Finding coherent node groups in directed graphs

Iiro Kumpulainen and Nikolaj Tatti

HIIT, University of Helsinki, Helsinki, firstname.lastname@helsinki.fi

Abstract. Summarizing a large graph by grouping the nodes into clusters is a standard technique for studying the given network. Traditionally, the order of the discovered groups does not matter. However, there are applications where, for example, given a directed graph, we would like to find coherent groups while minimizing the backward cross edges. More formally, in this paper, we study a problem where we are given a directed network and are asked to partition the graph into a sequence of coherent groups while attempting to conform to the cross edges. We assume that nodes in the network have features, and we measure the group coherence by comparing these features. Furthermore, we incorporate the cross edges by penalizing the forward cross edges and backward cross edges with different weights. If the weights are set to 0, then the problem is equivalent to clustering. However, if we penalize the backward edges significantly more, then the order of discovered groups matters, and we can view our problem as a generalization of a classic segmentation problem. To solve the algorithm we consider a common iterative approach where we solve the groups given the centroids, and then find the centroids given the groups. We show that—unlike in clustering—the first subproblem is **NP**-hard. However, we show that if the underlying graph is a tree we can solve the subproblem with dynamic programming. In addition, if the number of groups is 2, we can solve the subproblem with a minimum cut. For the more general case, we propose a heuristic where we optimize each pair of groups separately while keeping the remaining groups intact. We also propose a greedy search where nodes are moved between the groups while optimizing the overall loss. We demonstrate with our experiments that the algorithms are practical and yield interpretable results.

1 Introduction

Summarizing a large graph by grouping the nodes into clusters is a standard technique for studying networks. While many techniques have been proposed for clustering undirected graphs, directed graphs pose additional challenges.

On the other hand, much data can be naturally represented using directed networks such as discussion threads in social media platforms or a citation graph. In addition to edges we also typically have additional information attached to the nodes, typically expressed as categorical labels or real-valued features. These features allow us to measure the similarity of the nodes, which in turn allows us to cluster similar nodes together. When clustering nodes we would like to take edges into account. For example, given a citation graph, our goal is to partition nodes into similar groups such that one group cites the other. Another example is a discussion thread where our goal is to group early messages in one group and replies (and the following replies) in the other group.

We consider discovering ordered partitions in a directed graph. That is, given a graph, our goal is to divide the vertices in a *sequence* of k groups such that each group is as coherent as possible while (backward) cross edges are minimized.

We focus on using L_2 error though our approach will work on any centroid-based objective. The **NP**-hardness of clustering immediately implies the hardness of our problem. In order to solve our problem we consider two approaches.

The first approach is a greedy search where we decrease the cost by moving the vertices from one cluster to another. We show that by using a common L_2 decomposition we can run a single iteration in $\mathcal{O}(k(nd + m))$ time, where n and m are the numbers of nodes and edges, and d is the number of features.

We also propose an iterative approach where we fix centroids and optimize partition, and then fix partition and optimize centroids. Unlike with k-means algorithm, finding an optimal partition for fixed centroids is **NP**-hard. We then consider two common special cases. We show that if the input graph is a tree, we can find the partition with dynamic programming in $\mathcal{O}(dn)$ time. We also show that if $k = 2$, we can find the partition with a minimum cut in $\mathcal{O}(n(d + m))$ time. For a general case, we propose an algorithm that enumerates all pairs of groups and optimizes them using minimum cut while keeping the remaining groups fixed.

2 Preliminary notation and problem definition

We start by establishing the notation that we will use throughout the paper and define our main optimization problem.

We assume that we are given a directed graph $G = (V, E)$, where V is the set of vertices, and E is the set of edges between vertices. We typically define n to be the number of vertices $|V|$ and m to be the number of edges $|E|$. Assume two disjoint set of vertices A and B. We will write $E(A, B) = \{e = (v, w) \in E \mid v \in A, w \in B\}$ to be the edges from A to B.

We assume that we are given a function $H : 2^V \to \mathbb{R}$ that measures the coherency of a vertex set. We are particularly interested in using L_2 error as a measure of coherence. More specifically, assume that we have a map $a : V \to \mathbb{R}^D$ that maps a vertex v to a real-valued vector of D features $a(v)$. Then the measure is $L_2(S) = \min_\mu \sum_{v \in S} \|a(v) - \mu\|_2^2$, where μ is the centroid $\mu = \frac{1}{|S|} \sum_{v \in S} a(v)$.

Our goal is to partition the graph into a sequence of k groups that are at the same time coherent and minimize the cross-edges. In order to measure the cost of such a partition, we introduce two weight parameters λ_f and λ_b for the forward and backward edges, respectively. Given an ordered partition $\mathcal{S} = S_1, \ldots, S_k$, we define a cost function q as

$$q(\mathcal{S} \mid \lambda_f, \lambda_b, H) = \sum_{i=1}^{k} H(S_i) + \sum_{j=i+1}^{k} \lambda_f |E(S_i, S_j)| + \lambda_b |E(S_j, S_i)| \quad .$$

We often drop H, λ_f, or λ_b from the notation if they are clear from the context.

The definition of q leads immediately to our main optimization problem

Problem 1 (directed graph segmentation (DGS)). Given a directed graph $G = (V, E)$, integer k, two weights λ_f and λ_b, and a function $H : 2^V \to \mathbb{R}$, find an ordered k-partition $\mathcal{S} = S_1, \ldots, S_k$ of V such that $q(\mathcal{S} \mid \lambda_f, \lambda_b, H)$ is minimized.

Note that if we set $\lambda_b = \lambda_f$, then the order of sets in $\mathcal{S}$ does not matter. Moreover, if $\lambda_b = \lambda_f = 0$, then DGS reduces to a clustering problem. Especially, if we set H to L_2, then the optimization problem is equal to k-means. Our main interest is to study cases when $\lambda_f = 0$ and λ_b is large, possibly infinite.

3 Related work

Clustering is a staple method in supervised learning with k-means problem (see [8], for example) being the most common optimization problem. The **NP**-hardness of clustering, even in the plane [15], makes our problem immediately **NP**-hard when we set $\lambda_f = \lambda_b = 0$ and H to be L_2 loss.

The closest framework to our problem setting is pairwise constrained clustering (POC), where selected pairs of data points must be in the same cluster or must belong to different clusters [7, 27]. Other constraints such as balancing constraints or minimum-size constraints have also been studied; we refer the reader to [2] for more details. The key technical difference is that in POC the constraints have no direction. Consequently, the order of the resulting clusters does not matter. However, in our case, if $\lambda_f \neq \lambda_b$ the order of groups matters, especially if we set $\lambda_b = \infty$ and $\lambda_f = 0$.

We can also view our problem as a directed network clustering problem. Undirected graph clustering has been well-studied. Popular methods include minimizing modularity [18] as well as stochastic blockmodelling [1], spectral clustering [26], or closely related normalized cuts [17]. We refer the reader to [9, 22] for surveys on undirected graph clustering.

The clustering of directed graphs poses additional challenges, as measures need to be adapted. Leicht and Newman [14] proposed a modularity measure for directed graphs. Chung [5] proposed a Laplacian matrix for directed graphs allowing the use of spectral clustering. Moreover, a random-walk approach was proposed by Rosvall and Bergstrom [21]. We refer the reader to [16] for a survey on the clustering of directed graphs. The main difference between graph clustering and our problem is that graph clustering methods focus on optimizing measures based solely on edges. In contrast, we use additional information, for example, L_2 error over the features while also minimizing the number of cross edges.

An interesting special case of our problem occurs when the underlying graph is a directed path, and we set the backward weight to $\lambda_b = \infty$. In such a case, the clusters will respect the order of the vertices, and DGS reduces to a segmentation problem, in which we are given a *sequence* of points and are asked to segment the sequence into k coherent groups. Segmentation can be solved with

Algorithm 1: Iterative algorihm

1 **for** $v \in V$ **do** assign v to a set S_i randomly;
2 **while** *the loss is decreasing or until a set amount of iterations* **do**
3 compute the centroids $\mu_i = \frac{1}{|S_i|} \sum_{v \in S_i} a(v)$ for $i = 1, \ldots, k$;
4 optimize $\mathcal{S}$ minimizing $q(\mathcal{S}, M \mid \lambda_f, \lambda_b)$ while the centroids $M = \{\mu_i \mid i = 1, \ldots, k\}$ remain fixed;
5 **return** sets $S_1, \ldots, S_k$;

dynamic programming in quadratic time [3] and can be efficiently approximated in quasilinear time [10] or linear time [11, 25].

Finally, let us point out an interesting connection to isotonic regression [13]. Assume that the underlying graph is a DAG. If we set $\lambda_b = \infty$ and $\lambda_f = 0$, use L_2 error, and additionally require that the L_2 norms of the centroids need to be monotonically increasing $\|\mu_{i-1}\|_2 < \|\mu_i\|_2$. Then we can show that the optimization problem can be solved in polynomial time by first applying isotonic regression, ordering the nodes by the obtained mapping, and segmenting the nodes in k segments using dynamic programming [3].

4 Iterative algorithm for L_2 error

Let us now focus on using L_2 error as the measure of coherency. A standard algorithm to solve the k-means problem is to iteratively fix centroids and optimize the partition and then optimize centroids while keeping the partition fixed.

In order to adapt this idea to our approach, assume that we are given an ordered k-partition $\mathcal{S}$ of vertices V and k centroids. Let us define the cost

$$q(\mathcal{S}, \{\mu_i\} \mid \lambda_f, \lambda_b) = \sum_{i=1}^{k} \sum_{v \in S_i} \|a(v) - \mu_i\|_2^2 + \sum_{j=i+1}^{k} \lambda_f |E(S_i, S_j)| + \lambda_b |E(S_j, S_i)| \quad .$$

We will then consider the two related sub-problems: in the first, we optimize the partition while keeping the centroid fixed while in the second we optimize the centroids while keeping the partition fixed, as given in Algorithm 1.

*Problem 2 (*DGS-PARTITION*).* Given a directed graph $G = (V, E)$, integer k, two weights λ_f and λ_b, and k centroids $\mu_1, \ldots, \mu_k$, find an ordered k-partition $\mathcal{S} = S_1, \ldots, S_k$ of V such that $q(\mathcal{S}, \{\mu_i\} \mid \lambda_f, \lambda_b)$ is minimized.

*Problem 3 (*DGS-CENTROID*).* Given a directed graph $G = (V, E)$, integer k, two weights λ_f and λ_b, and an ordered k-partition $\mathcal{S} = S_1, \ldots, S_k$ of V, find k centroids $\mu_1, \ldots, \mu_k$ such that $q(\mathcal{S}, \{\mu_i\} \mid \lambda_f, \lambda_b)$ is minimized.

Note that DGS-CENTROID has an analytical solution, $\mu_i = \frac{1}{|S_i|} \sum_{v \in S_i} a(v)$. However, DGS-PARTITION is an **NP**-hard problem.

Theorem 1. DGS-PARTITION *is* ***NP****-hard.*

Proof. We will show that the unweighted minimum multiterminal cut problem (MTC) can be reduced to DGS-PARTITION. MTC is an **NP**-hard problem [6] where we are given a graph and a set of terminals $T = t_1, \ldots, t_k$ and are asked to partition the vertices in k groups $\mathcal{C} = C_1, \ldots, C_k$ such that $t_i \in C_i$ and the number of cross-edges is minimized.

Let $G = (V, E)$ be an undirected instance of MTC with k terminals $T = t_1, \ldots, t_k$. Create an instance of DGS-PARTITION as follows: Set the number of disjoint sets to find as k and set $\lambda_f = \lambda_b = 1$. Define k centroids $\mu_1, \ldots, \mu_k$ to be standard unit vectors of length k such that μ_i has 1 as the ith entry and 0 as all the other entries. Create graph G' as an instance of DGS-PARTITION so that it contains all the vertices in V and each undirected edge in E becomes a directed edge in an arbitrarily chosen direction. Additionally, for each terminal t_i create a set U_i of $|U_i| = |V|$ new vertices that are only connected to t_i. We set the feature vectors $a(t_i) = \mu_i$ and $a(u) = \mu_i$ for any $u \in U_i$. The remaining vertices $v \in V \setminus T$ have $a(v) = 0$.

Let $S_1, \ldots, S_k$ be the solution for DGS-PARTITION.

The cost of including a vertex $u \in U_i$ in S_i is $\|a(u) - \mu_i\|^2 = 0$, while the cost of including u in S_j, for $j \neq i$, is $\|a(u) - \mu_j\|^2 = 2$. It is then optimal to include $u \in U_i$ in S_i as the possible loss of 1 from the edge between the vertex u and t_i is less than the loss of 2 that we would have if u was in another set. Therefore, an optimal solution will include all the vertices in U_i in S_i. This means that the terminal t_i will also have to be in S_i as otherwise, the cost from the edges between t_i and the $|U_i| = |V|$ vertices in U_i will be more than any possible loss from the edges between t_i and any other vertices $v \in V$. Thus, $t_i \in S_i$.

Finally, the remaining non-terminal vertices in $V \setminus T$ have the same loss of 1 regardless of which set they belong to, so an optimal solution will assign them such that the cost arising from the edges between the sets is minimized.

Given a partition $\mathcal{S}$, we define a cut $\mathcal{C}$ for G by setting $C_i = V \cap S_i$. This is a valid cut since $t_i \in S_i$, and since DGS-PARTITION minimizes the number of cross-edges, $\mathcal{C}$ is optimal. □

5 Exact algorithms for special cases

We showed that in general solving DGS-PARTITION is an **NP**-hard problem. However, there are two cases where we can solve the sub-problem in polynomial time: (i) the input graph is a tree or (ii) $k = 2$. In this section, we will consider these two cases. The more general case is discussed in the next section.

We should point out that while we focus on L_2 error this approach works with any error as long as it can be decomposed as a sum over the nodes.

Case when the input graph is a tree. We will first consider a case when the input graph G is a tree. For simplicity, we will assume that G is also arborescence, that is, there is a root vertex, say r, from that connects to each vertex with a *directed* path, but we can extend this approach to trees and forests.

Given an arborescence $G = (V, E)$ and a vertex $v \in V$, let G_v be the subtree containing v and its descendants. We define $c[v, i]$ to be the cost of the optimal partition $\mathcal{S}$ of G_v such that $v \in S_i$. Note that $\min_i c[r, i]$ is equal to the cost of the solution to DGS-PARTITION.

In order to compute $c[v, i]$, let us first define

$$\ell[v, i] = \min_{j<i} c[v, j] \quad \text{and} \quad u[v, i] = \min_{j>i} c[v, j],$$

that is, $\ell[v, i]$ is the cost of the optimal partition $\mathcal{S}$ of G_v such that $v \in S_j$ for some $j < i$. Similarly, $u[v, i]$ is the cost of the optimal partition $\mathcal{S}$ of G_v such that $v \in S_j$ for some $j > i$. For simplicity, we define $\ell[v, 1] = u[c, k] = \infty$.

Next, we compute $c[v, i]$ using only u and ℓ of the children of v.

Theorem 2. *Let c, u, and v be as above. Then for $v \in V$ and $i \in 1, \ldots, k$,*

$$c[v, i] = \left\|a(v) - \mu_i\right\|^2 + \sum_{w \mid (v,w) \in E} \min(c[w, i], \lambda_f + u[v, i], \lambda_b + \ell[v, i]) \quad . \tag{1}$$

Proof. Define for notational simplicity $M = \mu_1, \ldots, \mu_k$. Let $\mathcal{S}$ be the partition responsible for $c[v, i]$ For any child w of v, let us write $\mathcal{S}_w$ to be $\mathcal{S}$ projected to G_w. Let $g(w)$ be the (possibly zero) cost of the possible cross edge (v, w). Since G_v is a tree, we can decompose the cost as

$$q(\mathcal{S}, M) = \left\|a(v) - \mu_i\right\|^2 + \sum_{w \mid (v,w) \in E} g(w) + q(\mathcal{S}_w, M) \quad .$$

Let w be a child of v. We have 3 possible cases. If $w \in S_i$, then $g(w) = 0$ and, due to optimality, $q(\mathcal{S}_w, M) = c[w, i]$. If $w \in S_j$ for $j < i$, then $g(w) = \lambda_b$ and, due to optimality, $q(\mathcal{S}_w, M) = \ell[w, i]$. Similarly, if $w \in S_j$ for $j > i$, then $g(w) = \lambda_f$ and, due to optimality, $q(\mathcal{S}_w, M) = u[w, i]$. Finally, due to the optimality, the actual case will be the one yielding the smallest cost. $\square$

Computing $c[v, i]$ requires $\mathcal{O}(d + \deg v)$ time, where d is the length of the feature vector. Since $\ell[v, i] = \min(\ell[v, i-1], c[v, i])$ and $u[v, i] = \min(u[v, i+1], c[v, i])$ we can compute both quantities in constant time. In summary, we can find the optimal cost in $\mathcal{O}(dn + m) \in \mathcal{O}(dn)$ time. To obtain the corresponding partition we store the indices that were responsible for $c[v, i]$ in Eq. 1.

In summary, if the input graph is a tree, we can search for the partition using Algorithm 1 and solve the sub-problem DGS-PARTITION with dynamic programming. We refer to this algorithm as TREEDP.

Case when $k = 2$. Next, we allow the input graph to be any directed graph but we require that $k = 2$. We will argue that we can then solve DGS-PARTITION using a (weighted) minimum directed cut.

In order to do the mapping assume that we are given a graph $G = (V, E)$ and two centroids μ_1 and μ_2. We define a weighted graph $H = (W, A)$ as follows. The vertices W consist of V and two additional vertices s and t. For each v we introduce an edge (s, v) to A with a weight $c(s, v) = \left\|v - \mu_2\right\|^2$ and an edge (v, t) with a weight $c(v, t) = \left\|v - \mu_1\right\|^2$. For each, $(v, w) \in E$ we add an edge (v, w) with a weight λ_f and an edge (w, v) with a weight λ_b.

The next theorem connects the $s - t$ cut with the cost of the partition.

Algorithm 2: MCUT, iterative local search based on a minimum cut

for $v \in V$ **do** assign v to a set S_i randomly ;
compute the centroids $\{\mu_i\}$ for the current partition;
while *the loss is decreasing or until a set amount of iterations* **do**
  foreach *pair* i, j *with* $1 \leq i < j \leq k$ **do**
    solve DGS-PARTITION(i, j), and update μ_i and μ_j;
return sets $S_1, \ldots, S_k$;

Theorem 3. *Let C_1, C_2 be the s-t cut for H. Let $\mathcal{S} = S_1, S_2$ where $S_i = C_i \cap V$. Then $q(\mathcal{S}, \mu_1, \mu_2)$ is equal to the total weight of edges from C_1 to C_2.*

Proof. The cost of the partition is equal to

$$
\begin{aligned}
q(\mathcal{S}, \mu_1, \mu_2) &= \sum_{v \in S_1} \left\| v - \mu_1 \right\|^2 + \sum_{v \in S_2} \left\| v - \mu_1 \right\|^2 + \sum_{e \in E(S_1, S_2)} \lambda_f + \sum_{e \in E(S_2, S_1)} \lambda_b \\
&= \sum_{v \in S_1} c(v, t) + \sum_{v \in S_2} c(s, v) + \sum_{e \in A(S_1, S_2)} c(e).
\end{aligned}
$$

The sums amount to the total weight of edges from C_1 to C_2. □

The theorem states that we can solve DGS-PARTITION with a minimum cut on H. Solving cut can be done in $\mathcal{O}(nm)$ time [19], though theoretically slower algorithms, e.g., by Boykov and Kolmogorov [4], are faster in practice. Constructing the graph requires $\mathcal{O}(nd + m)$ time, where d is the length of the feature vectors. Thus, we can solve DGS-PARTITION for $k = 2$ in $\mathcal{O}(n(d + m))$ time.

6 Algorithms for the general case

In this section, we consider two algorithms for the general case. The first algorithm is based on the $k = 2$ case, where we iteratively select pairs $i < j$ and optimize S_i and S_j while keeping everything else fixed. The second algorithm is a greedy search where we update partitions by moving one node at a time.

Iterative two-group search. Our first approach, given in Algorithm 2, is based on the special case for $k = 2$. We iterate over all pairs $1 \leq i < j \leq k$ and for each pair, we optimize S_i and S_j while keeping the remaining groups fixed and all the centroids fixed. We will refer to this problem as DGS-PARTITION(i, j) Once S_i and S_j are updated, we update the centroids μ_i and μ_j.

Solving DGS-PARTITION(i, j) is almost the same as solving DGS-PARTITION for $k = 2$. The only main difference is that we need to take into account the cross edges from S_i and S_j to other groups. More formally, we construct the same graph H as in Section 5 except we set the costs

$$
\begin{aligned}
c(s, v) &= \left\| v - \mu_j \right\|^2 + \lambda_f |E(W, v)| + \lambda_b |E(v, W)|, \text{ and} \\
c(v, t) &= \left\| v - \mu_i \right\|^2 + \lambda_b |E(W, v)| + \lambda_f |E(v, W)|, \text{ where } W = S_{i+1} \cup \cdots \cup S_{j-1}.
\end{aligned}
$$

The next result implies that a minimum cut in H solves DGS-PARTITION(i, j).

Algorithm 3: GREEDY, greedy local search

1 **for** $v \in V$ **do** assign v to a set S_i randomly;
2 **while** *the loss is decreasing or until a set amount of iterations* **do**
3 | **for** $v \in V$ **do** find optimal S_j for v; move v to S_j;
4 **return** sets $S_1, \ldots, S_k$;

Theorem 4. *Let $C_1 = S_i \cup \{s\}$ and $C_2 = S_j \cup \{t\}$. Then $q(\mathcal{S}, \{\mu_t\})$ is equal to the total weight of edges from C_1 to C_2 in H.*

The proof is similar to the proof of Theorem 3 and is therefore omitted.

Similar to the case $k = 2$, solving the minimum cut can be done in $\mathcal{O}(nm)$ time, and constructing H requires $\mathcal{O}(dn + m)$ time, where d is the length of the feature vector. Consequently, since there are $k(k-1)/2$ pairs of i, j, a single iteration of the algorithm requires $\mathcal{O}\big(k^2 n(d + m)\big)$ time.

Greedy local search. As a final algorithm (see Algorithm 3), we consider a greedy approach where we start with a random partition and try to improve it by moving individual nodes from one group to another until convergence.

Next, we will prove the running time of the algorithm.

Theorem 5. *A single iteration of Algorithm 3 requires $\mathcal{O}(knd + km)$ time, where d is the length of the feature vectors.*

First, we need to compute the difference of L_2 error in $\mathcal{O}(d)$ time.

Lemma 1. *Let $\mathcal{S}$ be a partition. Select i and $v \in S_i$. Select j and let $\mathcal{S}'$ be the result of moving v from S_i to S_j. Let $\{\mu_t\}$ and $\{\mu'_t\}$ be the corresponding optimal centroids. Let $H(S_i)$ be the L_2 error of S_i and write $H(\mathcal{S}) = \sum H(\mathcal{S}_\rangle)$. Then*

$$H(\mathcal{S}') - H(\mathcal{S}) = |S_i| \left\|\mu_i\right\|^2 + |S_j| \left\|\mu_j\right\|^2 - |S'_i| \left\|\mu'_i\right\|^2 - \left|S'_j\right| \left\|\mu'_j\right\|^2$$

Proof. The identity $\sum_{w \in S_t} \left\|a(w) - \mu_t\right\|^2 = (\sum_{w \in S_t} \left\|a(w)\right\|^2) - |S_t| \left\|\mu_t\right\|^2$ immediately proves the claim. □

Proof (of Theorem 5). Computing the gain of moving v from S_i to S_j requires computing μ'_i and μ'_j which can be done in $\mathcal{O}(d)$ time using μ_i and μ_j. Lemma 1 allows us to compute the cost difference in $\mathcal{O}(d + \deg(v))$ time. Summing over v and j leads to a running time of $\mathcal{O}(knd + km)$. □

7 Experimental evaluation

In this section, we describe our experiments to test the GREEDY, TREEDP, and MCUT algorithms in practice. we evaluate the algorithms first on synthetically constructed graphs and then using two real-world datasets. All data is publicly available, and we make our source code available online.[1]

[1] https://version.helsinki.fi/dacs/coherent-groups-network

Experiments on synthetic data. To test our algorithms, we create two synthetic graphs: a tree graph *STree* and a directed acyclic graph *SDAG*. Each graph consists of 1000 vertices with 10-dimensional features that separate them into 5 clusters. For the tree graph, we have the vertices numbered from 1 to 1000, and for each vertex v except the first, we randomly add an edge from one of the earlier vertices to v. For *SDAG*, we also randomly add an edge between each pair of vertices v_i, v_j with $i < j$ with probability 0.01.

We create the features by first sampling 5 centroids uniformly distributed from $[0, 1]^{10}$. We then assign $v_{200(i-1)+1}, \ldots, v_{200i}$ to a cluster S_i with the ith centroid. Each node then gets a normally distributed feature centred at the centroid of the cluster it belongs to, with a variance of 0.1 in each dimension.

We define the sets $S_1, \ldots, S_5$ as the ground truth, as this partition initially minimizes the L_2 loss within the sets, and there cannot be backward edges between sets but only forward edges from a set S_i to a later set S_j with $i < j$. We also add random noise, by independently, with a probability p, reassigning each node a new normally distributed feature around a random centroid.

We compare the similarity between the ground truth partition $\mathcal{S}$ and the partition $\mathcal{S}'$ returned by our algorithms by computing the Adjusted Rand Index, $ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$. Here $RI(\mathcal{S}, \mathcal{S}') = \frac{a+b}{\binom{n}{2}}$ is the Rand Index, where a is the number of pairs of elements that are in the same set in both $\mathcal{S}$ and $\mathcal{S}'$, and b is the number of pairs of elements that are in different sets in both $\mathcal{S}$ and $\mathcal{S}'$ [12].

To test our algorithms, we run TreeDp on *STree*, Mcut on *SDAG*, and Greedy on both graphs. For each algorithm, we compare the results for the case when $\lambda_f = \lambda_b = 0$, which is equivalent to k-means clustering, to the case when $\lambda_f = 0$, $\lambda_b = 10^5$. The results of the algorithms may vary depending on how the nodes are randomly initialized to random sets and may get stuck in local minima. Therefore, we run each algorithm 10 times with different random initializations and keep the partition that results in the smallest loss.

In Figure 1, we plot ARI between the partition and the ground truth as a function of the probability p of assigning nodes new features.

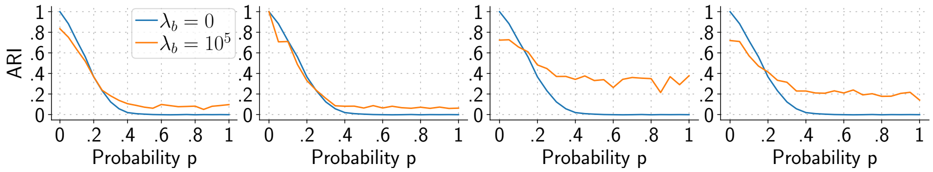
When $\lambda_b = 0$, the ARI starts from 1 but rapidly decreases to 0 as the probability p increases. However, for the $\lambda_b = 10^5$ case, our algorithms return a partition that respects the underlying network structure and remains somewhat similar to the initial ground truth partition. This kind of partitioning is more suitable for practical applications, where the features for individual nodes are noisy or unreliable, and the network structure is more important.

Experiments on real-world data. We perform experiments on two real-world datasets: a tree graph constructed from a discussion on Reddit and a graph constructed from scientific publications from the DBLP[2] dataset [24].

For the *Reddit* dataset, we use the most popular thread on */r/politics*, which discusses the last U.S. presidential election between Joe Biden and Donald Trump[3]. We use the Reddit API to collect the comments in the thread and

[2] `https://www.aminer.org/citation`

[3] `https://www.reddit.com/r/politics/comments/jptq5n/megathread_joe_biden_projected_to_defeat/`



(a) TreeDp(*STree*) (b) Greedy(*STree*) (c) Mcut(*SDAG*) (d) Greedy(*SDAG*)

Fig. 1: Adjusted Rand Index between the ground truth and the partition chosen by the algorithms as a function of the probability p of reassigning vertices a new random features from a random cluster.

Table 1: The number of vertices $|V|$ and the number of edges $|E|$ for the tree graphs, as well as the loss, runtime, and the number of iterations $\#I$ for Greedy and TreeDp.

			Greedy			TreeDp		
Dataset	$\|V\|$	$\|E\|$	Loss	Runtime	$\#I$	Loss	Runtime	$\#I$
Reddit	74 778	74 777	5 455 534.6	2 h 9 min	42	55 220.29	4 h 58 min	75
STree	1 000	999	500 253.44	0.3 s	6	200.26	1.4 s	20

construct a graph where the initial submission and each comment are nodes, and add an edge from each node to the comments responding to it.

To obtain the feature vectors from text, we use an LLM MPNet [23], to convert the text for each node into a 768-dimensional vector. We chose the all-mpnet-base-v2 language model for creating the feature vectors, as its sentence embeddings achieved the best performance for general-purpose tasks out of the HuggingFace sentence-transformer models [20].

For the *DBLP* dataset, we chose publications from ECMLPKDD, ICDM, KDD, NIPS, SDM, and WWW conferences. For each publication, we add an edge to those publications that are citing it. To create the features for each node, we again use the MPNet model to convert the titles of the scientific publications into sentence embeddings that we use as the feature vectors.

We set the number of clusters to 20 and set $\lambda_f = 0$ and $\lambda_b = 100000$. We compare TreeDp and Greedy on *Reddit* and use *DBLP* to compare Mcut in Table 1, and Greedy in Table 2. In addition, we compare the algorithms using the synthetic *STree* and *SDAG* datasets with the number of clusters as 5 and the probability of assigning nodes with features from a random cluster as $p = 0.2$.

In our experiments, the TreeDp and Mcut algorithms find partitions resulting in significantly lower loss than the Greedy algorithm, which gets stuck in a poor local minimum. In general, the number of iterations for each algorithm until convergence is relatively low, except for TreeDp on the *Reddit* dataset, where we halted the execution after 75 iterations while the loss was still decreasing. In particular, on the *DBLP* dataset, the Mcut converges in only 7

Table 2: The number of vertices $|V|$ and the number of edges $|E|$ for *DBLP* and *SDAG*, and the loss, runtime, and the number of iterations $\#I$ for Greedy and Mcut.

			Greedy			Mcut		
Dataset	$\lvert V\rvert$	$\lvert E\rvert$	Loss	Runtime	$\#I$	Loss	Runtime	$\#I$
DBLP	30 581	70 972	28 722 200.72	33 min	22	8 622 678.8	42 min	7
SDAG	1 000	6 060	400 388.59	4.6 s	11	312.07	16 s	7

iterations resulting in a comparable running time to the Greedy algorithm despite individual iterations taking much longer. The running times for the larger graphs *Reddit* and *DBLP* remain practical.

8 Concluding remarks

In this paper, we considered segmenting directed networks by grouping nodes into coherent groups while minimizing (backward) cross edges. We considered two general approaches: The first approach is an iterative algorithm, alternating between fixing the centroids and optimizing the partition, and vice versa. We showed that finding a partition is an NP-hard problem, but that we can find the partition exactly for tree graphs in $\mathcal{O}(dn)$ time, or when the number of groups is two in $\mathcal{O}(n(d+m))$ time. For a more general case, we proposed an algorithm optimizing the nodes between pairs of groups at a time, as well as a greedy local search algorithm. We performed experiments on both synthetic and real-world datasets to demonstrate that the algorithms are practical, finding coherent node groups with a low loss in a feasible number of iterations and running time.

While this paper focused on using the L_2 loss function for measuring the distance between the real-valued feature vectors, our methods could be used together with other types of loss functions. In particular, networks, where nodes have categorical features, provide an interesting line for future work. Another direction for future work would be to consider how more advanced methods for initializing the sets could be applied or developed for our problem, rather than assigning each node to a random set at the beginning of the algorithm.

References

[1] Abbe, E.: Community detection and stochastic block models: recent developments. JMLR 18(1), 6446–6531 (2017)

[2] Basu, S., Davidson, I., Wagstaff, K.: Constrained clustering: Advances in algorithms, theory, and applications. CRC Press (2008)

[3] Bellman, R.: On the approximation of curves by line segments using dynamic programming. Communications of the ACM 4(6), 284–284 (1961)

[4] Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. TPAMI 26(9), 1124–1137 (2004)

[5] Chung, F.: Laplacians and the cheeger inequality for directed graphs. Annals of Combinatorics 9, 1–19 (2005)

[6] Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM Journal on Computing 23(4), 864–894 (1994)

[7] Davidson, I., Ravi, S.: Clustering with constraints: Feasibility issues and the k-means algorithm. In: SDM. pp. 138–149. SIAM (2005)

[8] Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley (2001)

[9] Fortunato, S.: Community detection in graphs. Physics reports 486(3-5), 75–174 (2010)

[10] Guha, S., Koudas, N., Shim, K.: Data-streams and histograms. In: STOC. pp. 471–475 (2001)

[11] Guha, S., Koudas, N., Shim, K.: Approximation and streaming algorithms for histogram construction problems. TODS 31(1), 396–438 (2006)

[12] Hubert, L., Arabie, P.: Comparing partitions. J. Classif. 2, 193–218 (1985)

[13] Kyng, R., Rao, A., Sachdeva, S.: Fast, provable algorithms for isotonic regression in all ℓ_p-norms. NIPS pp. 2719–2727 (2015)

[14] Leicht, E.A., Newman, M.E.: Community structure in directed networks. Physical review letters 100(11), 118703 (2008)

[15] Mahajan, M., Nimbhorkar, P., Varadarajan, K.: The planar k-means problem is np-hard. Theoretical Computer Science 442, 13–21 (2012)

[16] Malliaros, F.D., Vazirgiannis, M.: Clustering and community detection in directed networks: A survey. Physics reports 533(4), 95–142 (2013)

[17] Meilă, M., Pentney, W.: Clustering by weighted cuts in directed graphs. In: SDM. pp. 135–144 (2007)

[18] Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Physical review E 69(2), 026113 (2004)

[19] Orlin, J.B.: Max flows in $O(nm)$ time, or better. In: STOC. pp. 765–774 (2013)

[20] Reimers, N.: SBert sentence-transformers documentation. `https://www.sbert.net/docs/pretrained_models.html` (2022), accessed: 2023-04-02

[21] Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. PNAS 105(4), 1118–1123 (2008)

[22] Schaeffer, S.E.: Graph clustering. Comput. Sci. Rev. 1(1), 27–64 (2007)

[23] Song, K., Tan, X., Qin, T., Lu, J., Liu, T.Y.: Mpnet: Masked and permuted pre-training for language understanding. NIPS 33, 16857–16867 (2020)

[24] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: Extraction and mining of academic social networks. In: KDD. pp. 990–998 (2008)

[25] Tatti, N.: Strongly polynomial efficient approximation scheme for segmentation. IPL 142, 1–8 (2019)

[26] Von Luxburg, U.: A tutorial on spectral clustering. Stat. Comput. 17, 395–416 (2007)

[27] Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained k-means clustering with background knowledge. In: ICML. vol. 1, pp. 577–584 (2001)