally Fair Pagerank* class of algorithms, where they adjust the transition probability matrix of the random walk, so that all nodes in the graph transition with probability ϕ to a red node, and with probability $1-\phi$ to a blue node. We can also think of each node as distributing their Pagerank weight to the two groups in a ϕ-fair manner. We say, that the nodes behave fairly, or that they are ϕ-fair. The jump vector $\mathbf{u}$ is also made to be ϕ-fair, with $u(i) = \phi/|R|$, if $i \in R$, and $u(i) = (1-\phi)/|B|$ if $i \in B$. They show that this local fair behavior results in global fairness of the stationary distribution.

5.1.3. Problem Definition

The locally fair algorithms require to modify the whole transition matrix P of the random walk so that the transitions out of the nodes are ϕ-fair (excluding only the nodes that are already ϕ-fair). Furthermore, the resulting Pagerank vector is considerably altered compared to the original one. This is excessively intrusive, given also that some nodes may have transition probability to red nodes already greater than ϕ, and they do not need to be made ϕ-fair. We consider the problem of identifying the set of local changes that achieve fairness, with the minimum effect.

Formally, for a subset $S \subseteq V$ of nodes, let $cost(S)$ be the cost of making the nodes in S ϕ-fair. Our goal is to find the set S of nodes to make fair, such that the cost $cost(S)$ is minimized.

We consider two definitions of cost. The first, considers the number of local changes that we need to achieve fairness. That is, $cost(S) = |S|$, is the number of nodes that we make ϕ-fair. Formally, we define our problem as follows.

Problem 2 (Minimum Modification Local Fairness (MMLF)) *Given a graph $G = (V, E)$ with group partition (R, B) and a fairness parameter $\phi \in (0, 1)$, find the smallest subset of nodes $S \subseteq V$ to make ϕ-fair, such that for the resulting Pagerank vector $\mathbf{p}'$ it holds that $p'(R) \geq \phi$.*

The next definition of cost considers the change in the resulting Pagerank vector. Let $\mathbf{p}'$ denote the Pagerank vector after making a subset of nodes fair, and let $\mathbf{p}$ denote the original Pagerank vector. In [25], they define the *utility loss* of $\mathbf{p}'$ as $loss(\mathbf{p}') = \|\mathbf{p}' - \mathbf{p}\|_2^2$, that is, the sum of squares error between the two vectors. We set $cost(S) = loss(S)$. Formally, we define our problem as follows.

Problem 3 (Minimum Loss Local Fairness (MLLF)) *Given a graph $G = (V, E)$ with group partition (R, B) and a fairness parameter $\phi \in (0, 1)$, find a subset of nodes $S \subseteq V$ to make ϕ-fair, such that for the resulting Pagerank vector $\mathbf{p}'$ it holds that $p'(R) \geq \phi$, and the utility loss $loss(p')$ is minimized.*

5.2. Single Node Modification

We now study the case where we modify a single node. We first describe the different ways we can make a node ϕ-fair, and then how to compute the effect of making a node ϕ-fair on the Pagerank vector and the red Pagerank value $p(R)$. We will utilize our derivations to design algorithms for the two problems.

5.2.1. Making a node ϕ-fair

In [25], they defined different ways of making a node ϕ-fair. We will consider two approaches in our work. For the following we assume that we have as input a graph $G = (V, E)$, with n nodes and m edges, and a group partition (R, B). For a node x, we use d_x to denote the outgoing degree of x, and d_x^R and d_x^B to denote the outgoing degree of x to red and blue nodes respectively. We use $\rho_x = d_x^R/d_x$ to denote the fraction of the neighbors of x in the red group, and $\beta_x = d_x^B/d_x$ to denote the fraction of blue neighbors. We only make ϕ-fair nodes that are ϕ-unfair to the red group, that is, nodes x, where $\rho_x < \phi$.

Neighborhood Locally Fair Pagerank: The first approach updates the transition matrix P, by redistributing the transition probability among the neighbors of the node, so that the transition probability to the red neighbors is ϕ. Formally, to make node x ϕ-fair, in the updated transition probability matrix P', we set the x row as follows:

$$P'[x, y] = \begin{cases} \phi/d_x^R & \text{if } (x, y) \in E, y \in R \\ (1-\phi)/d_x^B & \text{if } (x, y) \in E, y \in B \\ 0 & \text{if } (x, y) \notin E \end{cases} \tag{18}$$

If node x has no red neighbors, then the node allocates probability ϕ uniformly at random to all nodes in the red community R. Thus, we get

$$P'[x, y] = \begin{cases} \phi/|R| & \text{if } y \in R \\ (1-\phi)/d_x & \text{if } (x, y) \in E \\ 0 & \text{if } (x, y) \notin E, y \in B \end{cases} \tag{19}$$

Residual Locally Fair Pagerank: The second approach allocates probability mass $(1 - \delta_x)$ uniformly to the neighbors of node x, and δ_x uniformly to all nodes of the red group, where δ_x is such that we allocate probability mass ϕ to the red group.

Let $\mathbf{u}_R$ denote the n-dimensional probability vector that allocates the probability mass uniformly over the red nodes, that is, $\mathbf{u}_R(i) = 1/|R|$, if $i \in R$, and zero otherwise. Let P_x^T denote the x row of the transition matrix P, and $P_x'^T$ the x row of the updated transition matrix. We have

$$P_x'^T = (1 - \delta_x)P_x^T + \delta_x \mathbf{u}_R^T. \tag{20}$$

That is, when transitioning out of node x, with probability δ_x we transition to a red node uniformly at random, and with probability δ_x we transition to a randomly selected neighbor. To ensure fairness, it must be that $(1 - \delta_x)\rho_x + \delta_x = \phi$, therefore, $\delta_x = (\phi - \rho_x)/(1 - \rho_x) = (\phi - \rho_x)/\beta_x$.

5.2.2. Computing the Effect of a Single Modification

We will now derive closed-form expressions for computing the effect on fairness and utility loss of making a single node x ϕ-fair. Our derivations rely on perturbation theory [17].

Making a node x fair involves the modification of a single row of the transition matrix P to obtain a new transition matrix P'. We view this change as the addition of a perturbation matrix D to P, that is, $P' = P + D$. The matrix D is of a very specific form: all entries are zero, except for the row x, which is a vector D_x^T. We will sometimes refer to the perturbation matrix D as the perturbation vector D_x^T. The vector D_x^T depends on the type of modification we perform.

For the following, let $\mathbf{e}_x$ denote the unit vector with 1 at x and zero everywhere else (the x axis of $\mathbb{R}^n$). Note that $D = \mathbf{e}_x D_x^T$, therefore the matrix D has rank one. We can exploit perturbation theory [17] to prove the following Theorem for the updated matrix Q' after changing node x. In the following we use Q^x to denote the x column of matrix Q.

Theorem 1 *Let $G = (V, E)$ be a graph, and let P be the transition probability matrix of the Pagerank random walk on G. After the modification of node $x \in V$, with perturbation vector D_x^T, we can compute the updated matrix Q' as follows:*

$$Q' = Q + \frac{Q^x D_x^T Q}{\frac{\gamma}{1-\gamma} - D_x^T Q^x}. \tag{21}$$

Using the fact that $\mathbf{p}^T = \mathbf{u}^T Q$, and that $p(x) = \mathbf{u}^T Q^x$, we can compute the updated Pagerank vector $\mathbf{p}^T$ after modifying node x as follows:

$$\mathbf{p}'^T = \mathbf{p}^T + p(x)\frac{D_x^T Q}{Z_x},$$

where $Z_x = \frac{\gamma}{1-\gamma} - D_x^T Q^x$.

Consider now the set of red nodes R. Let $\mathbf{e}_R$ denote the vector with 1 at all entries of R and zero everywhere else. We have that $p(R) = \mathbf{p}^T \mathbf{e}_R$. Therefore, the updated red Pagerank after modifying node x is computed as:

$$p'(R) = p(R) + p(x)\frac{D_x^T Q \mathbf{e}_R}{Z_x}.$$

We define the *fairness gain* $gain(\mathbf{p}') = p'(R) - p(R)$ as the change in red Pagerank after changing node x. We have that:

$$gain(\mathbf{p}') = p(x)\frac{D_x^T Q \mathbf{e}_R}{Z_x}. \tag{22}$$

We can also compute the utility loss of $\mathbf{p}'$ with respect to vector $\mathbf{p}$, as follows:

$$loss(\mathbf{p}' \mid \mathbf{p}) = p^2(x)\frac{\|D_x^T Q\|^2}{Z_x^2}$$

5.2.3. The perturbation vector D_x

The vector D_x depends on the kind of modification we perform. We will now define D_x for the different modifications we defined above. For the following, for a node x, if N_x is the (outgoing) neighborhood of x, we use $R_x \subseteq N_x$ to denote the set of red neighbors of node x and $B_x \subseteq N_x$ to denote the set of blue neighbors of node x. Recall that $\rho_x = |R_x|/|N_x|$ is the fraction of red neighbors of x, while $\beta_x = |B_x|/|N_x|$ is the fraction of blue neighbors of x. We assume that the node we modify is ϕ-unfair, that is $\rho_x < \phi$ and $\beta_x > 1 - \phi$ for the input fairness parameter ϕ.

Neighborhood Locally Fair Pagerank: In this case, for a node x with a non-empty set of red neighbors, $R_x \neq \emptyset$, from Eq. (18) it follows that the perturbation vector D_x^T is

$$D_x^T = \frac{1}{d_x}\left((\phi/\rho_x - 1)\mathbf{e}_{R_x}^T - (\phi/\beta_x)\mathbf{e}_{B_x}^T\right)$$

In simple terms, we add to the red neighbors of x ($e_{R_x}(i) = 1$) transition probability $\frac{1}{d_x}(\phi/\rho_x - 1)$, while we subtract $\frac{1}{d_x}(\phi/\beta_x)$ from the blue neighbors ($e_{B_x}(i) = 1$).

When node x has no neighbors ($R_x = \emptyset$, and $N_x = B_x$), from Equation (19), the perturbation vector D_x^T becomes:

$$D_x^T = (\phi/|R|)\mathbf{e}_R^T - (\phi/d_x)\mathbf{e}_{N_x}^T$$

Residual Locally Fair Pagerank: In this case, from Eq. (20), if follows directly, that

$$D_x^T = -(\delta_x/d_x)\mathbf{e}_{N_x}^T + (\delta_x/|R|)\mathbf{e}_R^T$$

Using the appropriate D_x we can compute the updated matrix Q', the updated vector p', the gain $gain(p')$ and the loss $loss(p')$, using the formulas in Section 5.2.2.

5.2.4. Understanding the fairness gain

The formulas in Section 5.2.2 do not provide sufficient intuition as what is a good node to make ϕ-fair. To obtain such intuition we expand the formula in Eq. (22) for the fairness gain.

Algorithm 6 GREEDY-f Algorithm

Input: Graph $G = (V, E)$, group partition (R, B), target fairness ϕ, selection function f.
Output: The set $S \subseteq V$ of nodes to be made ϕ-fair.

```
1: Compute the set of candidate nodes $C \subseteq V$ that are $\phi$-unfair.
2: Compute initial matrix $Q$ and Pagerank vector $p$.
3: $S = \emptyset$
4: while $Q_R < \phi$ do
5:     $x = \arg\max_{x \in C} f(x)$
6:     $S = S \cup \{x\}$
7:     $C = C \setminus \{x\}$
8:     Compute updated matrix $Q$ using Eq. (21)
9:     Compute updated Pagerank vector $p$
10: end while
11: return $S$
```

Doing the computations for the Neighborhood Locally Fairness case, for a node with non-empty red neighbors set, we obtain:

$$gain(\mathbf{p}') = p(x) \frac{\left(\frac{\phi}{\rho_x} - 1\right) \sum_{z \in R_x} Q_z(R) - \frac{\phi}{\beta_x} \sum_{z \in B_x} Q_z(R)}{\frac{\gamma}{1-\gamma} - \left(\frac{\phi}{\rho_x} - 1\right) \sum_{z \in R_x} Q_z(x) + \frac{\phi}{\beta_x} \sum_{z \in B_x} Q_z(x)}. \quad (23)$$

The value $Q_z(x)$ corresponds to the entry $Q[z, x]$ of matrix Q.

The value $Q_i(j)$ is the personalized Pagerank that node i assigns to node j. The personalized Pagerank vector of node i is computed by performing the Pagerank random walk with jump (restart) vector $\mathbf{u} = \mathbf{e}_i$. That is, the random walk always restarts from node i. Given that $\mathbf{p} = \mathbf{u}Q$, it follows that the row Q_i is the personalized Pagerank vector of node i. Intuitively, the value $Q_i(j)$ captures how "close" node j is to node i. $Q_z(R) = \sum_{v \in R} Q_z(v)$ is the total personalized Pagerank that node z assigns to the red nodes.

Equation (23) characterizes the nodes that contribute to the increase of $Q(R)$ when made ϕ-fair. A node is a good selection if it has high Pagerank $p(x)$, so that it has a lot of weight to redistribute. The red neighbors in R_x should be close to other red nodes (high $Q_z(R)$), while blue neighbors in B_x far from red nodes (low $Q_z(R)$), so that the redistributed weight ends up in red nodes down the line. The denominator says that the red neighbors of x should have higher probability of returning to x than the blue, so that when redistributing the transition, we strengthen the Pagerank of node x.

5.3. Algorithms

We now present our algorithms for solving the MMLF and MLLF problems. Our algorithms follow the general outline of Algorithm 6. The algorithms take as input the graph G, the partition (R, B) the target parameter ϕ, and a function f that determines the selection criterion, depending on the problem we consider. They output the set S of the nodes that are made ϕ-fair.

The algorithms operate in a greedy fashion, building the solution set incrementally, each time adding the best candidate node to the set S, according to the selection function f. For the following let $\mathbf{p}^x$ denote the updated Pagerank vector, after making the node x ϕ-fair. We consider the following three selection functions, that result in three different greedy algorithms:

- GREEDYGAIN: $f(x) = gain(\mathbf{p}^x)$, the increase in the red Pagerank. This algorithm targets the MMLF problem and tries to find the smallest set nodes that achieve fairness.
- GREEDYLOSS: $f(x) = -\delta loss(\mathbf{p}^x)$, where $\delta loss(\mathbf{p}^x) = loss(\mathbf{p}^x) - loss(\mathbf{p})$, the increase in the utility loss. This algorithm targets the MLLF problem and tries to find the set nodes with the minimum utility loss.

- GreedyRatio: $f(x) = gain(\mathbf{p}^x)/\delta loss(\mathbf{p}^x)$, the ratio of fairness gain over utility loss increase. This algorithm tries to strike a balance between the goals of the MMLF and the MLLF problems, and find nodes that give high gain with low utility loss.

For each greedy algorithm, we have two variants depending on the local fairness approach we consider (neighborhood local fairness, or residual local fairness).

Complexity: A naive implementation of our algorithms would compute the updated matrix Q after each node modification via matrix inversion, which takes $O(n^{2.37})$ time [2]. Using the derivations from Section 5.2 we can implement the Greedy-f algorithm in time $O(kn^2)$, for $f = gain$, where $k = |S|$ is the size of the selected set,and $O(kmn^2)$ for the remaining selectors.

6. Fair k-NN Classification

6.1. Preliminaries

Let $\mathcal{X}$ be a (finite or infinite) set of objects. Each object $x \in \mathcal{X}$ is associated with a set of d attributes, $a(x) = (a_1, \dots, a_d) \in \mathbb{R}^d$, and a protected attribute $z(x)$ which is either 0 or 1. Abusing the notation we will use x to denote both the object and the numerical attribute vector $a(x)$. We will also use z_x to denote the sensitive attribute value for x. We will sometimes refer to the protected attribute as color, which takes values *red* and *blue*, corresponding to the values 0 and 1. We will assume that the red color, or the value 0, is the minority, protected, or underrepresented group for which we want to establish a fair treatment.

We also assume that each point $x \in \mathcal{X}$ is associated with a *class label*, $c(x)$. We assume that the class label also takes values 0 and 1. We assume that class 1 corresponds to a positive outcome for the point x, while class 0 to a negative outcome. Using the class labels we can train a classification model on a subset of $\mathcal{X}$, that can be used for predicting the class labels of new points. In this work we will consider the k-NN classification model. We will study the *fairness* of the k-NN model.

We assume the presence of three sets $\mathcal{T}, \mathcal{V}, \mathcal{S} \subset \mathcal{X}$: The set $\mathcal{T}$ is the *training set* that we will use for training the classification model. The set $\mathcal{V}$ is the *validation set* that will be used for calibrating the model to achieve fairness. The set $\mathcal{S}$ is the *test set* that will be used for testing the fairness and accuracy of the fair model. We will often assume that the sets $\mathcal{T}, \mathcal{V}, \mathcal{S}$ are drawn from an unknown random distribution $\mathcal{D}$ over $\mathcal{X}$.

Let $\mathcal{M}_k^{\mathcal{T}}$ be the k NN-model, trained on training data $\mathcal{T}$. We will sometimes omit the subscript and/or superscript when they are implied. The k-NN model is a lazy-learner, or a non-inductive classifier, since it uses the training data directly to classify new instances rather than creating an inductive model. Specifically, during training, the appropriate structures are created so that given a new point $x \in \mathcal{X}$, we can retrieve the set $N_{x,k} \subset \mathcal{S}$ with the k nearest neighbors of x in $\mathcal{T}$, according to some distance metric d defined over $\mathcal{X}$. We will assume that d is the Euclidean distance over the numeric vectors representing the objects in $\mathcal{X}$. The model produces a class label $y(x)$ for a point x, which is the majority of the class labels of the points in $N_{x,k}$. For simplicity, we will assume that k is odd, so the majority is well defined. Other ways of dealing with ties are also possible.

The goal is to study fairness of the k-NN classification model. There are several definitions of classification fairness in the literature [27, 3]. These definitions take into account the predicted class labels, the actual labels, and the sensitive attribute value, and rely on certain ratios that should take a "fair" value. In this work we will consider the *Demographic Parity* fairness. Let $P(y(p) = 1|z(p) = b)$ be the ratio of the test instances of type b that take a positive label (assumed to be desired output). Demographic Parity requires that $P(y(p) = 1|z(p) = 0) \approx P(y(p) = 1|z(p) = 1)$. Equivalently, let $\Delta_{DP} = P(y(p) = 1|z(p) = 0) - P(y(p) = 1|z(p) = 1)$ denote the *fairness deficit*. We want to have $\Delta_{DP} \leq \epsilon$. Intuitively, demographic parity requires that both types of objects have equal probability of receiving a positive outcome. For example, if the sensitive attribute is gender, and the class label denotes a positive or negative hiring decision, demographic parity requires that both men and women have equal probability of being hired.

Relaxing the idea of demographic parity, we can also define a version of fairness, where we only focus on the protected group (the red group in our case). In this case, given a parameter ϕ, we require that $P(y(p) = 1|z(p) = 0) \geq \phi$. This corresponds to cases where we want some sort of affirmative action policy, where at least a fraction ϕ of the members of the protected group receive a positive treatment.

6.2. Problem Definitions

The goal is to change the k-NN classifier to make it fair. Given that k-NN is not an inductive classifier, we can make k-NN fair only by altering the training data $\mathcal{T}$. We consider two types of alterations to the training set $\mathcal{T}$. The first, is to *flip* the labels of some of points in $\mathcal{T}$. The second is to *remove* some of points in $\mathcal{T}$. Other alterations could also be considered, such as adding or displacing points in the dataset.

We will achieve fairness by increasing the number of positive decisions for the protected (red) group (rather than decreasing the positive outputs for the non-protected (blue) group). Therefore, any alterations should focus on the negatively labeled points in the training dataset. Flipping, or removing a positive training instance cannot help us in increasing the positive outputs. We will use $\mathcal{T}_0 = \{x \in \mathcal{T} : c(x) = 0\}$ to denote the negatively labeled part of the training set. These are the points that our alterations will target.

The k-NN model is trained on the training set $\mathcal{T}$. The fairness of the model $\mathcal{M}_k^{\mathcal{T}}$ is measured on the validation set $\mathcal{V}$. The goal is to perform alterations on the training set $\mathcal{T}$, to obtain a new training set $\mathcal{T}'$, such that the model $\mathcal{M}_{\mathcal{T}',k}$ satisfies some fairness requirement.

We will now give two generic problem definitions that we will specify further later on.

Problem 4 (Min-Alterations:) *Given a training set $\mathcal{T}$, and the corresponding model $\mathcal{M}_{\mathcal{T},k}$, perform the minimum set of alterations on $\mathcal{T}$, such that the model $\mathcal{M}_k^{\mathcal{T}'}$ on the altered training set is fair on validation set $\mathcal{V}$.*

Problem 5 (Budgeted-Alterations:) *Given a training set $\mathcal{T}$, and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, and a budget of alterations B, perform B alterations on $\mathcal{T}$, such that the model $\mathcal{M}_k^{\mathcal{T}'}$ on the altered training set maximizes the* fairness gain *on validation set $\mathcal{V}$.*

Fairness is defined with respect to the positive decisions for the protected and non-protected groups on the validation set $\mathcal{V}$. For the following let $RPR_{\mathcal{V}}(\mathcal{M})$ denote the fraction of red (protected) group points in $\mathcal{V}$ that receive a positive outcome by model $\mathcal{M}$ Red Positive Ratio), and $BPR_{\mathcal{V}}(\mathcal{M})$ to denote the corresponding ratio for the blue (non-protected) group. We will use simply RPR (BPR), to denote the red (blue) positive ratio for the model $\mathcal{M}_k^{\mathcal{T}}$, and RPR' (BPR'), to denote the red (blue) positive ratio for the model $\mathcal{M}_k^{\mathcal{T}'}$. The fairness deficit is defined Δ_{DP} and Δ'_{DP} respectively.

Depending on the type of alteration we consider, and the fairness definition we employ, (demographic parity (DP), or ϕ-parity, ϕP), we obtain the following eight problem definitions.

Problem 6 (Min-Flip-ϕP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, find the smallest set of points $S \subset \mathcal{T}_0$ to flip, such that the $RPR' \geq \phi$.*

Problem 7 (Budgeted-Flip-ϕP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, and a budget B, find a subset of B points $S \subset \mathcal{T}_0$ to flip, such that the RPR' is maximized.*

Problem 8 (Min-Flip-DP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, find the smallest set of points $S \subset \mathcal{T}_0$ to flip, such that the $\Delta'_{DP} \leq \epsilon$.*

Problem 9 (Budgeted-Flip-DP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, and a budget B, find a subset of B points $S \subset \mathcal{T}_0$ to flip, such that the fairness gain $\Delta_{DP} - \Delta'_{DP}$ is maximized.*

Problem 10 (Min-Remove-ϕP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, find the smallest set of points $S \subset \mathcal{T}_0$ to remove, such that the $RPR' \geq \phi$.*

Problem 11 (Budgeted-Remove-ϕP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, and a budget B, find a subset of B points $S \subset \mathcal{T}_0$ to remove, such that the RPR' is maximized.*

Problem 12 (Min-Remove-DP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, find the smallest set of points $S \subset \mathcal{T}_0$ to remove, such that the $\Delta'_{DP} \leq \epsilon$.*

Problem 13 (Budgeted-Remove-DP:) *Given a training set $\mathcal{T}$ and the corresponding model $\mathcal{M}_k^{\mathcal{T}}$, and a budget B, find a subset of B points $S \subset \mathcal{T}_0$ to remove, such that the fairness gain $\Delta_{DP} - \Delta'_{DP}$ is maximized.*

6.3. Algorithms

We now consider algorithmic solutions for our problems. For the following let $\mathcal{V}_R = \{x \in \mathcal{V} : z(x) = 0\}$ denote the red (protected) group points in $\mathcal{V}$, and $\mathcal{V}_R^- = \{x \in \mathcal{V}_R : y(x) = 0\}$ denote the protected group points in $\mathcal{V}$ that receive a negative decision by the model $\mathcal{M}_k^{\mathcal{T}}$. The set $\mathcal{V}_R^+$ contains the positively classified instances in $\mathcal{V}_R$. The values $\mathcal{V}_B$, $\mathcal{V}_B^-$ $\mathcal{V}_B^+$ are defined similarly. When we need to specify the model used for the classification, we will use $\mathcal{V}_R^-(\mathcal{M})$. We will also use $\mathcal{M}'$ to denote the model $\mathcal{M}_k^{\mathcal{T}'}$.

The first observation is that the alterations in the training set we consider, can only have a positive effect on the classification results. That is, when flipping or removing a negative point from $\mathcal{T}_0$, we can only move points from $\mathcal{V}_R^-$ to $\mathcal{V}_R^+$, and from $\mathcal{V}_B^-$ to $\mathcal{V}_B^+$, and never the other way round. Therefore, the alterations monotonically increase RPR and BPR.

When looking at ϕP-fairness, we define the *gain* of an alteration as the increase in RPR which is essentially defined by the increase in the size of $\mathcal{V}_R^+$, that is, the number of red points in the validation set that were classified as negative, which are now classified as positive. Formally, the gain for an alteration involving point $t \in \mathcal{T}_0$ (flipping or removing) is defined as $gain_R(t) = \frac{|\mathcal{V}_R^+(\mathcal{M}')| - |\mathcal{V}_R^+(\mathcal{M})|}{|\mathcal{V}_R|}$. Abusing the notation we also define similarly $gain_R(S)$ for a set of points $S \subseteq \mathcal{T}_0$. Note that in order to achieve ϕP-fairness, we need $gain_R(S) \geq \theta_\phi$, where $\theta_\phi = \phi - \frac{|\mathcal{V}_R^+(\mathcal{M})|}{|\mathcal{V}_R|}$. The gain $gain_B(S)$ for the blue points is defined similarly.

When looking at DP-fairness, we define the *gain* of an alteration as the decrease of the fairness deficit $\Delta(\mathcal{M})$. Formally, the gain for an alteration involving point $t \in \mathcal{T}_0$ (flipping or removing) is defined as $gain_\Delta(t) = \Delta(\mathcal{M}) - \Delta(\mathcal{M}') = gain_R(t) - gain_B(t)$. The gain for a set of points S is defined as $gain_\Delta(S)$. To achieve fairness we want $gain_\Delta(S) \geq \theta_{DP}$, where $\theta_{DP} = \Delta(\mathcal{M}) - \epsilon$.

6.3.1. Flipping labels

We now consider the case where we select training points from $\mathcal{T}_0$ and we flip their label from negative to positive.

The 1-NN case

When the Nearest Neighbor classifier uses only $k = 1$ nearest neighbor for making a decision, then $\mathcal{X}$ is partitioned according to the Voronoi diagram defined by the $\mathcal{T}$. The validation set $\mathcal{V}$ is also partitioned accordingly, and each point $x \in \mathcal{V}$ is classified according to the label of the center of the Voronoi cell in which it falls. For a point $t \in \mathcal{T}_0$ let $C_t \subset \mathcal{X}$ denote the Voronoi cell of t, and $R_t^- = \{x \in \mathcal{V}_R^- : x \in C_t\}$ denote the set of negatively classified red points that fall in the Voronoi cell of t, and let $B_t^- = \{x \in \mathcal{V}_B^- : x \in C_t\}$ denote the negatively classified blue points that fall in the Voronoi cell of t.

Given that for any two points $t, t' \in \mathcal{T}_0$, $C_t \cap C_{t'} = \emptyset$, the points affected by flipping a point $t \in \mathcal{T}_0$ are independent of the flipping of any other point in $\mathcal{T}_0$. Therefore, the algorithm in this case for both Min-Flip-ϕP and Budgeted-Flip-ϕP is simple: Sort the points in $\mathcal{T}_0$ according to gain ($gain_R$ or $gain_\Delta$). Select the points to flip in this order until the fairness criterion has been achieved. This algorithm is polynomial and it is optimal.

Note: We assume here that our goal is to get $RPR \geq \phi$ and $\Delta_{DP} \leq \epsilon$, not $RPR = \phi$, or $|\Delta_{DP}| \leq \epsilon$, so we are allowed to overshoot. Otherwise, we probably have a subset sum problem which is NP-hard.

The $k > 1$ case

For the following we assume that k is odd, to avoid dealing with the ties. This problem is likely NP-hard. For a point $x \in \mathcal{V}$ let $D_x^k \subset \mathcal{T}$ denote the k-NN neighborhood of x in the training set. Assume that $k = 3$ and our dataset is such that each point $x \in \mathcal{V}_R^-$ has one positive point in the D_x^3, and two negative points. To change the label of x, we need to flip at least one of the negative points. The MIN-FLIP-ϕP problem then corresponds to the Min Edge Cover problem, while the BUDGETED-FLIP-ϕP to a Maximum Edge Coverage problem. We could probably construct a reduction based on this. In general we have a multi-coverage problem, which is NP-hard and does not have a good approximation ratio. (**Note:** We need to have reductions for this problem, as well as for the next cases).

We propose the following heuristic algorithm for the MIN-FLIP-ϕP problem. For each point $x \in \mathcal{V}_R^-$ compute the D_x^k neighborhood of x. Also compute a counter f_x of the number of points in D_x^k that need to be flipped in order for x to become positive. For each point $t \in \mathcal{T}_0$ compute $N_t^R = \{x \in \mathcal{V}_R^- : t \in D_x^k\}$, the set of negative red points affected by t. We iteratively select points $t \in \mathcal{T}_0$ to flip. Every time we flip a point we update the counters. When a counter becomes zero we remove the point from $\mathcal{V}_R^-$.

For the selection of the next point $t \in \mathcal{T}_0$ to be flipped we use the following two rules:

1. Select the point t with the largest set N_t^R.
2. For each point $x \in \mathcal{V}_R^-$ compute a weight $w_x = 1/f_x$. For the point t compute $W_t^R = \sum_{x \in \mathcal{V}_R^-} w_x$. Select the point t with maximum W_t^R.

We could modify the same idea for the DP fairness. We can compute the set N_t^B similar to N_t^R and select the point with the largest difference $|N_t^R| - |N_t^B|$, or compute the weight W_t^B and select the point with largest difference $W_t^R - W_t^B$. We can also use $gain_\Delta$ as a guide for the point selection.

6.3.2. Removing points

When removing a point $t \in \mathcal{T}_0$, we affect all the points that are in the neighborhood N_t of the point t. Let $x \in \mathcal{V}$ be one such point, and let D_x be the neighbors of x sorted in increasing distance from x. Let $K_x = D_x^k$ denote the neighbors of x the determine the classification of x. Removing t will result in a new neighborhood D'_x and new set K'_x.

Since the point x is classified negatively, this means that the set D_x^k contains less than $\lceil \frac{k}{2} \rceil$ positive training points. Let p be the order of the nearest neighbor in D_x such that D_x^p contains $\lceil \frac{k}{2} \rceil$ positive points. Let $P_x = D_x^p \cap T_0$ be the negative points in D_x^p. For the point x to be classified as positive we need to remove $d_x = p - \lfloor \frac{k}{2} \rfloor$ negative points from P_x.

Given a collection of points $S \subseteq \mathcal{T}_0$ to be removed, let $F_R(S) = \{x \in \mathcal{V}_R^- : P_x \cap S \geq d_x\}$ be the set of red points that will change classification. The gain for removing the set S is $gain_R(S) = \frac{|F_R(S)|}{|\mathcal{V}_R|}$. The MIN-REMOVE-$\phi P$ problem can now be formulated as follows: Find the smallest subset $S \subseteq \mathcal{T}$ such that $|F_R(S)| \geq \Theta_\phi$, where $\Theta_\phi = \theta_\phi |\mathcal{V}_R|$.

The BUDGETED-REMOVE-ϕP can be formulated as follows: Find a subset $S \subseteq \mathcal{T}$ of size B, such that $|F_R(S)|$ is maximized.

In the case of DP-fairness, we define $gain_B(S) = \frac{|F_B(S)|}{|\mathcal{V}_B|}$ where $F_B(S)$ is defined similarly to $F_R(S)$. Recall that $gain_\Delta(S) = gain_R(S) - gain_B(S)$. Therefore, to maximize $gain_\Delta$, we will find S that maximizes $|F_R(S)||\mathcal{V}_B| - |F_B(S)||\mathcal{V}_R|$. The Min-Remove-DP problem can now be formulated as follows: Find the smallest subset $S \subseteq \mathcal{T}$ such that $|F_R(S)||\mathcal{V}_B| - |F_B(S)||\mathcal{V}_R| \geq \Theta_{DP}$, where $\Theta_{DP} = \theta_{DP}|\mathcal{V}_R||\mathcal{V}_B|$.

The Budgeted-Remove-DP can be formulated as follows: Find a subset $S \subseteq \mathcal{T}$ of size B, such that $|F_R(S)||\mathcal{V}_B| - |F_B(S)||\mathcal{V}_R|$ is maximized.

The 1-NN case We will now consider in detail the case where $k = 1$. In this case, to change the classification of a point x we need to remove the full set P_x of negative training points. Therefore, $F_R(S) = \{x \in \mathcal{V}_R^- : P_x \subseteq S\}$. Note that we can view the sets P_x as hyperedges of a graph defined over the set of nodes $\mathcal{T}_0$. Then, the Budgeted-Remove-ϕP problem asks for a set of B nodes, that maximizes the hyperedge density. This is the densest B-subhypergraph problem [8], which is known to be NP-hard. The best known approximation is $O(n^{0.7})$ [8].

The work in [8] shows that the densest B-subhypergraph problem can be solved optimally in polynomial time for interval graphs, using Dynamic Programming. An interval hypergraph $H = (V, E)$ is a hypergraph where $V \subset \mathbb{N}$, and the for each edge $e \in E$, there are $a_e, b_e \in \mathbb{N}$, such that $e = \{i \in V : a_e \leq i \leq b_e\}$. We can apply this algorithm for the Budgeted-Remove-ϕP problem in the case of one-dimensional data. In this case the points in the sets $\mathcal{T}$ and $\mathcal{V}$ are numbers on the real line. Let $U = \mathcal{T} \cup \mathcal{V}$. We sort U in increasing order, and map each real number x to a natural number $\eta(x)$, where $\eta(\min U) = 1$ and $\eta(\max U) = |U|$. Let $V = \eta(U) = \{\eta(x) : x \in U\}$. For a value $x \in \mathcal{V}$, let $\eta(P_x) = \{\eta(y) : y \in P_x\}$ and $E = \{\eta(P_x) : x \ \in \mathcal{V}\}$. The pair $H = (E, V)$ defines a hypergraph, since the values in P_x are values closest to x until a negatively labeled value is found.

The Min-Remove-ϕP problem can be mapped to the minimum k-union problem (min-k-union) [8]. Let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be a family of m sets over a universe of n items, and let k be an integer parameter. The min-k-union problem asks for a family of k sets $C \subseteq \mathcal{S}$ such that $|\bigcup_{S_j \in C} \S_j|$ is minimized. For our problem, k is the required points in $\mathcal{V}_R^-$ that we need to make positive so that we have fairness. The collection $\mathcal{S} = \{P_x : x \in \mathcal{T}_0\}$ is the collection of the P_x sets. The union of the selected P_x's is the set of training points that need to be removed to achieve ϕ-parity.

The min-k-union problem is NP-hard. There is a $O(\sqrt{m})$-approximation algorithm [8], which was improved to $O(m^{1/4+\varepsilon})$-approximation [9]. There is also a known $m^{1/4+\varepsilon}$-approximation lower bound [9].

Given the hardness of the problem, we plan to explore the heuristic we described for the Flipping case to the Removal case as well. For each point $t \in \mathcal{T}_0$, for each $x \in \mathcal{V}_R^-$, we can compute the fraction of the (current) P_x that the point corresponds to. Summing over all points in $\mathcal{V}_R^-$, we obtain the weight of the point t. We can then greedily remove points until we achieve fairness.

7. Conclusion - State of Affairs

In this report we defined algorithms removing bias and achieving fairness in different Machine Learning and Data Mining problems. Specifically, we designed fair algorithms for the following problems:

1. **Community detection and clustering:** We have implemented and published our algorithms for modularity-fair community detection [16] and fair label propagation [23]. We are in the process of exploring extensions of these algorithms in different settings. The algorithms for modularity fair spectral and deep community detection algorithms have been submitted for publication to ICDM 2025. We are in the process of implementing and experimenting with physics-based algorithms for clustering numerical data.

2. **Opinion formation processes:** We have implemented and experimented with the algorithms for achieving fairness in opinion formation, and we have submitted a paper for publication to ICDM 2025.

3. **Pagerank:** We have implemented and experimented with with the algorithms for achieving

Pagerank fairness at minimum cost, and we have submitted a paper for publication to KDD 2026.

4. ***k*-NN classification:** We are currently in the process of implementing and experimenting with the heuristics for achieving fairness in k-NN classification. There is an implementation for the case of flipping labels, and we are working on an implementation for the case of point removal. This work is in collaboration with team member Stavros Sintos and his students. We plan to submit a paper for publication to SIGMOD 2026.

References

[1] Rediet Abebe et al. "Opinion Dynamics Optimization by Varying Susceptibility to Persuasion via Non-Convex Local Search". In: *ACM Trans. Knowl. Discov. Data* (2021).

[2] Josh Alman et al. "More Asymmetry Yields Faster Matrix Multiplication". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025.* Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 2005–2039. DOI: 10.1137/1.9781611978322.63. URL: https://doi.org/10.1137/1.9781611978322.63.

[3] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning: Limitations and Opportunities.* MIT Press, 2023.

[4] D. Bertsekas. *Nonlinear Programming.* Athena Scientific, 2016. ISBN: 9781886529052.

[5] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 10 (2008).

[6] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: https://doi.org/10.1016/S0169-7552(98)00110-X. URL: https://www.sciencedirect.com/science/article/pii/S016975529800110X.

[7] Flavio Chierichetti et al. "Fair clustering through fairlets". In: *Advances in neural information processing systems* (2017).

[8] Eden Chlamtác et al. "The densest k-subhypergraph problem". In: *SIAM Journal on Discrete Mathematics* 32.2 (2018), pp. 1458–1477.

[9] Eden Chlamtá, Michael Dinitz, and Yury Makarychev. "Minimizing the union: Tight approximations for small set bipartite vertex expansion". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM. 2017, pp. 881–899.

[10] A. Clauset, M. E. J. Newman, and C. Moore. "Finding community structure in very large networks". In: *Phys. Rev. E* 70 (6 2004).

[11] Cynthia Dwork et al. "Fairness through awareness". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference.* ITCS '12. 2012, pp. 214–226.

[12] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World.* Cambridge University Press, 2010. ISBN: 9781139490306.

[13] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* 486.35 (Feb. 2010), pp. 75–174. ISSN: 0370-1573.

[14] Noah E. Friedkin and Eugene C. Johnsen. "Social influence and opinions". In: *The Journal of Mathematical Sociology* 15.3-4 (1990), pp. 193–206. DOI: 10.1080/0022250X.1990.9990069.

[15] Aristides Gionis, Evimaria Terzi, and Panayiotis Tsaparas. *Opinion Maximization in Social Networks.* 2013. arXiv: 1301.7455 [cs.SI]. URL: https://arxiv.org/abs/1301.7455.

[16] Christos Gkartzios, Evaggelia Pitoura, and Panayiotis Tsaparas. "Fair Network Communities through Group Modularity". In: *Proceedings of the ACM Web Conference (WWW '25).* 2025.

[17] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 2013. ISBN: 9781421407944.

[18] M. E. J. Newman. "Fast algorithm for detecting community structure in networks". In: *Phys. Rev. E* 69 (2004).

[19] M. E. J. Newman. "Finding community structure in networks using the eigenvectors of matrices". In: *Phys. Rev. E* 74 (3 Sept. 2006), p. 036104. DOI: 10.1103/PhysRevE.74.036104.

[20] M. E. J. Newman. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23 (2006), pp. 8577–8582. DOI: 10.1073/pnas.0601602103. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0601602103. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0601602103.

[21] J. Nocedal and S. Wright. *Numerical Optimization.* Springer New York, 2000. ISBN: 9780387987934.

[22] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks". In: *Phys. Rev. E* 76 (3 2007), p. 036106.

[23] Glykeria Toulina and Panayiotis Tsaparas. "A Fair Label Propagation Community Detection Algorithm". In: 2025.

[24] Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. "From Louvain to Leiden: guaranteeing well-connected communities". In: *Scientific Reports* 9.1 (2019), p. 5233.

[25] Sotiris Tsioutsiouliklis et al. "Fairness-Aware PageRank". In: *Proceedings of the Web Conference 2021.* 2021, pp. 3815–3826. ISBN: 9781450383127.

[26] Anton Tsitsulin et al. "Graph clustering with graph neural networks". In: *J. Mach. Learn. Res.* 24.1 (2023).

[27] Sahil Verma and Julia Rubin. "Fairness definitions explained". In: *Proceedings of the International Workshop on Software Fairness.* FairWare '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 1–7. ISBN: 9781450357463. DOI: 10.1145/3194770.3194776. URL: https://doi.org/10.1145/3194770.3194776.

[28] Virginia Vassilevska Williams. "Multiplying matrices faster than coppersmith-winograd". In: *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing.* 2012.

[29] Donghui Yan, Ling Huang, and Michael I. Jordan. "Fast approximate spectral clustering". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 907–916.

[30] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social Media Mining: An Introduction.* Cambridge University Press, 2014.