Modularity-based Sparse Soft Graph Clustering

Alexandre Hollocou[1], Thomas Bonald[3], and Marc Lelarge[1,2]

[1]INRIA, Paris France
[2]ENS, Paris France
[3]Telecom Paristech, Paris France

2018

Abstract

Clustering is a central problem in machine learning for which graph-based approaches have proven their efficiency. In this paper, we study a relaxation of the modularity maximization problem, well-known in the graph partitioning literature. A solution of this relaxation gives to each element of the dataset a probability to belong to a given cluster, whereas a solution of the standard modularity problem is a partition. We introduce an efficient optimization algorithm to solve this relaxation, that is both memory efficient and local. Furthermore, we prove that our method includes, as a special case, the Louvain optimization scheme, a state-of-the-art technique to solve the traditional modularity problem. Experiments on both synthetic and real-world data illustrate that our approach provides meaningful information on various types of data.

1 Introduction

Unsupervised learning is getting more and more popular as the cost of acquiring labels for ever-increasing amounts of data remains a huge limit for training supervised models in many practical applications. Unsupervised techniques are very diverse and range from matrix factorization [21, 18] to representation learning [25, 16], but *clustering* is certainly the most emblematic and the most widespread unsupervised task. It consists in grouping objects into clusters of similar elements. Many approaches such as k-means [24] or hierarchical methods [32] have been proposed to address this problem, but in this paper, we specifically focus on a popular class of algorithms based on graphs. The graphs considered in this type of methods can either be implicitly defined by the structure of the input data (e.g. for social network data, it is natural to consider a graph where nodes correspond to users, and where edges correspond to friendships between these users), or be generated from any type of data using similarity-based approaches, including k-nearest neighbor graphs or kernel-based methods.

In the present work, we take a particular interest in a quality function defined on node partitions, called the *modularity*, that has been widely used to tackle the problem of node clustering in graphs [28, 9]. Numerous algorithms have been proposed to solve the problem of maximizing this objective function [26, 6, 14]. The Louvain algorithm [2], which is built in a number of graph analysis software packages, is arguably the most popular approach to solve this problem. A solution to the modularity optimization problem is a partition of the graph nodes. In particular, a node cannot belong to more than one cluster. Yet, in many real-life applications, it makes more sense to allow some elements to belong to multiple clusters. For instance, if we consider the problem of clustering the users of a social network such as Facebook or LinkedIn, we see that a person can belong to several social circles, such as her family, her colleagues and her friends, and we might be interested in recovering all these circles with a clustering algorithm, even if they do not form a partition.

In order to overcome this limit of the classical definition of modularity, we introduce a relaxation of the problem to perform a so-called *soft clustering*, also referred to as *fuzzy clustering* [8, 1]. A solution to this new problem gives for each node a degree of membership to each cluster, instead of returning a simple node partition. More precisely, we obtain a probability $p_{ik} \in [0, 1]$ for each node i to belong to a given cluster k, where the classical modularity-based techniques implicitly consider that this probability can be either 0 or 1.

The main contribution of this paper is to introduce an efficient algorithm to find an approximation of this *soft* version of the modularity maximization problem. This algorithm has four main advantages:

1. the number of clusters does not need to be specified;
2. the algorithm is local;
3. the solution found by the algorithm is sparse;
4. the algorithm includes the Louvain algorithm as a special case.

More precisely, property 2 indicates that each update of the membership information of a given node depends only on its direct neighbors in the graph, which implies a fast computation of these updates. Property 3 states that most of the membership probabilities p_{ik} returned by our algorithm are equal to 0, which guarantees an efficient storage of the solution. Finally, property 4 translates into the fact that an update performed by our algorithm reduces to an update performed by the Louvain algorithm as soon as the unique parameter of our algorithm is large enough.

The remainder of the paper is organized as follows. In section 3, we present the related work on the topic of modularity optimization. We introduce the relaxation of the modularity maximization problem in section 4. In section 5, we present our optimization method and obtain theoretical guarantees about its convergence. In section 6, we take benefits of the specificities of our optimization technique to propose an algorithm that is both local and memory efficient. In section 7, we study the ties of our approach to the Louvain algorithm. Finally, in section 8, we present experimental results on synthetic and real-world data, and section 9 concludes the paper. Complete proofs of the main results of the paper are presented in the appendices.

2 Notations

We are given a weighted and undirected graph $G = (V, E, \boldsymbol{W})$. We use V to denote the set of nodes, E the set of edges, and $\boldsymbol{W} = (W_{ij})_{i,j \in V}$ the adjacency matrix of the graph. If $(i, j) \in E$, $W_{ij} > 0$ is the weight of edge (i, j), and if $(i, j) \notin E$, $W_{ij} = 0$. We use w_i to denote the weighted degree of node i, $w_i = \sum_{j \in V} W_{ij}$, and w the total weight of the graph $w = \sum_{i \in V} w_i$. We use $\mathrm{Nei}(i)$ to denote the set of the neighbors of node $i \in V$, and $j \sim i$ as a notation for $j \in \mathrm{Nei}(i)$. Finally, we use $n = |V|$ to denote the number of nodes, and $m = |E|$ the number of edges of the graph.

3 Related work

3.1 Modularity optimization

The modularity function (1) has first been introduced by Newman and Girvan [28] to measure the quality of a partition of graph nodes. For a given node partition P, modularity is defined as

$$Q(P) = \frac{1}{w} \sum_{C \in P} \sum_{i,j \in C} \left(W_{ij} - \frac{w_i w_j}{w} \right). \tag{1}$$

Modularity can be interpreted as the difference between the probability to pick an edge in G between two nodes of the same cluster C, i.e. an *intra-cluster* edge, and the probability to pick such an intra-cluster edge in a random graph generated with the so-called *null model*, where the probability that an edge (i, j) exists is $w_i w_j / w$.

Clustering approaches based on modularity try to solve the modularity maximization problem

$$\max_{P \in \mathcal{P}} Q(P), \tag{2}$$

where $\mathcal{P}$ is the set of the partitions of V.

We refer to the Newman and Girvan modularity as the *hard modularity*, to distinguish it from the *soft modularity* that we introduce in Section 4. The hard modularity optimization problem (2) cannot be solved exactly in real applications as it has been proven to be NP-hard [3]. However, many methods have been proposed to find good approximations of the modularity optimum in reasonable time. These techniques includes greedy algorithms [26], spectral approaches [27], and simulated annealing methods [14]. A thorough review of existing algorithms can be found in [9].

One of the most popular algorithm for finding a good approximation of the modularity maximum is the Louvain algorithm [2], which is widely used in benchmarks for its ability to scale up to very large graphs. In this paper, we introduce an algorithm for optimizing the relaxation of the modularity maximization problem (5) that takes a parameter t, and we prove that each update performed by our algorithm reduces to the update performed by the Louvain algorithm if t is large enough. This constitutes a strong guarantee since the Louvain algorithm has proven its worth on multiple real-life applications.

3.2 Modularity relaxation

Relaxations of the hard modularity optimization problem have been tackled by few articles [29, 13, 15, 4, 19]. They all introduce membership matrices $\boldsymbol{p} \in \mathbb{R}^{n\times K}$, with $K \geq 1$, where $p_{ik} \geq 0$ represents the degree of membership of node $i \in V$ to the k^{th} cluster, $k \in [\![1, K]\!]$. K is a given positive integer, that represents the maximum number of clusters. These relaxations all have similar forms and are very close to the one we introduce in Section 4.

These works differ in the optimization techniques they use to solve the problem. In [4], Chang et al. use a softmax parameterization for $\boldsymbol{p}$, defining for each $i \in V$ and k

$$p_{ik} = \frac{\exp(\theta_{ik})}{\sum_l \exp(\theta_{il})} \tag{3}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{n\times K}$ is the parameter to optimize. In [29], Nicosia et al. use a genetic algorithm to optimize a more general relaxation of the modularity problem. In [13], Griechisch et al. do not directly study the optimization of the relaxation problem, but they rely on an external quadratic solver. Finally, in [15], Havens et al. use a spectral approach that does not directly solve the relaxation of the modularity problem, but where modularity is used as a selection criterion.

The main limitation of these methods lies in the maximum number of clusters K that must be specified. We could get around this issue by taking large values for K, but all the approaches cited above do not scale well to large K. Indeed, the solutions $\boldsymbol{p} \in \mathbb{R}^{n\times K}$ found by these methods are dense matrices. In other words, the number of parameters to store in memory and to optimize is in $O(nK)$, which quickly becomes prohibitive for large values of K. For instance, we can see from (3) that, in the approach of [4], the matrix $\boldsymbol{p}$ is the densest possible matrix, i.e. all its coefficients are positive. In the approach of [29], the genetic algorithm starts with dense random matrices of $\mathbb{R}^{n\times K}$, and its hybridation and mutation mechanisms do not lead to sparser matrices, so that all the coefficients of the solution $\boldsymbol{p}$ found by this algorithm are positive with probability 1.

In this paper, we introduce an efficient algorithm to solve a relaxation of the modularity optimization problem with a membership matrix $\boldsymbol{p} \in \mathbb{R}^{n\times n}$. Thus, the maximum number of clusters does not need to be specified. Besides, unlike the approaches presented above, the updates performed by our algorithm preserve the sparsity of the solution $\boldsymbol{p}$ and are *local*, in the sense that the membership vector $\boldsymbol{p}_{i\cdot}$ of a node $i \in V$ is updated using only membership information from its neighbors in the graph. Thus, our algorithm can easily scale up to large datasets with large number of clusters, which is not the case of the algorithms listed above.

As proposed in [33], the problem of soft graph clustering can also be tackled by a combination of spectral embedding [31, 10] and soft-clustering in the vector space, with algorithms such as the fuzzy C-means algorithm [8] or the NEO k-means algorithm [33]. However, the number of clusters K has to be specified for these techniques. Besides, spectral embedding methods do not scale well to large size graphs [30]. For instance, they are unable to handle the subgraph of Wikipedia that we use in our experiments in Section 8. Finally, it is worth mentioning that several algorithms have been proposed for the related problem of overlapping community detection in networks [20, 34]. Whereas these approaches return clusters $C \subset V$, soft-clustering methods give for each node the degrees of membership to all clusters.

4 Soft modularity

The problem of modularity maximization (2) can be reformulated as an optimization problem with constraints on matrices by introducing the membership matrix associated with a partition P.

Definition 4.1. *Given an ordered partition $P = (C_1, \dots, C_K)$ of the nodes of V (i.e. a partition of V whose sets are given in a certain order), we define the membership matrix $\boldsymbol{p}$ associated with P as follows: $\forall i \in V, \forall k \in [\![1, K]\!]$, $p_{ik} = 1$ if $i \in C_k$, and 0 otherwise.*

It is easy to see that P is an optimal solution of the modularity maximization problem (2) if and only if its associated membership matrix $\boldsymbol{p}$ is an optimal solution of

$$\begin{aligned} \underset{\boldsymbol{p} \in \mathbb{Z}^{n \times n}}{\text{maximize}} \quad & \frac{1}{w} \sum_{i,j \in V} \sum_{k=0}^{n} \left(W_{ij} - \frac{w_i w_j}{w} \right) p_{ik} p_{jk} \\ \text{subject to} \quad & \forall i \in V, \sum_{k=1}^{n} p_{ik} = 1 \\ & \forall i \in V, \forall k \in [\![1, n]\!], p_{ik} \geq 0. \end{aligned} \tag{4}$$

Given a membership matrix $\boldsymbol{p}$, we use $\boldsymbol{p}_{i\cdot}$ to denote the vector that corresponds to the i^{th} line of $\boldsymbol{p}$ and $\boldsymbol{p}_{\cdot k}$ to denote the vector that corresponds to its k^{th} column. From the new formulation (4), we introduce the natural relaxation of the problem where coefficients of $\boldsymbol{p}$ can take real values

$$\begin{aligned} \underset{\boldsymbol{p} \in \mathbb{R}^{n \times n}}{\text{maximize}} \quad & \frac{1}{w} \sum_{i,j \in V} \left(W_{ij} - \frac{w_i w_j}{w} \right) \boldsymbol{p}_{i\cdot}{}^T \boldsymbol{p}_{j\cdot} \\ \text{subject to} \quad & \forall i \in V, \boldsymbol{1}^T \boldsymbol{p}_{i\cdot} = 1 \\ & \forall i \in V, \boldsymbol{p}_{i\cdot} \geq 0. \end{aligned} \tag{5}$$

Note that if $\boldsymbol{p}$ is a feasible solution of this problem, then p_{ik} can be interpreted as a probability. It corresponds to the probability for a node i to belong to cluster k. So, by solving this relaxation of the modularity optimization problem we can obtain nodes that belong to multiple clusters unlike in the classical modularity maximization problem. We refer to a solution to this problem as a *soft clustering* of the nodes of G. We use $Q(\boldsymbol{p})$ to refer to this new objective function, that we call *soft modularity*:

$$Q(\boldsymbol{p}) = \frac{1}{w} \sum_{i,j \in V} \left(W_{ij} - \frac{w_i w_j}{w} \right) \boldsymbol{p}_{i\cdot}{}^T \boldsymbol{p}_{j\cdot}. \tag{6}$$

Remark that, when $\boldsymbol{p}_{i\cdot} \geq 0$, the constraint $\forall i \in V, \boldsymbol{1}^T \boldsymbol{p}_{i\cdot} = 1$ is equivalent to the l_1 constraint $\|\boldsymbol{p}_{i\cdot}\|_1 = 1$, where $\|\cdot\|_1$ denotes the l_1 norm. Therefore, we expect a solution to problem (5) to be sparse [11].

5 Alternating projected gradient descent

We can rewrite the relaxation of the modularity optimization problem as a classic minimization problem $\min_{\boldsymbol{p} \in \mathcal{X}} J(\boldsymbol{p})$, where $J : \boldsymbol{p} \mapsto -Q(\boldsymbol{p})$ is the cost function and $\mathcal{X} = \{\boldsymbol{p} \in \mathbb{R}^{n \times n} : \forall i, \boldsymbol{p}_{i\cdot} \geq 0, \boldsymbol{1}^T \boldsymbol{p}_{i\cdot} = 1\}$ is the set of constraints.

We define the normalized weighted Laplacian of the graph, as $\boldsymbol{\mathcal{L}} = \boldsymbol{I} - \boldsymbol{D}^{-1/2} \boldsymbol{W} \boldsymbol{D}^{-1/2}$, where $\boldsymbol{D}$ is the diagonal matrix whose entries are the weighted degrees of the nodes: $\boldsymbol{D} = \text{diag}(w_1, \dots, w_n)$. The Laplacian matrix is symmetric and semi-definite positive [5]. We use $0 \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ to denote its eigenvalues.

Theorem 5.1. *The function J is convex if and only if the second lowest eigenvalue λ_2 of the normalized Laplacian $\boldsymbol{\mathcal{L}}$ verifies $\lambda_2 \geq 1$.*

Proof. We prove in Appendix A material that the eigenvalues of the hessian matrix $\boldsymbol{H}$ of J are $0, \lambda_2 - 1, \dots, \lambda_n - 1$. □

The condition $\lambda_2 \geq 1$, which corresponds to a large spectral gap, is in general not satisfied. For instance, Lemma 1.7. in [5] proves that $\lambda_2 \leq 1$ if G is unweighted and not complete. Therefore, the loss function associated with our problem is in general non-convex. However, it is convex in $\boldsymbol{p}_{i\cdot}$ for a graph with no self-loop as shown in the following proposition. In the rest of the paper, we assume that the graph G does not contain any self-loop.

Proposition 5.2. *If the graph does not contain any self loops, i.e. if $W_{ii} = 0$ for all node i, the function $\boldsymbol{p} \mapsto J(\boldsymbol{p})$ is convex with respect to variable $\boldsymbol{p}_{i\cdot}$ for all $i \in V$.*

Proof. If $W_{ii} = 0$, the hessian matrix $\boldsymbol{H}_i$ of J with respect to $\boldsymbol{p}_{i\cdot}$ is $\boldsymbol{H}_i = 2(w_i/w)^2\boldsymbol{I}$. □

With this result, we can apply the methods of convex optimization with convex constraints to the problem $\min_{\boldsymbol{p}_{i\cdot}\in\mathcal{Y}} J(\boldsymbol{p})$, with fixed $\boldsymbol{p}_{j\cdot}$ for $j \neq i$, where the set $\mathcal{Y}$ corresponds to the probability simplex of $\mathbb{R}^n$: $\mathcal{Y} = \{\boldsymbol{q} \in \mathbb{R}^n : \boldsymbol{q} \geq 0, \boldsymbol{1}^T\boldsymbol{q} = 1\}$.

We use $\pi_{\mathcal{Y}}$ to denote the euclidean projection onto the probability simplex $\mathcal{Y}$, $\pi_{\mathcal{Y}}(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\mathcal{Y}} \|\boldsymbol{x} - \boldsymbol{y}\|^2$. Then, using the projected gradient descent method [12, 22], we can define an update rule for variable $\boldsymbol{p}_{i\cdot}$ such that modularity is non-decreasing at each step. The gradient of J with respect to $\boldsymbol{p}_{i\cdot}$ is: $\nabla_i J = -\frac{2}{w}\sum_{j\in V}(W_{ij} - w_i w_j/w)\boldsymbol{p}_{j\cdot}$.

Theorem 5.3. *The soft modularity objective function $Q(\boldsymbol{p})$ is non-decreasing under the update rule*

$$\boldsymbol{p}_{i\cdot} \leftarrow \pi_{\mathcal{Y}}\left(\boldsymbol{p}_{i\cdot} + \frac{2t}{w}\sum_{j\in V}\left(W_{ij} - \frac{w_i w_j}{w}\right)\boldsymbol{p}_{j\cdot}\right) \tag{7}$$

for all node $i \in V$ if the step size t verifies $t < (w/w_i)^2$.
$Q(\boldsymbol{p})$ is invariant under all these update rules iff $\boldsymbol{p}_{i\cdot}$ minimizes $Q(\boldsymbol{p})$ with fixed $\boldsymbol{p}_{j\cdot}$, $j \neq i$, for all node $i \in V$.

Proof. In Appendix C, we prove that $Q(\boldsymbol{p}^+) \geq Q(\boldsymbol{p}) + t\left(1 - t\left(\frac{w_i}{w}\right)^2\right)\|\boldsymbol{G}_i(\boldsymbol{p})\|^2$ where $\boldsymbol{p}^+$ is the matrix $\boldsymbol{p}$ after the update, and $\boldsymbol{G}_i(\boldsymbol{p}) = (\boldsymbol{p}_{i\cdot} - \boldsymbol{p}^+_{i\cdot})/t$. □

We now consider the natural algorithm that cycles through the nodes of the graph to apply the update (7).

Theorem 5.4. *The algorithm converges to a local maximum of the soft modularity function $\boldsymbol{p} \mapsto Q(\boldsymbol{p})$ which is a fixed point of the updates of (7).*

Proof. This is a direct consequence of the analysis conducted in the proof of Theorem 5.3 provided in Appendix C. □

6 Soft clustering algorithm

In the previous section, we have presented updates rules that can be applied in an alternating fashion to find a local maximum of the soft modularity $Q(\boldsymbol{p})$. However, there are three major issues with a naive implementation of this method.

1. The gradient descent update for each node i is computionally expensive because the update rule (7) involves a sum over all nodes of V.
2. The size of a solution $\boldsymbol{p}$ is n^2, which is prohibitive for large datasets.

3. The computational cost of the projection $\pi_{\mathcal{Y}}$ onto the probability simplex is classically in $O(n \log n)$ [7], which can be a limiting factor in practice.

In the present section, we present an efficient implementation of the algorithm that solves these three problems. In particular, our implementation guarantees that the update step for node i only requires local computation, is memory efficient, and uses a fast mechanism for projection.

6.1 Local gradient descent step

The update of Theorem 5.3 relative to node $i \in V$ can be decomposed into two steps:

1. $\hat{\boldsymbol{p}}_{i\cdot} \leftarrow \boldsymbol{p}_{i\cdot} + \frac{2t}{w} \sum_{j \in V} \left(W_{ij} - \frac{w_i w_j}{w}\right) \boldsymbol{p}_{j\cdot}$
2. $\boldsymbol{p}_{i\cdot} \leftarrow \pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot})$.

At first sight, step 1 seems to depend on information from all nodes of V, and to require the computation of a sum over n terms. However, the gradient descent step can be written as

$$\hat{\boldsymbol{p}}_{i\cdot} \leftarrow \boldsymbol{p}_{i\cdot} + \frac{2t}{w} \sum_{j \sim i} W_{ij} (\boldsymbol{p}_{j\cdot} - \bar{\boldsymbol{p}}),$$

where $\bar{\boldsymbol{p}}$ is weighted average of vector $\boldsymbol{p}_{j\cdot}$: $\bar{\boldsymbol{p}} = \sum_{j \in V} \frac{w_j}{w} \boldsymbol{p}_{j\cdot}$.

The vector $\bar{\boldsymbol{p}} \in \mathbb{R}^n$ can be stored and updated throughout the algorithm. Then, the computation of $\hat{\boldsymbol{p}}_{i\cdot}$ is purely local, in the sense that it only requires information from neighbors of node i. Moreover, the sum to compute contains only d_i terms, where $d_i = |\mathrm{Nei}(i)|$ is the unweighted degree of node i. In most real-life graphs, d_i is much smaller than the number of nodes n.

6.2 Cluster membership representation

The cluster membership variable $\boldsymbol{p}$ is a $n \times n$ matrix. In a naive implementation, the memory cost of storing $\boldsymbol{p}$ is therefore in $O(n^2)$. However, we will see below that the projection step can be written $\pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot}) = (\hat{\boldsymbol{p}}_{i\cdot} - \theta \mathbf{1})_+$ with $\theta \geq 0$, where $[(\boldsymbol{x})_+]_k = \max(x_k, 0)$. Thus, the projection acts as a thresholding step. We expect it to have the same effect as a Lasso regularization[1] and to lead to a sparse vector $\boldsymbol{p}_{i\cdot}$.

To take benefit of this sparsity, we only store the non-zero coefficients of $\boldsymbol{p}$. If, for all node $i \in V$, the number of non-zero values in vector $\boldsymbol{p}_{i\cdot}$ is lower than a given L, then the memory cost of storing $\boldsymbol{p}$ is in $O(nL)$. We will see, in our experiments in Section 8, that the average number of positive components that we observe for vectors $\boldsymbol{p}_{i\cdot}$ throughout the execution of our algorithm is lower than 2, and that the maximum number of positive components that we record is $L = 65$ for a graph with $731,293$ nodes.

6.3 Projection onto the probability simplex

The classic algorithm to perform the projection $\pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot})$ onto $\mathcal{Y}$ is described in [7]. It starts by sorting the vector $\hat{\boldsymbol{p}}_{i\cdot}$ into $\boldsymbol{\mu}$: $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$. Then, it finds $\rho = \max\left\{j \in [n],\ \mu_j - \frac{1}{j}\left(\sum_{r=1}^{j} \mu_r - 1\right) > 0\right\}$, and defines $\theta = \frac{1}{\rho}\left(\sum_{i=1}^{\rho} \mu_i - 1\right)$ so that $\pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot}) = (\hat{\boldsymbol{p}}_{i\cdot} - \theta \mathbf{1})_+$. This algorithm runs in $O(n \log n)$ because of the sorting step.

In our case, the complexity can be reduced to $O(L_i \log L_i)$, with $L_i = |\mathrm{supp}_i(\boldsymbol{p})|$, where $\mathrm{supp}_i(\boldsymbol{p}) = \{k \in [\![1, n]\!], \exists j \in \mathrm{Nei}(i) \cup \{i\}, p_{jk} > 0\}$ is the union of the supports of vectors $\boldsymbol{p}_{j\cdot}$ for $j \in \mathrm{Nei}(i) \cup \{i\}$. L_i is upper-bounded by $L(d_i + 1)$ which is typically much smaller than the total number of nodes n.

Proposition 6.1. *In order to compute $\pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot})$, we only need to sort the components $k \in \mathrm{supp}_i(\boldsymbol{p})$ of $\hat{\boldsymbol{p}}_{i\cdot}$ to determine ρ and θ. All components $k \notin \mathrm{supp}_i(\boldsymbol{p})$ of $\pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot})$ are set to zero.*

Proof. First note that

$$\sum_{k=1}^{n} \hat{p}_{ik} = \sum_k p_{ik} + \frac{2t}{w} \sum_j W_{ij} \left(\sum_k p_{ik} - \sum_k \bar{p}_k \right) = 1$$

[1] Note that the l_1 proximal operator for one component is $x \mapsto \mathrm{sign}(x)(|x| - \lambda)_+$.

since $\sum_k \sum_j \frac{w_j}{w} p_{jk} = \sum_j \frac{w_j}{w} \sum_k p_{jk} = 1$.

Let j_0 be defined by $j_0 = \max\{j : \mu_j \geq 0\}$. The function $j \mapsto \left(\sum_{r=1}^{j} \mu_r - 1\right)$ increases for $j \leq j_0$ and then decreases to 0 when $j = n$. In particular, this function is non-negative for $j \geq j_0$. This implies that $\rho \leq j_0$ and $\theta \geq 0$.

Now, for $k \notin \text{supp}_i(\boldsymbol{p})$, we have $\hat{p}_{ik} = -\frac{2tw_i}{w}\bar{p}_k \leq 0$ so that the k-th component of $\pi_{\mathcal{Y}}(\hat{\boldsymbol{p}}_{i\cdot})$ will be zero since $\theta \geq 0$. Moreover, since $\rho \leq j_0$, the value of $\hat{p}_{ik}$ is not used for the determination of ρ and θ. □

6.4 MODSOFT

Finally, the algorithm can be described as follows.

- **Initialization:** $\boldsymbol{p} \leftarrow \boldsymbol{I}$ and $\bar{\boldsymbol{p}} \leftarrow \boldsymbol{w}/w$.
- **One epoch:** For each node $i \in V$,
  - $\forall k \in \text{supp}_i(\boldsymbol{p})$, $\hat{p}_{ik} \leftarrow p_{ik} + t' \sum_{j\sim i} W_{ij}(p_{jk} - \bar{p}_k)$
  - $\boldsymbol{p}_{i\cdot}^{+} \leftarrow \text{project}(\hat{\boldsymbol{p}}_{i\cdot})$
  - $\bar{\boldsymbol{p}} \leftarrow (w_i/w)(\boldsymbol{p}_{i\cdot}^{+} - \boldsymbol{p}_{i\cdot})$ and $\boldsymbol{p}_{i\cdot} \leftarrow \boldsymbol{p}_{i\cdot}^{+}$.

where $\boldsymbol{w}$ is the vector $(w_i)_{1\leq i\leq n}$, project is the adaptation of the algorithm of [7] presented above, and t' is the effective learning rate $t' = 2t/w$. One epoch of the algorithm is typically repeated until the increase of modularity falls below a given threshold $\epsilon > 0$, as in the Louvain algorithm. Note that we chose, in this implementation, to put each node in its own cluster at initialization, but our algorithm could also be initialized with the results of another algorithm (e.g. Louvain). We refer to this algorithm as MODSOFT (MODularity SOFT clustering). We give the algorithm pseudo-code as working python code in Appendix G.

7 Link with the Louvain algorithm

In this section, we show that if the learning rate t is larger than a graph-specific threshold, one update performed by our algorithm reduces to a local update performed by the Louvain algorithm, which is commonly used to find a local solution for the *hard* modularity maximization problem (2).

First, we observe that, if the membership matrix $\boldsymbol{p}$ verifies $\boldsymbol{p} \in \{0,1\}^{n^2}$, then the update rule (7) for a node $i \in V$ becomes

$$\boldsymbol{p}_{i\cdot} \leftarrow \pi_{\mathcal{Y}}\left(\left[\mathbf{1}_{\{i\in C_k\}} + \frac{2t}{w}\left(w_i(C_k) - w_i\frac{\text{Vol}(C_k)}{w}\right)\right]_k\right)$$

where $C_k = \{i \in V : p_{ik} = 1\}$ refers to the k^{th} cluster defined by $\boldsymbol{p}$, $w_i(C)$ denotes the degree of node i in cluster $C \subset V$, $w_i(C) = \sum_{j\in C} W_{ij}$, and $\text{Vol}(C)$ denotes the volume of cluster C, $\text{Vol}(C) = \sum_{j\in C} w_j$.

In the following, we assume that: $\forall C, C' \subset V, \forall i \in V$,

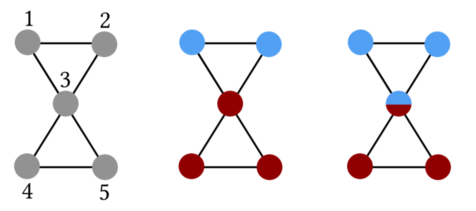
$$C \neq C' \Rightarrow w_i(C) - \tfrac{w_i \text{Vol}(C)}{w} \neq w_i(C') - \tfrac{w_i \text{Vol}(C')}{w}. \tag{H}$$

The hypothesis (H) is non-binding in real cases, because, if the weights $(W_{ij})_{(i,j)\in E}$ are continuously distributed, then we will have (H) with probability 1; and if they follow a discrete distribution, we can always add a small noise to the weights, so that (H) is verified with probability 1.

Proposition 7.1. *Let $\boldsymbol{p} \in \mathcal{X}$ be a membership matrix. If $\boldsymbol{p} \in \{0,1\}^{n^2}$, if the hypothesis (H) is verified, and if $t > w/\delta$ where*

$$\delta = \min_{\substack{C,C'\subset V\\ i\in V\\ C\neq C'}} \left|\left(w_i(C) - \frac{w_i\text{Vol}(C)}{w}\right) - \left(w_i(C') - \frac{w_i\text{Vol}(C')}{w}\right)\right|,$$

then the update rule (7) for node $i \in V$ reduces to $p_{ik} \leftarrow 1$ if $k = \underset{l:j\in C_l, j\sim i}{\arg\max}\left[w_i(C_l) - w_i\frac{\text{Vol}(C_l)}{w}\right]$, and $p_{ik} \leftarrow 0$ otherwise, where C_k denotes the k^{th} cluster defined by $\boldsymbol{p}$.



(a) Graph (b) Louvain (c) Modsoft

Figure 1: Bow tie graph

Proof. We show in Appendix E that assuming that the updated vector $\boldsymbol{p}_{i\cdot}$ has more than one positive component leads to a contradiction with $t > w/\delta$. □

In particular, this result shows that if $\boldsymbol{p} \in \{0,1\}^{n^2}$ and t is large enough, then for each update of our algorithm, the updated membership matrix $\boldsymbol{p}^+$ verifies $\boldsymbol{p}^+ \in \{0,1\}^{n^2}$, which corresponds to the maximum sparsity that can be achieved by a feasible solution $\boldsymbol{p} \in \mathcal{X}$.

The Louvain algorithm [2] is a popular heuristic to find a local maximum for the *hard* modularity maximization problem (2). One epoch of the main routine of the Louvain algorithm considers successively each node $i \in V$, and (1) removes i from its current cluster, (2) applies an update rule, that we refer to as LouvainUpdate(i), which consists in transferring i to the cluster that leads to the largest increase in hard modularity. Proposition 7.2 gives an explicit formula to pick the cluster C_{k^*} to which node i is transferred by LouvainUpdate(i).

Proposition 7.2. *The update performed by* LouvainUpdate(i) *is equivalent to transferring node i to the cluster C_k^* such that:*

$$k^* = \underset{k: j\in C_k, j\sim i}{\arg\max} \left[w_i(C_k) - w_i \frac{\mathrm{Vol}(C_k)}{w} \right]$$

Proof. We derive this result from the definition of the hard modularity in Appendix F. □

Using Proposition 7.1, we see that the update performed by LouvainUpdate(i) and the update rule (7) are equivalent if (H) is verified and $t > w/\delta$ (with δ defined in Proposition 7.1). Therefore, under these assumptions, one epoch of our algorithm is strictly equivalent to one epoch of a slightly modified version of the Louvain algorithm in which we do not apply step (1) before applying LouvainUpdate(i). Note that step (1) should have no impact on LouvainUpdate(i) if clusters are large enough. Indeed, it only replaces in Proposition 7.2 the volume of the current cluster C_k of i, $\mathrm{Vol}(C_k)$, with $\mathrm{Vol}(C_k) - w_i$. It is also worth noting that the Louvain algorithm uses an aggregation routine that is not considered in the present analysis.

8 Experimental results

8.1 Synthetic data

Bow tie

We first consider a toy example that we call the *bow tie* graph, that is defined in Figure 1 (a). The Louvain algorithm applied to this graph returns partition $\{\{1,2,3\},\{4,5\}\}$ or partition $\{\{1,2\},\{3,4,5\}\}$ depending on the order in which the updates LouvainUpdate are performed. It can be easily verified that these partitions correspond to the maximum of hard modularity over all possible partitions. In Figure 1 (b), we represent the results of the Louvain algorithm where colors code for clusters. In Figure 1 (c), we represent the results of our algorithm where the cluster membership is coded with a pie chart for each node. We see that our algorithm assigns nodes 1 and 2 to one cluster, and nodes 4 and 5 to another cluster, whereas node 3 has a

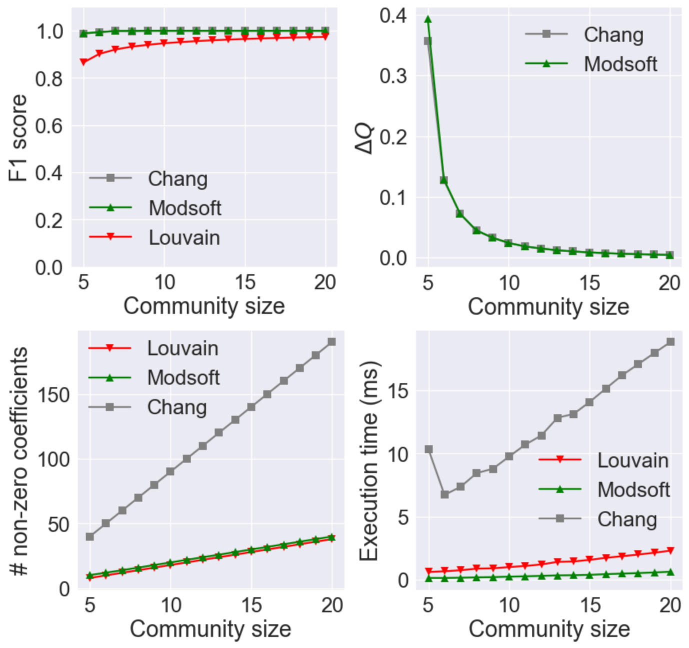


Figure 2: Overlapping SBM

mixed membership, with a probability 0.5 to belong to each cluster. It is easy to check that the membership matrix $\boldsymbol{p}$ returned by our algorithm corresponds to the optimal solution of the problem (5). The modularity Q_{soft} found by our algorithm is 50% higher than that found by the Louvain algorithm Q_{hard}.

Overlapping SBM

Now, we consider a variant with overlaps of the popular Stochastic Block Model (SBM) [17], defined as follows. We have $V = C_1 \cup \dots \cup C_K$ with $|C_1| = \ldots = |C_K| = c$ and $|C_k \cap C_{k+1}| = o$ for all k. For all nodes i and j such that $i \neq j$, the edge (i, j) exists with probability p_{in} if $i, j \in C_k$ for some k, and with probability p_{out} otherwise.

In order to numerically evaluate the performance of our algorithm on this model, we first compute the relative difference between the modularity Q_{soft} obtained with our algorithm and the modularity Q_{hard} obtained with the Louvain algorithm, $\Delta Q = (Q_{\text{soft}} - Q_{\text{hard}})/Q_{\text{hard}}$. We also compare the planted clusters of our model C_k with the clusters $\hat{C}_k$ returned by our algorithm. We consider that these clusters are the non-empty sets $\hat{C}_k$ where $\hat{C}_k = \{i \in V : p_{ik} > 0\}$. Note that the clusters $\hat{C}_k$ returned by our algorithm can overlap just as the C_k. In order to compare the clusters C_k and $\hat{C}_k$, we use the average of the classic F1 score defined as the harmonic mean of the precision and the recall, which takes values between 0 and 1, and is equal to 1 if the estimated clusters $\hat{C}_k$ correspond exactly to the initial clusters C_k.

We compute the same two metrics for Chang's algorithm [4]. Since the membership matrix returned by this algorithm is not sparse, we define $\hat{C}_k$ as $\hat{C}_k = \{i \in V : p_{ik} > \alpha\}$ for the F1 score evaluation. We present the results for $\alpha = 0.05$, but other values of α leads to similar results.

For different values of the cluster size c, we run our algorithm, Chang's algorithm and the Louvain algorithm on 100 random instances of this model, with $o = 2$, $K = 2$, $p_{\text{in}} = 0.9$, and $p_{\text{out}} = 0.1$, We display the results in Figure 2. The average F1 scores for our algorithm and Chang's algorithm are close to 1 (> 0.99 on average), and are higher than those obtained by Louvain (0.94 on average). Besides, the relative difference to the modularity found by Louvain ΔQ for Chang's algorithm and our algorithm is always positive. However, we note that ΔQ and the differences between the F1 scores become negligible as the cluster size c increases. This can be explained by the fact that, as the cluster size c increases, the proportion of nodes with

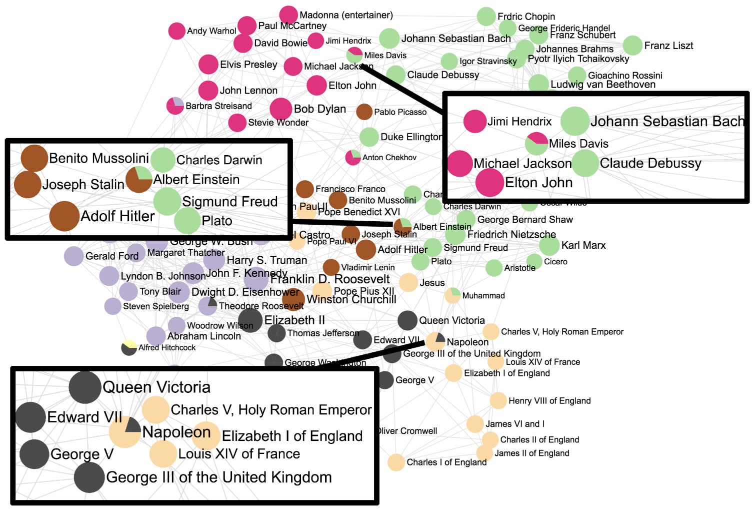


Figure 3: MODSOFT on a small subgraph of Wikipedia

mixed membership, i.e. nodes that belong to multiple clusters C_k, decreases. If this proportion is low, the misclassification of nodes with mixed membership has a low impact on the modularity score Q and on the F1 score.

We also represent in Figure 2 the sparsity of the solution in term of the number of non-zero coefficients of the matrix $\boldsymbol{p}$, and the average execution time in milliseconds for the different algorithms. We observe that our algorithm provides performance comparable to Louvain (but with richer membership information). Besides, we see that MODSOFT runs more than 10 times faster and uses 5 times less memory than Chang's algorithm.

8.2 Real data

Wikipedia

We consider a graph built from Wikipedia data, where nodes correspond to Wikipedia articles and edges correspond to hyperlinks between these articles (we consider that the edges are undirected and unweighted). Wikipedia articles can correspond to a wide variety of entities: persons, locations, organizations, concepts etc. In order to restrict ourselves to homogeneous entities, we choose to limit ourselves to Wikipedia pages that correspond to humans. For this purpose, we use Wikidata, a knowledge base that provides structured data about Wikipedia articles. In particular, Wikidata objects have an `instance_of` field that we use to determine if a Wikipedia article represents a human or not. The subgraph induced by the human entities has 731,293 nodes and 3,266,258 edges.

We run the Louvain algorithm, Chang's algorithm and MODSOFT on this subgraph. As described in Section 3, Chang's algorithm requires the maximum number of clusters K to be specified. We run it with $K = n$ (which corresponds to the implicit choice made for our algorithm), K equal to the number of clusters returned by Louvain (i.e. $K = 12,820$) and $K = 100$. Chang's algorithm is unable to handle the graph (the execution raises a memory error) for all these choices of K, whereas our algorithm runs in 38 seconds on an Intel Xeon Broadwell CPU with 32 GB of RAM. As a baseline for comparison, the Louvain algorithm runs in 20 seconds on this dataset. The difference in memory consumption between our algorithm and Chang's algorithm can easily be explained by the fact that the membership matrix $\boldsymbol{p}$ in Chang's approach is a dense $n \times K$ matrix (see Section 3), i.e. the algorithm needs to store and perform operations on $73,129,300$ coefficients for $K = 100$. By contrast, the matrix returned by our algorithm is very sparse, since it contains

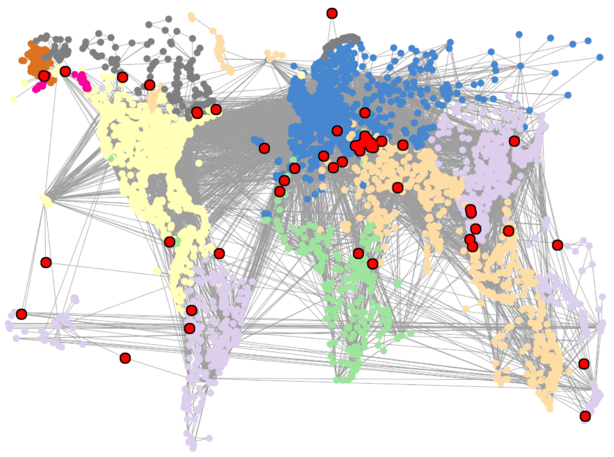


Figure 4: MODSOFT on the OpenFlights graph

only $760,546$ non-zero coefficients in our experiments, which represents a proportion of approximately 10^{-6} of the coefficients of the $n \times n$ matrix. The average number of positive components of vectors $\boldsymbol{p}_{i\cdot}$ throughout the execution of the algorithm is 1.21, and the densest vector $\boldsymbol{p}_{i\cdot}$ has 65 positive components.

Our algorithm leads to a higher modularity than the Louvain algorithm on this dataset. However, this increase represents a modularity increase ΔQ of slightly less than 1%. This can be explained by the fact that the nodes with mixed membership found by our algorithm represent less than 5% of the total number of nodes. Even if the performance increase brought by our algorithm in terms of modularity is marginal, we qualitatively observe that the results of our algorithm, and in particular the nodes with mixed membership identified by our approach, are particularly relevant.

In order to be able to graphically represent the results of our algorithm, we consider the subgraph induced by the top 100 nodes in term of degree. We display the results of our algorithm on this smaller graph in Figure 3. In particular, we observe that Napoleon (1769-1821), who became Emperor of the French in 1804, appears to belong to the same cluster as European leaders from the old regime, such as Charles V (1338-1380) and Louis XIV (1638-1715), and, at the same time, to the same cluster as post-French-revolution leaders such as Queen Victoria (1819-1901) and Edward VII (1841-1910). We also see that Miles Davis (1926-1951), a famous American jazz trumpeter, is found to belong to the same cluster as classical music composers, such as Claude Debussy (1862-1918) and Johan Sebastian Bach (1685-1750), but also to the same cluster as pop/rock musicians such as Jimi Hendrix (1942-1970) and Michael Jackson (1958-2009). Finally, we observe that Albert Einstein who became politically involved during World War II, belongs to the same cluster as thinkers, intellectuals and scientists, such as Sigmond Freud and Plato, and to the same cluster as political leaders during World War II such as Adolf Hitler and Joseph Stalin.

Open Flights

We consider a graph built from the Open Flights database that regroups open-source information on airports and airlines. The graph we consider is built as follows: the nodes correspond to the airports and the edges correspond to the airline routes, i.e. there is an edge between two airports if an airline operates services between these airports.

We apply our algorithm and the Louvain algorithm to this graph. During the execution of the algorithm, the number of positive components of vectors $\boldsymbol{p}_{i\cdot}$ remains lower than 11 and is on average 1.17. We measure a relative increase in modularity ΔQ of less that 1%, and yet we find that our algorithm brings insightful information about the dataset by identifying bridges between important zones of the world. In Figure 4, we display the results of our algorithm. We use plain colors to code for the cluster membership of *pure* nodes, i.e. nodes that belong to a unique cluster, and bigger red dots to identify nodes with a mixed membership.

Note that most of these nodes with mixed membership are located on the borders between large regions of the world. For instance, we remark that numbers of nodes are located on the frontier between Europe, Africa and Middle East, such as the Dubai airport in the United Arab Emirates. We also observe that six airports with mixed membership are located all along the border between the United States on the one hand, and Canada and Alaska on the other hand.

9 Conclusion

We studied a relaxation of the popular modularity maximization problem, with the aim of performing soft clustering of graph nodes instead of hard partitioning. In order to efficiently solve this relaxation, we introduced a novel algorithm that is based on an alternating projected gradient descent. By diving into the specific form of the gradient descent updates, we were able to guarantee the locality of each algorithm step, and to make good use of the sparsity of the solutions. As a result, unlike existing methods, our approach is both local and memory efficient. Furthermore, we proved that our algorithm includes the main routine of the popular Louvain algorithm for $t > w/\delta$. We illustrated on real-life examples, that our algorithm has an execution time and a memory consumption comparable to the Louvain algorithm, but goes further than Louvain by identifying nodes with mixed memberships that represent important bridges between clusters, even if this does not always translate into a large modularity increase.

In future works, we would like to compare our approach to a non-negative matrix factorization method applied to the matrix $\boldsymbol{W}$, which can be performed with a comparable alternating projected gradient descent [23]

A Proof of Theorem 5.1

The cost function J can be written as $J(\boldsymbol{p}) = \frac{1}{w}\sum_k \tilde{J}(\boldsymbol{p}_{\cdot k})$ where $\tilde{J}(\boldsymbol{q}) = \sum_i \sum_j (W_{ij} - w_i w_j / w) q_i q_j$. So J is convex if and only if $\tilde{J}$ is convex. Moreover, the function $\tilde{J}$ is convex iff its hessian matrix $\boldsymbol{H}$ is semi-definite positive. An expression of $\boldsymbol{H}$ is given by $\boldsymbol{H} = -2\left(\boldsymbol{W} - \frac{\boldsymbol{w}\boldsymbol{w}^T}{w}\right)$, where $\boldsymbol{w}$ denotes the vector $(w_i)_{i\in V}$.
Let $\boldsymbol{x}$ be a vector of $\mathbb{R}^n$, and $\boldsymbol{y} = \boldsymbol{D}^{1/2}\boldsymbol{x}$. Then we have:

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} &= -2\left(\boldsymbol{y}^T\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}\boldsymbol{y} - \|\boldsymbol{u}_1^T\boldsymbol{y}\|^2\right)\\ &= 2(\boldsymbol{y}^T(\boldsymbol{\mathcal{L}} - \boldsymbol{I})\boldsymbol{y} + \|\boldsymbol{u}_1^T\boldsymbol{y}\|^2)\end{aligned}$$

where $\boldsymbol{u}_1 = (1/\sqrt{w})\boldsymbol{D}^{-1/2}\boldsymbol{w}$. It is easy to verify that $\boldsymbol{u}_1$ is a normalized eigenvector associated with the first eigenvalue $\lambda_1 = 0$ of the normalized Laplacian $\boldsymbol{\mathcal{L}}$. We use $\boldsymbol{u}_1, \dots, \boldsymbol{u}_n$ to denote the orthonormal basis of eigenvectors corresponding to the eigenvalues $(\lambda_1, \dots, \lambda_n)$. Thus, we can write:

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} &= 2(\sum_{k=1}^{n}(\lambda_k - 1)\|\boldsymbol{u}_k^T\boldsymbol{y}\|^2 + \|\boldsymbol{u}_1^T\boldsymbol{y}\|^2)\\ &= 2(\sum_{k=2}^{n}(\lambda_k - 1)\|\boldsymbol{u}_k^T\boldsymbol{y}\|^2)\end{aligned}$$

We see that $\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x}$ if and only if $1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, which concludes the proof.

B Proof of Proposition 5.2

Let i be a given node. We have for all $k, k' \in [\![1, n]\!]$,

$$\frac{\partial J}{\partial p_{ik}} = -\frac{2}{w}\sum_{j\in V}(W_{ij} - w_i w_j/w)p_{jk},$$
$$\frac{\partial J}{\partial p_{ik}\partial p_{ik'}} = \frac{2}{w}(w_i^2/w - W_{ii})\delta(k,k'),$$

where $\delta(k,k') = 1$ if $k = k'$ and 0 otherwise. If $W_{ii} = 0$, the hessian matrix $\boldsymbol{H}_i$ of J with respect to $\boldsymbol{p}_{i\cdot}$ can be written $\boldsymbol{H}_i = 2(w_i/w)^2\boldsymbol{I}$. It is thus definite positive, which proves the convexity of J in $\boldsymbol{p}_{i\cdot}$.

C Proof of Theorem 5.3

Let i be a given node of V and $\boldsymbol{p} \in \mathcal{X}$ be a feasible solution of the relaxation of the soft modularity optimization problem. We define $\boldsymbol{p}^+$ with $\boldsymbol{p}^+_{i\cdot} = \pi_{\mathcal{Y}}\left[\boldsymbol{p}_{i\cdot} + \frac{2t}{w}\sum_{j\in V}\left(W_{ij} - \frac{w_i w_j}{w}\right)p_{j\cdot}\right]$, and $\boldsymbol{p}^+_{j\cdot} = \boldsymbol{p}_{j\cdot}$ for all $j \neq i$.

The objective function $J(\boldsymbol{p}) = -Q(\boldsymbol{p})$ is quadratic in p_{ik}, $k \in [\![1, n]\!]$, so we have:

$$\begin{aligned} J(\boldsymbol{p}^+) &= J(\boldsymbol{p}) + \nabla_i J(\boldsymbol{p})^T(\boldsymbol{p}_{i\cdot}{}^+ - \boldsymbol{p}_{i\cdot}) + \frac{1}{2}(\boldsymbol{p}_{i\cdot}{}^+ - \boldsymbol{p}_{i\cdot})\boldsymbol{H}_i(\boldsymbol{p}_{i\cdot}{}^+ - \boldsymbol{p}_{i\cdot}) \\ &= J(\boldsymbol{p}) - t\nabla_i J(\boldsymbol{p})^T\boldsymbol{G}_i(\boldsymbol{p}) + \left(\frac{w_i t}{w}\right)^2\|\boldsymbol{G}_i(\boldsymbol{p})\|^2 \end{aligned} \tag{8}$$

where $\nabla_i J(\boldsymbol{p})$ is the gradient of J with respect to $\boldsymbol{p}_{i\cdot}$, $\boldsymbol{H}_i$ is the hessian matrix of J with respect to $\boldsymbol{p}_{i\cdot}$ whose expression was given in the proof of Proposition 5.2, and $\boldsymbol{G}_i(\boldsymbol{p}) = (\boldsymbol{p}_{i\cdot} - \boldsymbol{p}_{i\cdot}{}^+)/t$.

Besides, by definition of $\pi_{\mathcal{Y}}$, and using the expression of $\nabla_i J(\boldsymbol{p})$ given in the proof of Proposition 5.2, we have

$$\begin{aligned} \boldsymbol{p}^+_{i\cdot} &= \arg\min_{\boldsymbol{q}\in\mathcal{Y}}\left\{\|\boldsymbol{q} - (\boldsymbol{p}_{i\cdot} - t\nabla_i J(\boldsymbol{p}))\|^2\right\} \\ &= \arg\min_{\boldsymbol{q}\in\mathcal{Y}}\left\{2t\nabla_i J(\boldsymbol{p})^T(\boldsymbol{q} - \boldsymbol{p}_{i\cdot}) + \|\boldsymbol{q} - \boldsymbol{p}_{i\cdot}\|^2\right\} \\ &= \arg\min_{\boldsymbol{q}\in\mathbb{R}^n}\left\{\nabla_i J(\boldsymbol{p})^T(\boldsymbol{q} - \boldsymbol{p}_{i\cdot}) + \frac{\|\boldsymbol{q} - \boldsymbol{p}_{i\cdot}\|^2}{2t} + h(\boldsymbol{q})\right\} \end{aligned} \tag{9}$$

where $h(\boldsymbol{q}) = 0$ if $\boldsymbol{q} \in \mathcal{Y}$, and $h(\boldsymbol{q}) = +\infty$ otherwise.

$\mathcal{Y}$ is a convex set, so h is a convex function. Note that h is non-differentiable. We use $\partial h(\boldsymbol{q})$ to denote the subdifferential of h at $\boldsymbol{q}$ i.e. the set of all its subgradients. Remember that a vector $\boldsymbol{v}$ is defined as a subgradient of h at $\boldsymbol{q}$ if it verifies, for all $\boldsymbol{q}'$, $h(\boldsymbol{q}') - h(\boldsymbol{q}) \geq \boldsymbol{v}^T(\boldsymbol{q}' - \boldsymbol{q})$.

Equation (9) can be written $\boldsymbol{p}^+_i = \arg\min_{\boldsymbol{q}} L(\boldsymbol{q})$, where L is a non-differentiable convex function. The optimality of $\boldsymbol{p}^+_{i\cdot}$ gives us $\boldsymbol{0} \in \partial L(\boldsymbol{p}^+_{i\cdot})$. Therefore, there exists a $\boldsymbol{v} \in \partial h(\boldsymbol{p}^+_{i\cdot})$, such that

$$\nabla_i J(\boldsymbol{p}) + \frac{1}{t}(\boldsymbol{p}_{i\cdot}{}^+ - \boldsymbol{p}_{i\cdot}) + \boldsymbol{v} = \boldsymbol{0}.$$

Using this result in equation (8), we obtain

$$J(\boldsymbol{p}^+) = J(\boldsymbol{p}) + t\boldsymbol{v}^T\boldsymbol{G}_i(\boldsymbol{p}) - t\left(1 - t\left(\frac{w_i}{w}\right)^2\right)\|\boldsymbol{G}_i(\boldsymbol{p})\|^2.$$

We have $t\boldsymbol{v}^T\boldsymbol{G}_i(\boldsymbol{p}) = \boldsymbol{v}^T(\boldsymbol{p}_{i\cdot} - \boldsymbol{p}_{i\cdot}{}^+) \leq h(\boldsymbol{p}_{i\cdot}) - h(\boldsymbol{p}_{i\cdot}{}^+)$ because $\boldsymbol{v} \in \partial h(\boldsymbol{p}^+_{i\cdot})$. Both $\boldsymbol{p}_{i\cdot}$ and $\boldsymbol{p}^+_{i\cdot}$ belongs to $\mathcal{Y}$, thus $\boldsymbol{v}^T\boldsymbol{G}_i(\boldsymbol{p}) \leq 0$.

Finally, we obtain

$$J(\boldsymbol{p}^{+}) \leq J(\boldsymbol{p}) - t\left(1 - t\left(\frac{w_i}{w}\right)^2\right)\|\boldsymbol{G}_i(\boldsymbol{p})\|^2.$$

We have $Q(\boldsymbol{p}^{+}) \geq Q(\boldsymbol{p})$ if $t < (w/w_i)^2$. The inequality becomes an equality if and only if $\boldsymbol{G}_i(\boldsymbol{p}) = \boldsymbol{0}$, which gives us $\nabla_i J(\boldsymbol{p}) + \boldsymbol{v} = 0$. This is equivalent to say that $\boldsymbol{p}$ is an optimum of $J + h$ with respect to $\boldsymbol{p}_{i\cdot}$.

D Proof of Theorem 5.4

The sequence of matrices $\boldsymbol{p}$ built by the algorithm converges to a limit $\boldsymbol{p}^* \in \mathcal{Y}$, since the soft modularity Q is non-decreasing under each update, $\mathcal{Y}$ is a compact, and $\boldsymbol{p} \mapsto Q(\boldsymbol{p})$ is continuous and upper bounded. From the proof of Theorem 5.3, we have for all $i \in V$, $\boldsymbol{G}_i(\boldsymbol{p}) = 0$, which gives us $(\boldsymbol{p}^*_{i\cdot})^+ = \boldsymbol{p}^*_{i\cdot}$, thus $\boldsymbol{p}^*$ is a fixed point for the update relative to node i. Moreover, we have for all node $i \in V$, $\nabla_i J(\boldsymbol{p}^*) + \boldsymbol{v} = 0$ for some $\boldsymbol{v} \in \partial h(\boldsymbol{p}^*_{i\cdot})$, which proves that $\boldsymbol{p}^*$ is a local optimum of the function $\boldsymbol{p} \mapsto J(\boldsymbol{p}) = -Q(\boldsymbol{p})$.

E Proof of Proposition 7.1

Given $\boldsymbol{p} \in \mathcal{X} \cap \{0,1\}^{n^2}$ and $i \in V$, we use $\boldsymbol{p}^+$ to denote the vector obtained by applying the update rule of Theorem 5.3 for node i. We have, for all $k \in [\![1, n]\!]$, $p^+_{ik} = \max(\hat{p}_{ik} - \lambda, 0)$ where $\hat{p}_{ik} = p_{ik} + \frac{2t}{w}[w_i(C_k) - w_i\mathrm{Vol}(C_k)/w]$ and λ is chosen so that $\sum_k p^+_{ik} = 1$.

Let $k^* = \arg\max_k[w_i(C_k) - w_i\mathrm{Vol}(C_k)/w]$. We assume that (H) is verified and that $t > w/\delta$. Thus, for all $k \in [\![1, n]\!]$ s.t. $k \neq k^*$,

$$\hat{p}_{ik^*} - \hat{p}_{ik} \geq p_{ik^*} - p_{ik} + \frac{2t}{w}\delta \geq -1 + \frac{2t}{w}\delta > 1. \tag{10}$$

In particular, this implies $p^+_{ik^*} > p^+_{ik}$. Now, note that if we had $p^+_{ik} > 0$ for a certain $k \neq k^*$, we would have $p^+_{ik^*} \in (0,1]$, $p^+_{ik} \in (0,1]$, and therefore $p^+_{ik^*} - p^+_{ik} < 1$. Yet, we would also have $p^+_{ik^*} - p^+_{ik} = (\hat{p}_{ik^*} - \lambda) - (\hat{p}_{ik} - \lambda) = \hat{p}_{ik^*} - \hat{p}_{ik}$, which leads to a contradiction with (10).

Therefore, we have $\forall k \neq k^*$, $p^+_{ik} = 0$, which immediately gives us $p^+_{ik^*} = 1$.

Finally, we see have seen in our simplification of the projection onto the probability simplex that the $\arg\max$ in the definition of k^* can be taken over the communities in the neighborhood of i.

F Proof of Proposition 7.2

The variation of *hard* modularity when node $i \in V$ joins cluster C_k can be written as

$$\Delta Q(i \to C_k) = \frac{1}{w}\left[2\left(w_i(C_k) - w_i\frac{\mathrm{Vol}(C_k)}{w}\right) - \frac{w_i^2}{w}\right]. \tag{11}$$

Therefore, $\arg\max_k \Delta Q(i \to C_k) = \arg\max_k \left(w_i(C_k) - \frac{w_i\mathrm{Vol}(C_k)}{w}\right)$.

G Algorithm pseudo-code

We give the pseudo-code of the algorithm as working python code in in Algorithm 1. The algorithm only requires one parameter, the learning rate `lr`, which corresponds to the $\frac{2t}{w}$ factor in the previous section.

The algorithm stores two variables `p` and `avg_p`. The variable `p` corresponds to the membership matrix $\boldsymbol{p}$, i.e. the output of our algorithm, and is stored as a dictionary of dictionary, so that each `p`$[i][k]$ corresponds to a positive coefficient of $\boldsymbol{p}$, p_{ik}. The variable `avg_p` is a dictionary used to store the average cluster membership vector $\bar{\boldsymbol{p}}$, so that `avg_p`$[k]$ corresponds to $\bar{p}_k$.

Algorithm 1 Soft-modularity optimization

Require: `nodes`, `edges`, `degree`, `w`, `lr` (learning rate)

Initialization

```
p = dict()
avg_p = dict()
for node in nodes:
    p[node] = {node: 1.}
    avg_p[node] = (1./w) * degree[node]
```

One epoch (update the membership matrix `p`)

```
for node in nodes:
    new_p = dict()
    # Gradient descent step
    for com in p[node]:
        new_p[com] = p[node][com]
    for neighbor in edges[node]:
        weight = edges[node][neighbor]
        for com in p[neighbor]:
            if com not in new_p:
                new_p[com] = 0.
            new_p[com] += lr * weight * p[neighbor][com]
    for com in new_p:
        new_p[com] -= lr * degree[node] * avg_p[com]
    # Projection step
    new_p = project(new_p)
    # Updating average membership vector
    for com in p[node]:
        avg_p[com] -= (1./w) * degree[node] * p[node][com]
    p[node] = dict()
    for com in new_p:
        avg_p[com] += (1./w) * degree[node] * new_p[com]
        p[node][com] = new_p[com]
```

Sub-routine `project`

```
def project(in_dict):
    # Sort the values of in_dict in decreasing order
    values = sorted(in_dict.values(), reverse=True)
    # Find the value of lambda
    cum_sum = 0.; lamb = 0.
    i = 1
    for val in values:
        cum_sum += val
        new_lamb = (1. / i) * (cum_sum - 1.)
        if val - new_lamb <= 0.:
            break
        else:
            lamb = new_lamb
            i += 1
    # Create the output dictionary
    out_dict = dict()
    for key in in_dict:
        out_dict[key] = max(in_dict[key] - lamb, 0)
    return out_dict
```

The graph is given to the algorithm as four variables: `nodes`, `edges`, `degree` and `w`. The variable `nodes` contains the list of the nodes. The variable `edges` is a dictionary of dictionary, where `edges`$[i][j]$ contains the weight W_{ij}. The variable `degree` is a dictionary containing the node degrees, and `w` is the total weight of the graph.

One epoch in our algorithm corresponds to the application of all the update rules of Theorem 5.3, each update rule corresponding to the update of the membership probabilities of one node. Several strategies can be adopted regarding the choice of the number of epochs. A fixed number of epoches can be given in advance as an additional parameter to the algorithm, or the soft-modularity can be computed at the end of each epoch, and be used as a stopping criterion. For instance, we can stop the algorithm when the modularity increase becomes smaller than a given precision factore $\epsilon > 0$. This later strategy is the equivalent of the strategy used by the Louvain algorithm for the *hard* modularity problem.

In practice, in some cases, it proves more efficient to consider the more general update rule

$$\boldsymbol{p}_{i\cdot} \leftarrow \pi_{\mathcal{Y}} \left(b\boldsymbol{p}_{i\cdot} + t' \sum_{j \sim i} W_{ij}(\boldsymbol{p}_{j\cdot} - \bar{\boldsymbol{p}}) \right)$$

where $b \geq 0$ is a bias parameter. In future work, we would like to understand the impact of a $b \neq 1$ on the solution.

References

[1] J. C. Bezdek. Objective function clustering. In *Pattern recognition with fuzzy objective function algorithms*, pages 43–93. Springer, 1981.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[3] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer, 2007.

[4] C.-T. Chang and C.-S. Chang. A unified framework for sampling, clustering and embedding data points in semi-metric spaces. *arXiv preprint arXiv:1708.00316*, 2017.

[5] F. R. Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[6] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[7] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

[8] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. 1973.

[9] S. Fortunato and C. Castellano. Community structure in graphs. In *Computational Complexity*, pages 490–512. Springer, 2012.

[10] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. *IEEE transactions on pattern analysis and machine intelligence*, 26(2):214–225, 2004.

[11] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[12] A. A. Goldstein. Convex programming in hilbert space. *Bulletin of the American Mathematical Society*, 70(5):709–710, 1964.

[13] E. Griechisch and A. Pluhár. Community detection by using the extended modularity. *Acta Cybern.*, 20(1):69–85, 2011.

[14] R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004.

[15] T. C. Havens, J. C. Bezdek, C. Leckie, K. Ramamohanarao, and M. Palaniswami. A soft modularity function for detecting fuzzy communities in social networks. *IEEE Transactions on Fuzzy Systems*, 21(6):1170–1175, 2013.

[16] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[17] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[18] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[19] Y. Kawase, T. Matsui, and A. Miyauchi. Additive approximation algorithms for modularity maximization. *arXiv preprint arXiv:1601.03316*, 2016.

[20] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.

[21] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[22] E. S. Levitin and B. T. Polyak. Constrained minimization methods. *USSR Computational mathematics and mathematical physics*, 6(5):1–50, 1966.

[23] C.-J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[24] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[26] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[27] M. E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

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

[29] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):P03024, 2009.

[30] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[31] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[32] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[33] J. J. Whang, I. S. Dhillon, and D. F. Gleich. Non-exhaustive, overlapping k-means. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 936–944. SIAM, 2015.

[34] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.