Learning with subsampling using zeros of random fields

Research Internship Report - Master MVA

Clément Bonet
Internship with **Raphaël Lachièze-Rey** and **Pierre Latouche**
MAP5 at Université de Paris
April - September 2020

ENS Paris Saclay, Telecom Paris

Remerciements

Je tiens à remercier mes maîtres de stage Raphaël Lachièze-Rey et Pierre Latouche, ainsi que Antoine Marchina, qui m'ont accompagné pendant ces 6 mois et aidé à avancer tout au long de ce travail malgré le contexte sanitaire compliqué (lié à l'épidémie de COVID19).

Je voudrais aussi remercier Eric Lemoine pour avoir accepté de relire ce travail et pour ses conseils avisés sur la rédaction de ce rapport.

Contents

1 Introduction

Subsampling – sampling a subset of a dataset – is largely used in Machine Learning, in the contexts of supervised or non supervised learning. As we are able to collect more data, we want to be able to analyze them. In order to do that, we either need algorithms which can scale with the amount of data, or we need to choose carefully chosen subsamples.

Depending on the context in which we want to apply subsampling, we do not have the same needs. For example, if we want to use a costly algorithm such as fitting Gaussian process to data, we can use costly subsampling techniques such as determinantal point processes (Kulesza and Taskar [2012], Burt et al. [2020]), and sometimes even have coreset guarantees (Tremblay et al. [2018]). We can also want to initialize an algorithm such as an online version of the classification expectation maximization (Samé et al. [2007]). Unfortunately, subsampling algorithms using determinantal point processes can be heavy and we therefore need a cheaper way of subsampling.

In graph theory, we know that, e.g. with social networks, we often want to analyze data from massive networks. Unfortunately, some algorithms do not scale well and therefore, we find ourselves with the same problem as previously stated. Subsampling methods, naive as they can be, already exist; but some can be rather computationally heavy. Moreover, it is not straightforward to define what a good subsample is, as it depends on the context and on the objective of the subsampling.

In this work, we aim at presenting an alternative subsampling algorithm using Gaussian random fields, and proposed by Mr Lachièze-Rey. We will first define it in the general case, before adapting it in different contexts, such as in Euclidean space or graphs, and for different goals. We will then try to compare its efficiency with existing methods.

More precisely, in section 2, we will present the general idea of the algorithm. Then, in section 3, we will see existing techniques of sampling based on determinantal point processes as well as the notion of coreset, before presenting the adaptation of our proposed algorithm on $\mathbb{R}^d$. Finally, in section 4, we will first present existing algorithms allowing to sample on graphs, and the adaptation of our algorithm on graphs. In the third part of each section, we will try to apply this algorithm in order to sample an initialization set for online classification expectation maximization algorithms. Moreover, in section 4.4, we will see a promising clustering algorithm on graphs derived following the study of our algorithm of subsampling on graphs.

This work was conducted in a 24 weeks internship at the laboratory MAP5 under the supervision of Raphaël Lachièze-Rey and Pierre Latouche.

2 Overall Presentation of the Algorithm

This section will present the general idea of the algorithm. However, we will see that the algorithm itself cannot be used directly. We first need to tune it according to the configuration on which we want to use it. For example, we will not use the same kernels when we want to subsample nodes on a graph, or when we want to choose a subsample of points in $\mathbb{R}^d$. Moreover, we will see that there are several parameters which can be modified as well in order to better suit a given dataset.

2.1 Principle of the Algorithm

The algorithm we studied aims at subsampling a dataset using zeros of random fields. To do so, the first principle is to sample independently random variables following for example a reduced centered Gaussian law. Then we apply on each point a kind of convolution to create a random field. The idea is that this field is "almost" continuous so that two close points for the corresponding distance used in the convolution will have a close value. Then finally, we have to define an appropriate neighborhood in order to be able to keep the local maxima. The idea is that it should keep proportionally more points in areas where there are fewer points than in areas which are very densely populated.

Algorithm 1: Algorithm of Subsampling

Result: Subsample S

Initialization: A sample of points X, a kernel k, a radius r

- $\forall x \in X,\ f_0(x) \sim \mathcal{N}(0,1)$
- $\forall x \in X,\ f(x) = \sum_{y \in X} f_0(y) k(x,y) \mathbb{1}_{\{\|y-x\| \leq r\}}$

foreach $x \in X$ **do**

  add x to S if x is a local maximum for the function f_0 on a defined neighborhood

end

2.2 Degrees of Freedom of the Algorithm

Based on the previous description, we see that we need to define many parameters as well as concepts. Moreover, they will depend on the configuration of the problem.

Firstly, we have to choose the distribution law for the initial random variables. Naturally, we can take a normal distribution. It will most likely be an arbitrary choice. Then, we have to choose how to do the convolution, and on which points. We must chose a kernel; for instance in $\mathbb{R}^d$, let us have the Gaussian kernel:

$$\forall x, y \in \mathbb{R}^{\mathrm{d}},\ k(x,y) = e^{-\|x-y\|^2/2}$$

We recall that a kernel should be positive definite, ie $\forall n \in \mathbb{N}^*,\ \forall x_1, ..., x_n \in \mathbb{R}^d,\ \forall a_1, ..., a_n \in \mathbb{R}$,

$$\sum_{i,j} a_i a_j k(x_i, x_j) \geq 0$$

But we could as well choose another kernel, or even another type of function which makes sense for what we want to do.

Let us also choose on which neighborhood we want to do the convolution. This adds a parameter r of a radius of a ball, as well as a distance. On the other hand, for graphs for example, we could choose to only sum on neighbours instead of using a ball (and additional computation) based on some distance.

Finally, choosing the neighborhood may be the trickiest thing to do. This notion is indeed ill-defined in discrete settings, and it can be hard to choose a neighborhood and a rule of subsampling while keeping a reasonable amount of points.

3 Application in Euclidean Space

This section will first introduce the different motivations for which we would want an efficient subsampling algorithm in Euclidean space such as $\mathbb{R}^d$. Then we will present existing techniques based on sampling of determinantal point processes, or k-determinant point processes, before presenting the coreset theory which allows us to have guarantees on the samples. In the second part, we will present how we can adapt the algorithm presented in section 2 for this type of setting, and then we will use it to find initialization sets for algorithms such as online classification expectation maximization (OCEM) (Samé et al. [2007]). We will therefore compare this algorithm with the traditional expectation maximization (EM) algorithms, and uniform initialization for example.

3.1 Current Methods and Motivations

3.1.1 Motivations

As we live in a time of big data, subsampling is crucial since it can help for the scalability of algorithms. In the context of machine learning, subsampling can be used for several reasons. For example, when we split a dataset into a training and a testing set, we can do it by splitting arbitrarily the data, or we can take an uniform sample as a test set. But, when taking an uniform sample, we have no guarantees that this sample is distributed as the original data, which can cause bias when we train our algorithm on these data.

We will have the same issue with algorithm such as *Online Classification Expectation Maximization* (OCEM) (Samé et al. [2007]), where we want first to apply the *Classification Expectation Maximization* (CEM) algorithm (Celeux and Govaert [1992]) on a subset, and then classify and update the parameters for each point sequentially. If the subset is wrong and does not represent well all the clusters, the initial CEM will provide bad results which will influence the final results of the OCEM.

Another interesting goal can be to learn models on subsets when a dataset is too heavy for an algorithm. Then, we want to be able to learn on a subset which represents the initial dataset accurately, which likely will not happen in most cases if we sample uniformly.

To overcome the possible bias induced by selecting an uniform sample, our goal is to sample from a set in a smart way. Indeed, we want to be able to sample from the totality of the distribution. Our algorithm will even let us proportionally sample fewer points in areas where more points are present; whereas in areas wither fewer points, (outliers) which could otherwise be ignored, we can choose a higher number of points to avoid losing valuable information.

In what follows, we will first see an interesting and elegant model called the *Determinantal Point Processes* (DPP), which allows us to model repulsion between points. It has been studied a lot in recent years in the context of machine learning (Kulesza and Taskar [2012]) and it can be used to get subsamples of datasets as we want. Then, we will also see the so-called coresets methods which allow us to have guarantees when approximating the functions of costs on subsets.

3.1.2 Determinantal Point Process

3.1.2.1 Presentation and Properties

Determinantal point processes (DPP) were originally used in random matrix theory (Ginibre [1965]) or in quantum physics (Macchi [1975]). Recently, it became rather popular in the field of machine learning (Kulesza and Taskar [2012]), as it allows us to draw subsamples accounting for the diversity of the data. So, the community found use of it for recommendation systems (Wilhelm et al. [2018]), for coresets (Tremblay et al. [2018]) or more recently for sparse inference of Gaussian processes (Burt et al. [2020]) for example. The main point of interest of DPP is that it enables to model repulsion between points. It means that two very close points will most likely not be in the same sample of a DPP. Moreover, we can find pretty simple algorithms to sample from a DPP.

In what follows, we will focus on determinantal point processes in the discrete case, as we are interested in the subsampling of finite datasets. We will suppose that the entire dataset contains n elements that we will name $\mathcal{X} = \{x_i\}_{1 \leq i \leq n}$. Then a discrete DPP will be a probability measure over $\mathcal{P}(\mathcal{X})$. It will be characterized by a kernel matrix K that we will conveniently assume to be a positive semi-definite matrix, which does not have to be necessarily the case (Gartrell et al. [2019]). This kernel will be called a marginal kernel.

Definition 3.1. Discrete Determinantal Point Process: Let Y be a random subset of $\mathcal{X}$. Then Y follows a determinantal point process P of marginal kernel K if:

$$\forall A \in \mathcal{P}(\mathcal{X}),\ P(A \subseteq Y) = \det(K_A)$$

where $K_A = [K_{ij}]_{(x_i, x_j) \in A}$.

The marginal kernel should encode similarity between points, but it is pretty restrictive. Indeed, it can be shown (Hough et al. [2006]) that there exists a determinantal point process of kernel K if and only if $0 \preceq K \preceq I$.

We can see the repulsion for 2 points for example. Let $A = \{x_i, x_j\}$, then:

$$\begin{aligned} P(\{x_i, x_j\} \in Y) &= \begin{vmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{vmatrix} \\ &= K_{ii}K_{jj} - K_{ij}K_{ji} \\ &= P(\{x_i\} \in Y)P(\{x_j\} \in Y) - K_{ij}^2 \end{aligned}$$

So, we see that if x_i and x_j are very similar, then the probability to have both points in the sample Y will be smaller as K_{ij} will be bigger.

If $0 \preceq K \prec I$, we can also define DPP through *L-ensembles* (Borodin and Rains [2005]), which are characterized by a matrix L which only needs to be real and symmetric.

Definition 3.2. L-ensemble: Let Y be a random subset of $\mathcal{X}$. Then Y follows an L-ensemble P_L of matrix L if:

$$\forall A \in \mathcal{P}(\mathcal{X}),\ P_L(Y = A) = \frac{\det(L_A)}{\det(L + I)}$$

Then, we have the following result from *Macchi*, and the proof can be found in Kulesza and Taskar [2012].

Theorem 1. An L-ensemble is a DPP, and its marginal kernel is:

$$K = L(L+I)^{-1} = I - (L+I)^{-1}$$

3.1.2.2 Sampling a DPP

The simplest algorithm to sample from a DPP was introduced in Hough et al. [2006]. This algorithm relies on the fact that a DPP can be expressed as a mixture of elementary DPP, also called determinantal projection processes.

Definition 3.3. Elementary DPP/Determinantal Projection Process: A DPP is called elementary if every eigenvalue of its marginal kernel K are in $\{0,1\}$.
Let V be a set of orthonormal vectors, and a kernel $K^V = \sum_{v \in V} vv^T$, then we will denote P^V an elementary DPP of kernel K^V.

Theorem 2. (Kulesza and Taskar [2012],Hough et al. [2006]) If Y is drawn according to an elementary DPP of kernel K^V, then $|Y| = |V|$ almost surely.

Proof.

$$\begin{aligned}\mathbb{E}[|Y|] &= \mathbb{E}[\sum_{i=1}^{n} \mathbb{1}_{\{x_i \in Y\}}] \\ &= \sum_{i=1}^{n} P^V(x_i \in Y) \\ &= \sum_{i=1}^{n} K^V_{ii} = Tr(K^V) = |V|\end{aligned}$$

since K^V is of rank $|V|$.
Moreover, let $A \in \mathcal{P}(\mathcal{X})$ such that $|A| > |V|$, then $P(A \subseteq Y) = \det(K^V_A) = 0$ since K^V is of rank $|V| < |A|$.
So,

$$\begin{cases} |Y| \leq |V| \quad \text{almost surely} \\ \mathbb{E}[|Y|] = |V| \end{cases}$$

Then we can conclude that $|Y| = |V|$ almost surely. □

Theorem 3. (Kulesza and Taskar [2012],Hough et al. [2006]) A DPP with kernel $L = \sum_{i=1}^{n} \mu_i v_i v_i^T$ is a mixture of elementary DPPs:

$$\forall A \in \mathcal{P}(\mathcal{X}),\ P_L(A) = \frac{1}{\det(L+I)} \sum_{J \subseteq \{1,\dots,n\}} P^{V_J}(A) \prod_{i \in J} \mu_i$$

where it can be shown that $\det(L+I) = \prod_{i=1}^{n}(\mu_i + 1)$.

Algorithm 2: Sampling from a DPP [Kulesza and Taskar, 2012]

Result: Subsample S

- Compute the orthonormal eigendecomposition $\{(v_i, \lambda_i)\}_{1 \leq i \leq n}$ of K, or the orthonormal eigendecomposition $\{(u_i, \mu_i)\}_{1 \leq i \leq n}$ of L ($\forall i \in \{1, \dots, n\}, \lambda_i = \frac{\mu_i}{\mu_i + 1}$)
- Denote $(e_i)_i$ the canonical basis of $\mathbb{R}^n$
- $J = \emptyset$

for i=1 to n **do**

  $U \sim Unif([0, 1])$

  if $U < \lambda_i$ **then**

    $J = J \cup \{i\}$

end

- $V = \{v_i\}_{i \in J}$
- $S = \emptyset$

while $|V| > 0$ **do**

  Select x_i from $\mathcal{X}$ from the discrete distribution:

$$P(x_i) = \frac{1}{|V|} \sum_{v \in V} (v^T e_i)^2$$

  $S = S \cup \{x_i\}$

  $V = V_\perp$, an orthonormal basis from $V \setminus \{v_i\}$, orthogonal to e_i (by Gram-Schmidt)

end

We have then the following theorem which is proved in Kulesza and Taskar [2012] and in Hough et al. [2006]:

Theorem 4. (Kulesza and Taskar [2012],Hough et al. [2006]) Let $L = \sum_{i=1}^{n} \mu_i v_i v_i^T$ be an orthonormal eigendecomposition of a positive semidefinite matrix L. Then algorithm 2 samples $Y \sim P_L$.

Indeed, the first loop selects an elementary DPP:

$$\begin{aligned} P(J) &= \prod_{i \in J} \frac{\mu_i}{\mu_i + 1} \prod_{i \notin J} \left(1 - \frac{\mu_i}{\mu_i + 1}\right) \\ &= \prod_{i \in J} \frac{\mu_i}{\mu_i + 1} \prod_{i \notin J} \frac{1}{\mu_i + 1} \\ &= \frac{\prod_{i \in J} \mu_i}{\prod_i (\mu_i + 1)} \\ &= \frac{\prod_{i \in J} \mu_i}{\det(L + I)} \end{aligned}$$

Then, the second loop samples $Y \sim P_J^V$ (Kulesza and Taskar [2012]).

Unfortunately, this algorithm has pretty high complexity. Indeed, it first requires an eigendecomposition of a kernel, which will be in $O(n^3)$. Then, the algorithm itself

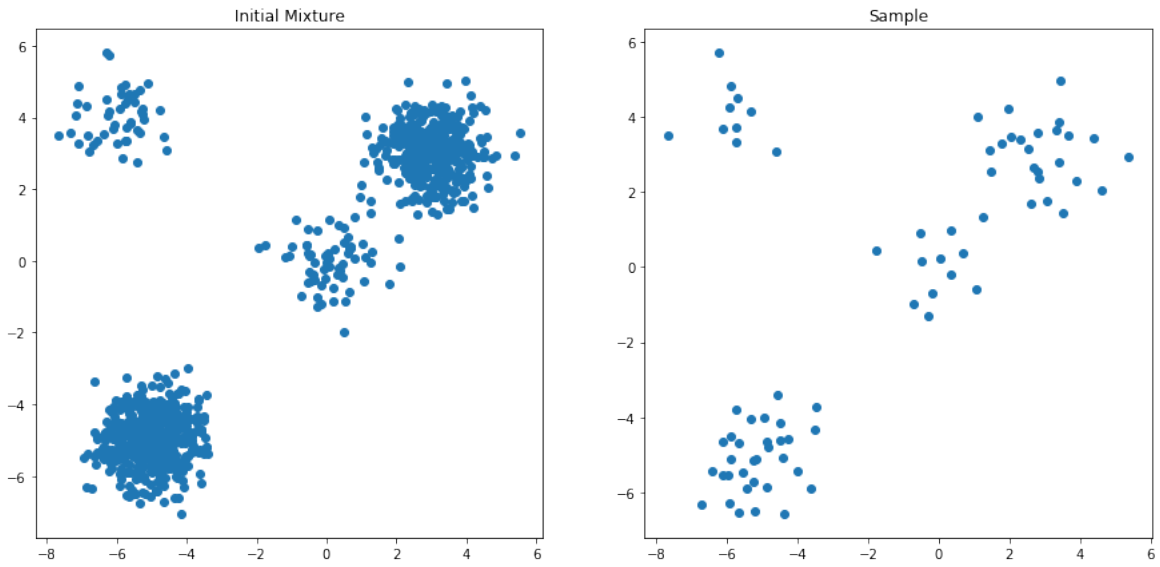


Figure 3.1: Subsampling using DPP

runs in $O(nk^3)$ with $k = |V| = |J|$, the number of eigenvectors selected in the first loop of the algorithm, because of the Gram-Schmidt orthonormalization which runs in $O(nk^2)$. According to Kulesza and Taskar, a modern multi-core machine can compute an eigendecomposition in around ten minutes with $n \approx 10000$. We can suppose that it can be faster today, but it is still prohibitive as it cannot run on huge datasets, which would be the main interest of subsampling in order to approximate cost functions for example.

Other exact sampling algorithms were proposed as in Launay et al. [2020], Poulson [2020] or Derezinski et al. [2019], but they still remain quite expensive. The last one is maybe the current fastest way to sample exactly a DPP as it requires polynomial time in k and nearly-linear time in n. It is based on a Nyström approximation of the L-kernel and on rejection sampling. In their experiments, they succeed to draw samples from a dataset with $n = 10^6$ points, which is not even possible for approximation approaches such as MCMC. However, the polynomial cost is still steep (k^6).

We can find approximate algorithms which do not allow us to sample exactly from DPP, but which are faster to compute. For example, there are MCMC based methods (Anari et al. [2016],Li et al. [2016]).

To try these different algorithms, and then to use them to compare with our own, we used the DPPy toolbox (Gautier et al. [2019]).

3.1.3 k-DPP

In some cases, we may want to be able to sample a subset with a predetermined cardinal. Unfortunately, we saw that only elementaries DPP (3.3) return these kinds of samples. And, these DPP are pretty restrictive as they require their kernels to have eigenvalues only in $\{0, 1\}$. A solution was proposed in Kulesza and Taskar [2011] with k-DPP, which are distributions over all subsets $X \subseteq \mathcal{X}$ with cardinality k. They are obtained by simply conditioning a standard DPP by the event that the cardinal is k. Let Y be a random subset of $\mathcal{X}$, then Y is a k-DPP if:

$$\forall A \in \mathcal{P}(\mathcal{X}),\ P_L^k(Y=A) = P_L(Y=A||Y|=k) = \frac{P_L(Y=A,|Y|=k)}{P_L(|Y|=k)}$$
$$= \begin{cases} 0 & \text{if } |A| \neq k \\ \frac{P_L(Y=A)}{\sum_{|B|=k} P_L(Y=B)} & \text{otherwise} \end{cases}$$
$$= \begin{cases} 0 & \text{if } |A| \neq k \\ \frac{\det(L_A)}{\det(L+I)} \frac{1}{\sum_{|B|=k} \frac{\det(L_B)}{\det(L+I)}} & \text{otherwise} \end{cases}$$
$$= \begin{cases} 0 & \text{if } |A| \neq k \\ \frac{\det(L_A)}{\sum_{|B|=k} \det(L_B)} & \text{otherwise} \end{cases}$$

If we denote $(\lambda_i)_{1 \leq i \leq N}$ the eigenvalues of L, then it can be shown (Kulesza and Taskar [2012]) that:

$$\sum_{|B|=k} \det(L_B) = e_k(\lambda_1, ..., \lambda_N) = \sum_{J \subseteq \{1,...,N\}, |J|=k} \prod_{i \in J} \lambda_i$$

So we can write,

$$\forall A \in \mathcal{P}(\mathcal{X}),\ |A| = k,\ P_L^k(Y=A) = \frac{\det(L_A)}{e_k(\lambda_1, ..., \lambda_N)} = \frac{\det(L+I)P_L(Y=A)}{e_k(\lambda_1, ..., \lambda_N)}$$
$$= \frac{1}{e_k(\lambda_1, ..., \lambda_N)} \sum_{J \subseteq \{1,...,n\}, |J|=k} P^{V_J}(A) \prod_{i \in J} \lambda_i$$

using the elementary decomposition 3 of a DPP and the fact that if $|J| \neq |A|$, then $P^{V_J}(A) = 0$.

Thus, a k-DPP is also a mixture of elementary DPP and we need therefore to sample a first subset J with probability

$$P(J) = \frac{\prod_{i \in J} \lambda_i}{e_k(\lambda_1, ..., \lambda_N)}$$

Then, the second part with the loop of the algorithm 2 remains the same. An algorithm for sampling this kind of set of eigenvectors is provided in Kulesza and Taskar [2012] (Algorithm 8), as well as an algorithm to compute the elementary symmetric polynomials e_k (Algorithm 7).

3.1.4 Coresets

Coresets are weighted subsets on which there are theoretical guarantees that models fitted on this set will provide almost the same results than if it had been fitted on the original dataset, up to some error over the loss function which can be controlled. Many algorithms were proposed to sample coresets. They often focus on one specific algorithm such as k-means (Bachem et al. [2017]), the expectation maximization (Feldman et al. [2011]) or the logistic regression for example. In Tremblay et al. [2018], they even use DPP in order to construct coresets. In what follows, we will briefly present coresets in the

random sampling setting. There are others types of algorithms which can create coreset such as geometric decomposition (Agarwal et al. [2005]) for example.

Let $\theta \in \Theta$ be a parameter and $L(\mathcal{X}, \theta) = \sum_{x \in \mathcal{X}} f(x, \theta)$ be a cost function where f is a non-negative γ-Lipschitz function with respect to θ. This setting can cover many classical problems in machine learning. For example, for k-means, we have $\Theta = \mathbb{R}^k$, θ corresponding to the vector of the k centers:

$$\forall x \in \mathcal{X},\ f(x, \theta) = \min_{c \in \theta} \|x - c\|^2$$

Definition 3.4. Coresets: (Tremblay et al. [2018]) Let $\epsilon \in]0, 1[$. The weighted subset S is a ϵ-coreset for L if:

$$\forall \theta \in \Theta,\ \left|\frac{\hat{L}}{L} - 1\right| \leq \epsilon$$

where $\hat{L}(S, \theta) = \sum_{x_s \in S} \omega(x_s) f(x_s, \theta)$.

A consequence is that if we write $\hat{\theta}^* = argmin_{\theta \in \Theta}\ \hat{L}(S, \theta)$ and $\theta^* = argmin_{\theta \in \Theta}\ L(\mathcal{X}, \theta)$, we have the following inequality:

$$(1 - \epsilon)L(\mathcal{X}, \theta^*) \leq (1 - \epsilon)L(\mathcal{X}, \hat{\theta}^*) \leq \hat{L}(S, \hat{\theta}^*) \leq L(S, \theta^*) \leq (1 + \epsilon)L(\mathcal{X}, \theta^*)$$

The difficulties of using coresets then depends on finding a sampling method with which we can guarantee this inequality. In order to do that, importance sampling techniques are being used. For example, we can use for independent and identically distributed random sampling the Horvitz-Thompson estimator:

Definition 3.5. Horvitz-Thompson Estimator: [Kolaczyk and Csárdi, 2014] Let $\pi_i = P(x_i \in S)$, denote ϵ_i the Bernoulli variable of parameter π_i, then the Horvitz-Thompson estimator is:

$$\hat{L}(S, \theta) = \sum_i \frac{f(x_i, \theta)\epsilon_i}{\pi_i}$$

As $\forall i,\ \mathbb{E}[\epsilon_i] = \pi_i$, we see that it is an unbiased estimator of L:

$$\mathbb{E}[\hat{L}(S, \theta)] = \sum_i f(x_i, \theta) = L(\mathcal{X}, \theta)$$

This estimator can be modified allowing several draws of a sample if we denote instead ϵ_i the random variable counting the number of occurrence of x_i, i.e. $\epsilon_i \sim B(|S|, p_i)$ with p_i the probability of sampling x_i at each draw. Then, the estimator of the loss is modified as:

$$\hat{L}(S, \theta) = \sum_i \frac{f(x_i, \theta)\epsilon_i}{m p_i}$$

with m the size of the sample.

Then there exist theorems giving theoretical results on how to construct a coreset using this kind of estimator as the Theorem 2.3 in Tremblay et al. [2018] for the last loss. In this article, they even show a coreset result with the Horvitz-Thompson estimator by sampling k-DPP (3.1.3).

3.2 Adaptation of the Algorithm on $\mathbb{R}^d$

3.2.1 Neighborhood with local maxima

As stated in section 2.2, there are some parameters that need to be adapted depending on the setting. In $\mathbb{R}^d$, the first important thing which needs to be chosen is the kernel which will create the Gaussian random field. In a first time, we choose the well known Gaussian kernel:

$$\forall x, y \in \mathbb{R}^d,\ K(x,y) = \exp\Big(-\frac{\|x-y\|^2}{a}\Big)$$

Then, the second thing to choose is the neighborhood. We first tried to take a ball of radius b and to compare the values of the Gaussian random field with the points present in this ball. And then, we added the point in the sample if it was the maximum on the ball centered on this point. To sum up, the algorithm is as follows:

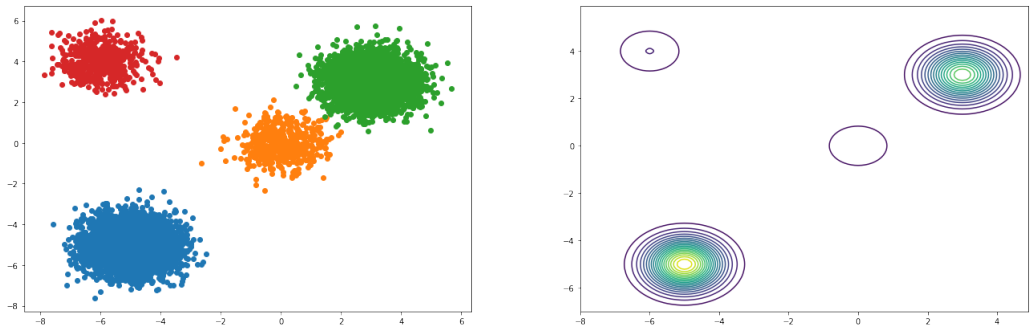
Algorithm 3: Algorithm of Subsampling in $\mathbb{R}^d$

Result: Subsample S
Initialization: A sample of points X, a kernel k, a radius r, a radius b
- $\forall x \in X,\ f_0(x) \sim \mathcal{N}(0,1)$
- $\forall x \in X,\ f(x) = \sum_{y \in X} f_0(y) k(x,y) \mathbb{1}_{\{\|y-x\| \leq r\}}$
- $S = \emptyset$
foreach $x \in X$ **do**
  if $\forall y \in B(x,b),\ f(y) < f(x)$ **then**
    $S = S \cup \{x\}$
end

We tried the algorithm on a Gaussian mixture model as it is simple to create different datasets which have dense and less dense areas. Our running example on which we will present some results is a Gaussian Mixture Model with 4 clusters and 10,000 points:

	μ	Σ	p
Cluster 1	(-5;5)	0.5 I_2	0.5
Cluster 2	(0;0)	0.5 I_2	0.05
Cluster 3	(3;3)	0.5 I_2	0.4
Cluster 4	(-6;4)	0.5 I_2	0.05

Table 3.1: Example of Gaussian Mixture Model



(a) Points GMM **(b)** Contour GMM

Figure 3.2: Gaussian Mixture Model

We choose the parameter a arbitrarily. Then, for r, we tried to take $r = 4a$ because $e^{4^2} \approx 0$. And we tuned b to have good results. Indeed, if we take b too small, then every point is going to be alone in its ball and all points will be in the subsample which is not really what is expected. And on the other hand, if b is too big, then all points will be in one neighborhood and only one point is gonna be in the subsample. So, we have to tune it in order to have the expected amount of points in the sample. We can for example try to take the mean or the median of the distance between an arbitrarily chosen point and all others. However, we can notice that if there are different areas of the data which are on different scales, then this algorithm will most likely not work as choosing a b which fits well all the areas will be impossible.

Even in a simple case with data on the same scale, it remains pretty hard to have a good tradeoff between the number of points in the sample and a good coverage of the initial dataset. Indeed, here is a table (3.2) showing the number of points in the subsample with different b. We took $a = 3$, $r = 4a$:

	Initial Dataset	$b = a$	$b = \frac{a}{10}$	$b = \frac{a}{20}$	$b = \frac{a}{40}$	$b = \frac{a}{50}$	$b = \frac{a}{100}$
Cluster 1	4,942	2	18	80	322	561	2,003
Cluster 2	534	1	21	95	279	349	480
Cluster 3	4,041	0	10	63	351	546	1,871
Cluster 4	483	1	13	96	266	328	436
Total	10,000	4	62	334	1,218	1,784	4,790

Table 3.2: Points by cluster for Gaussian Kernel

We see that as expected, at least when few points are sampled, the Gaussian with less points has nearly as much points in the subsample as the others. However, we see on the figure 3.3 that the subsample contains holes. We can understand that by the propagation of the maximum. Indeed, a point may not be its local maximum, then it will not be kept in the subsample. However, the point which had a higher value than this point could not be its own local maximum either. This phenomenon materializes itself in the figure as holes. The problem is that it occurs even when we increase the subsample size up to 20% of the data, which defeats the purpose of the subsampling when the initial dataset is very large.

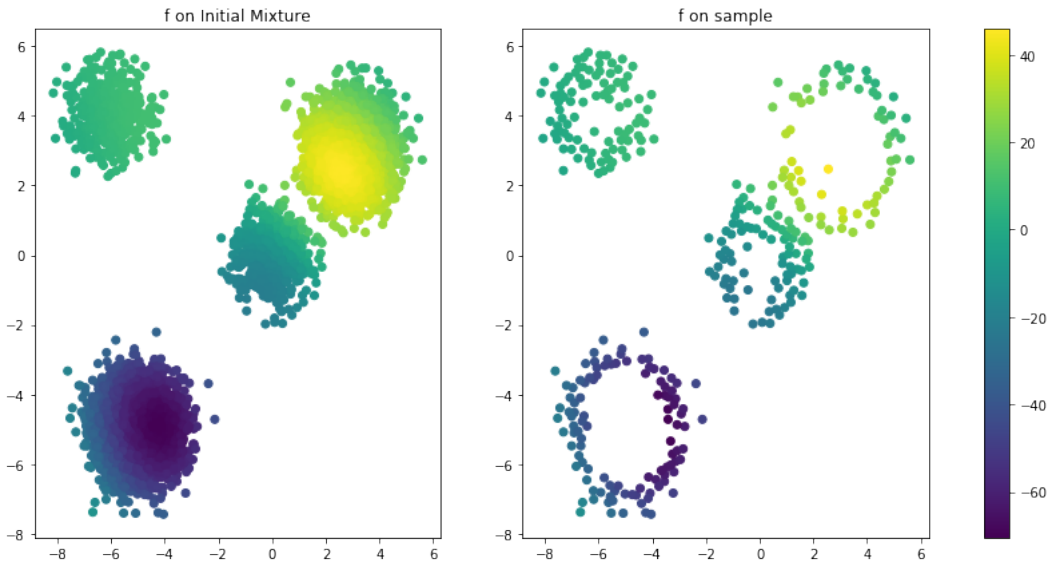


Figure 3.3: Subsampling using Gaussian Random Field with Gaussian kernel with 334 points

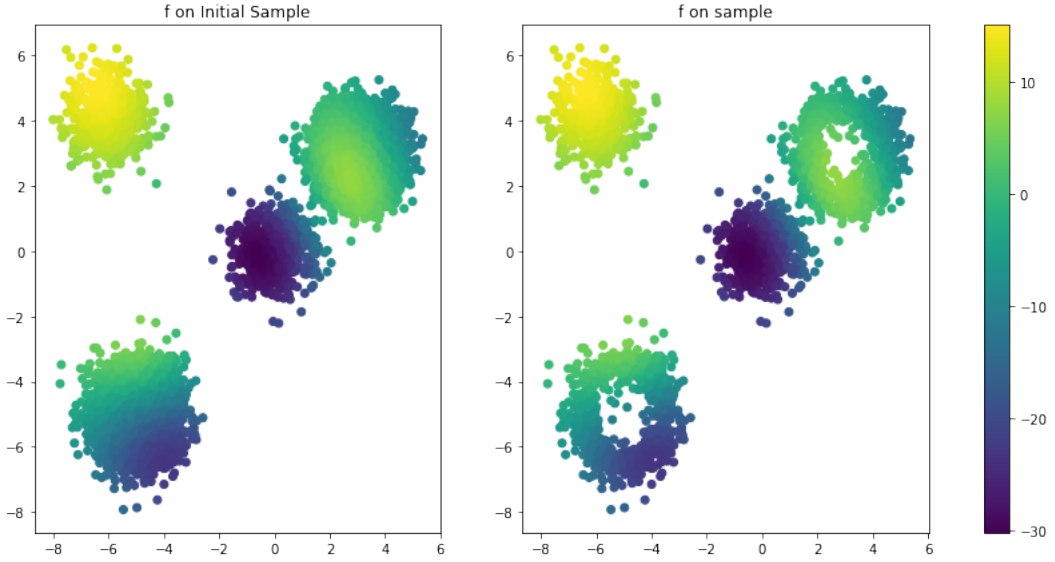


Figure 3.4: Subsampling using Gaussian Random Field with Gaussian kernel with 1,784 points

To prevent that from happening, we can try several things. Firstly, we can try to modify the kernel, for example by multiplying it by a function with more oscillations as a cosine:

$$\forall x, y \in \mathbb{R}^d,\ K(x,y) = \exp\left(-\frac{\|x-y\|^2}{a}\right)\cos(5a\|x-y\|)$$

By doing so, it is quite possible that we lose the kernel properties, however we have note demosntrated this assumption.

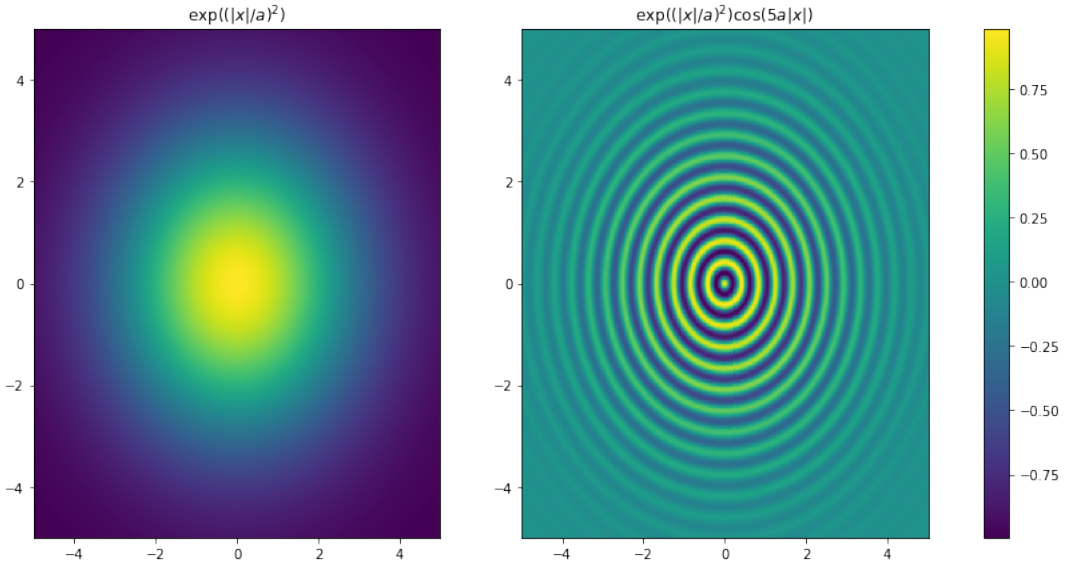


Figure 3.5: Variation of both kernels

	Initial Dataset	$b = a$	$b = \frac{a}{10}$	$b = \frac{a}{20}$	$b = \frac{a}{40}$	$b = \frac{a}{50}$	$b = \frac{a}{100}$
Cluster 1	4,942	1	79	150	399	582	2,005
Cluster 2	534	0	47	118	279	348	481
Cluster 3	4,041	1	71	141	408	583	1,881
Cluster 4	483	1	51	120	277	327	433
Total	10,000	3	248	529	1,363	1,840	4,800

Table 3.3: Points by cluster for Cosine Kernel

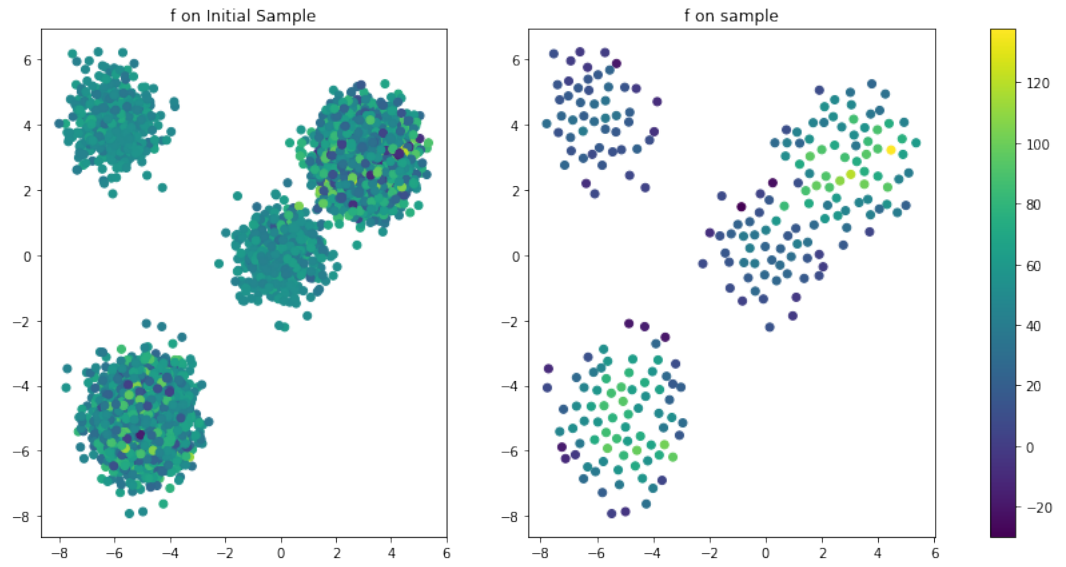


Figure 3.6: Subsampling using Gaussian Random Field with kernel (3.2.1) with 248 points

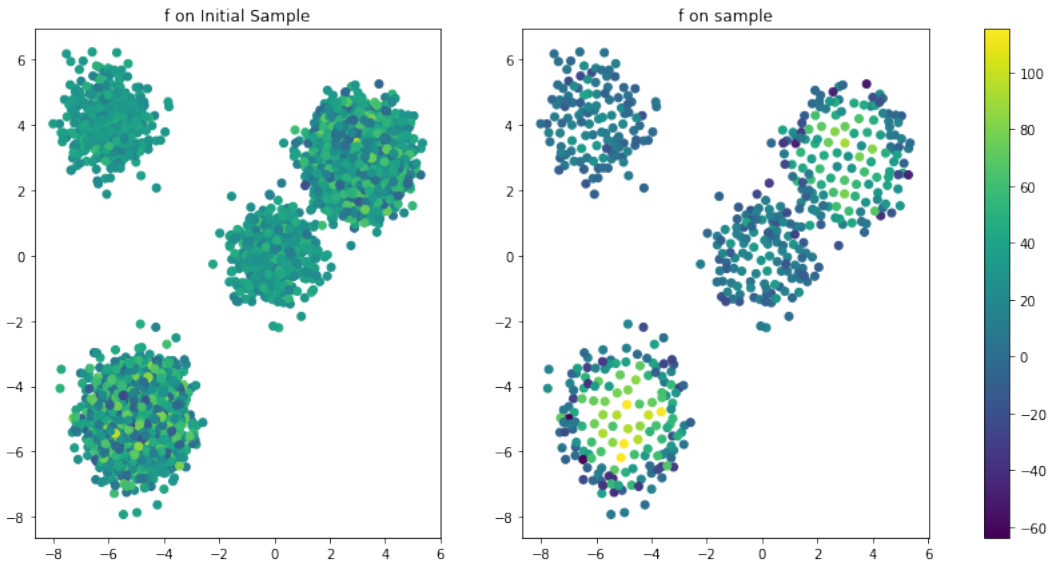


Figure 3.7: Subsampling using Gaussian Random Field with kernel (3.2.1) with 529 points

We observe that it seems to keep the points more evenly spread out in every area. The fast oscillations of the "kernel" allow more fluctuation in each cluster which helps the sampling. Indeed, the local maxima are better distributed in each area so the propagation of the maximum is less visible.

3.2.2 Greedy Algorithm

We could also use different ideas for the neighborhood. A first idea was to pick the sample in a greedy way. We would take at first the global maximum and add it to the sample. Then, we would remove all the points in the ball of radius b. And then reiterate. The advantage of doing that is to prevent the problem of the maximum propagation.

Algorithm 4: Greedy Algorithm of Subsampling in $\mathbb{R}^d$

Result: Subsample S
Initialization: A sample of points X, a kernel k, a radius r, a radius b
- $\forall x \in X,\ f_0(x) \sim \mathcal{N}(0,1)$
- $\forall x \in X,\ f(x) = \sum_{y \in X} f_0(y) k(x,y) \mathbb{1}_{\{\|y-x\| \leq r\}}$
- $S = \emptyset$
- $M = X$
while $M \neq \emptyset$ **do**
  $x_0 = max_{x \in M} f(x)$
  $S = S \cup \{x\}$
  foreach $y \in B(x,b)$ **do**
    $M = M \setminus \{y\}$
  end
end

	Initial Dataset	$b = a$	$b = \frac{a}{10}$	$b = \frac{a}{20}$	$b = \frac{a}{40}$	$b = \frac{a}{50}$	$b = \frac{a}{100}$
Cluster 1	4,942	3	154	430	1,064	1,375	2,681
Cluster 2	534	2	78	186	329	384	488
Cluster 3	4,041	2	137	394	995	1,264	2,399
Cluster 4	483	2	82	183	316	352	438
Total	10,000	9	451	1,193	2,704	3,375	6,006

Table 3.4: Points by cluster for Greedy Neighborhood

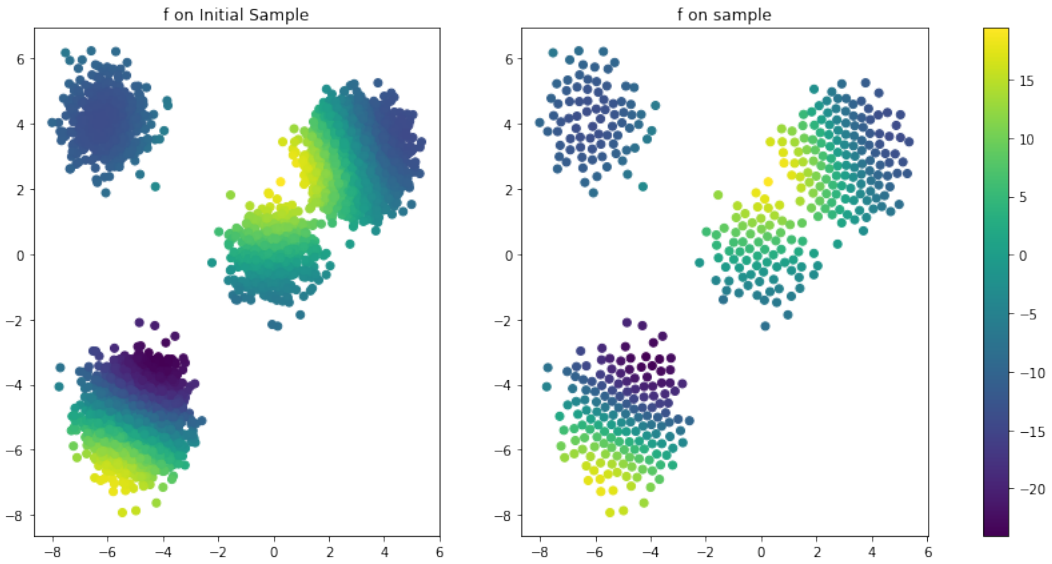


Figure 3.8: Greedy Subsampling using Gaussian Random Field with Gaussian kernel and with 451 points

We observe in the table 3.4 that this method seems to take more points compared to the other neighborhood method with the same parameters. Moreover, it seems to sample proportionally less in the areas with fewer points compared to the previous algorithm. However, it seems to properly avoid the maximum propagation problem.

3.2.3 Complexity

One of the main interests of a subsampling method is that it should be fast, and therefore have a satisfying complexity. Indeed, if the subsampling is longer than the initial algorithm on the whole dataset, then it is rather useless.

For algorithm 3, the first thing to compute is the distance and the kernel values between points which is $O(n^2 d)$ if it is done directly. And then, the whole complexity is $O(n^2 d)$ which is disastrous, especially when n is huge.

However, one can use data structures such as *k-d tree* (Bentley [1975]) or *range tree* (Bentley [1978]) which can improve it. Indeed, the *k-d-tree* can be constructed in $O(n \log(n))$ and the range search can be done in $O(dn^{1-\frac{1}{d}})$. So, the whole complexity would be $O(n \log(n) + dn^{2-\frac{1}{d}})$. This complexity is better but unfortunately does not scale well with high dimensions as it comes back to the same $O(dn^2)$ when d is big. The *range tree* can be constructed in $O(n \log(n)^{d-1})$, but range queries can be performed in

$O(k + \log(n)^{d-1})$ (Chazelle [1990]) with k the number of points returned by a query (so in the worst case k is n). It would give a complexity of $O(n\log(n)^{d-1} + nk)$ which is better, but can also be huge in high dimensions.

3.2.4 Conclusion

As we saw, the algorithm is promising as it allows in the right conditions to subsample a dataset properly and has a pretty appealing complexity compared to DPP subsample algorithms, but unfortunately it suffers from several drawbacks. Indeed, on one hand, it is not practical to choose well the parameters in order to fit a dataset. On the other hand, it is not possible either to choose directly the number of samples that we want. Of course, it is not crippling, if it returns too much sample, we can always use a k-DPP (3.1.3) on the subsample to get the number of sample that we want. Indeed, it would be more convenient than sampling directly a k-DPP if there are too many samples in the initial dataset. Another drawback is the difficulty to analyse the number of samples that it will return as it depends a lot on the dataset. A consequence is the difficulty to have coreset guarantees on the results returned. Moreover, if there are areas of the dataset which are at different scales, then having good parameters is impossible if we use algorithm 3. For these different reasons, we have then tried it on another data structure with natural neighborhood properties such as graphs in section 4.

3.3 Application on Online Type of Expectation Maximization Algorithms

3.3.1 Expectation Maximization

A well known algorithm for soft clustering is the *Expectation-Maximization* (EM) (Dempster et al. [1977]). Initially, it consists of computing the maximum likelihood estimator by introducing latent variables. Let us denote θ some unknown parameter that we want to estimate, let $X = (x_i)_{1\leq i\leq n}$ be the observations and Z be the latent variables. Then:

$$p(X|\theta) = \prod_{i=1}^{n} p(x_i|\theta) = \prod_{i=1}^{n} \int p(x_i, z_i|\theta)\nu(dz_i)$$

$$\begin{aligned}\log(p(X|\theta)) = \sum_{i=1}^{n} \log(p(x_i|\theta)) &= \sum_{i=1}^{n} \log\Big(\int p(x_i, z_i|\theta)\nu(dz_i)\Big) \\ &\geqslant \sum_{i=1}^{n} \int \log\Big(\frac{p(x_i, z_i)}{f_\theta(z_i)}\Big) f_\theta(z_i)\nu(dz_i)\end{aligned}$$

where $\int f_\theta(z)\nu(dz) = 1$, by the Jensen inequality.

It can be shown that the right part of the inequality is maximum when $f_\theta(z) = p(z|X, \theta)$. The algorithm then consists of maximizing this term with respect to θ. It is composed of two steps, the first which computes the term to be maximized, and the second which

updates the parameters in order to maximize the term.

Algorithm 5: Expectation-Maximization Algorithm

while $|\theta^{(t+1)} - \theta^{(t)}| \leq \epsilon$ **do**

- *E-step:* Compute $Q(\theta|\theta^{(t)}) = \mathbb{E}_Z[\log(p(X, Z|\theta))|X, \theta^{(t)}]$
- *M-step:* $\theta^{(t+1)} = \underset{\theta}{\operatorname{argmax}}\ Q(\theta|\theta^{(t)})$

end

We will focus in what follows on the particular case of a Gaussian Mixture Model with K classes. In this case, the parameters are $\theta = ((\alpha_k)_k, (\mu_k)_k, (\Sigma_k)_k)$.

$$\begin{cases} \forall k \in \{1, ..., K\},\ P(Z = k) = \alpha_k \\ \forall i \in \{1, ..., n\},\ x_i|z_i = k \sim \mathcal{N}(\mu_k, \Sigma_k) \\ \forall i \in \{1, ..., n\},\ p(x_i) = \sum_{k=1}^{K} p(x_i, k) = \sum_{k=1}^{K} p(x_i|z = k)p(z = k) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(x_i; \mu_k, \Sigma_k) \end{cases}$$

And the algorithm writes itself as:

Algorithm 6: Expectation-Maximization Algorithm for GMM

while $|\theta^{(t+1)} - \theta^{(t)}| \leq \epsilon$ **do**

- *E-step:* Compute $\forall i \in \{1, ..., n\}, \forall k \in \{1, ..., K\}$,

$$P(z_i = k|x_i, \theta^{(t)}) = \frac{\mathcal{N}(x_i; \mu_k^{(t)}, \Sigma_k^{(t)})\alpha_k^{(t)}}{\sum_l \mathcal{N}(x_i; \mu_l^{(t)}, \Sigma_l^{(t)})\alpha_l^{(t)}}$$

- *M-step:* Compute $\forall k \in \{1, ..., K\}$,

$$\begin{cases} \mu_k^{(t+1)} = \frac{\sum_{i=1}^{n} x_i P(z_i=k|x_i,\theta^{(t)})}{\sum_{i=1}^{n} P(z_i=k|x_i,\theta^{(t)})} \\ \Sigma_k^{(t+1)} = \frac{\sum_{i=1}^{n} (x_i-\mu_k^{(t+1)})(x_i-\mu_k^{(t+1)})^T P(z_i=k|x_i,\theta^{(t)})}{\sum_{i=1}^{n} P(z_i=k|x_i,\theta^{(t)})} \end{cases}$$

end

Then, we can assign each point x_i to the class Z_i such that: $Z_i = \underset{z}{\operatorname{argmax}}\ p(z|x_i)$.

To illustrate this algorithm, we applied it on a Gaussian mixture following the same parameters as (3.1). We applied the EM algorithm on the initial mixture and on a subsample of 514 points obtained with our algorithm and the kernel (3.2.1), and we plotted the clustering given by the algorithm, as well as the centers and the variances as ellipses.

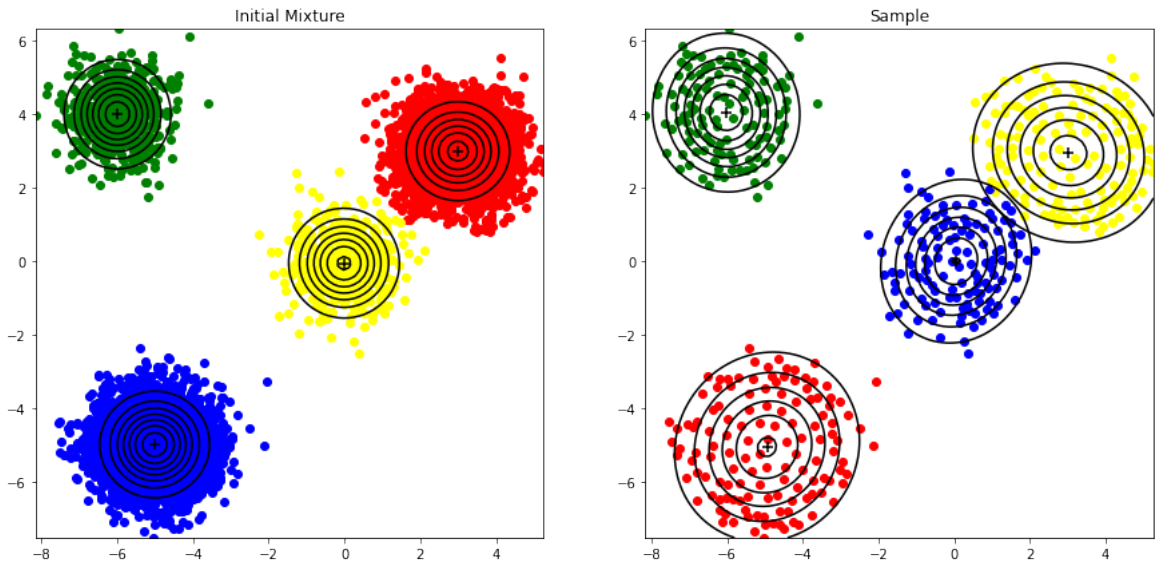


Figure 3.9: Example of EM

And we obtained the following estimated parameters:

	μ	Σ	p
C_1	(-5.01;-5)	$\begin{pmatrix} 0.51 & 0.002 \\ 0.002 & 0.51 \end{pmatrix}$	0.5
C_2	(2.99;3)	$\begin{pmatrix} 0.5 & 0 \\ 0 & 0.48 \end{pmatrix}$	0.41
C_3	(-5.99;4.01)	$\begin{pmatrix} 0.48 & 0 \\ 0 & 0.54 \end{pmatrix}$	0.05
C_4	(-0.01;-0.06)	$\begin{pmatrix} 0.54 & -0.002 \\ -0.002 & 0.56 \end{pmatrix}$	0.05

Table 3.5: EM on Full Dataset

	μ	Σ	p
C_1	(0.05;0)	$\begin{pmatrix} 1.01 & 0.1 \\ 0.1 & 1.27 \end{pmatrix}$	0.24
C_2	(-4.94;-5.06)	$\begin{pmatrix} 1.64 & 0.12 \\ 0.12 & 1.84 \end{pmatrix}$	0.25
C_3	(-6.02;4.05)	$\begin{pmatrix} 0.92 & -0.02 \\ -0.02 & 1.14 \end{pmatrix}$	0.23
C_4	(3.02;2.95)	$\begin{pmatrix} 1.71 & -0.09 \\ -0.09 & 1.59 \end{pmatrix}$	0.28

Table 3.6: EM on Subsample

We can notice that the centers are pretty well estimated even with the subsample. On the other hand, as expected, the weights are completely wrong because each cluster has approximately the same number of points in the subsample. We see that the covariance matrices are not very well estimated either, because the concentration of points was lost. This could be interesting to use if we were only interested in the centers of the clusters. Indeed, the EM algorithm runs much faster on a smaller dataset as the complexity is in $O(nk)$. But unfortunately, the complexity of the subsampling is bigger than the one of the EM so it is pretty useless to subsample the data using this technique.

3.3.2 Classification Expectation Maximization

Modifications of the EM algorithm were proposed in order to perform hard clustering. An example of such an algorithm is the *Classification Expectation Maximization* (CEM) algorithm, which was proposed in Celeux and Govaert [1992]. This algorithm aims at maximizing the following criterion:

$$L(z,\theta) = \sum_{k=1}^{K}\sum_{i=1}^{n} \log(\alpha_k p(x_i|\theta_k))\mathbb{1}_{\{z_{ik}=1\}}$$

with $z_{ik} = 1$ if x_i is in the cluster k.

We can notice that it looks a lot like the K-Means algorithm. Indeed, the K-Means algorithm is actually a special case where the weights are all equal and the density is a Gaussian with an isotropic covariance matrix (Celeux and Govaert [1992]). Then, we understand that this algorithm allows more flexibility for the clusters as they do not need to be isotropic and they can have different weights.
The CEM algorithm is then:

Algorithm 7: Classification Expectation-Maximization Algorithm

while $|\theta^{(t+1)} - \theta^{(t)}| \leq \epsilon$ **do**

- *E-step:* Compute $\forall i \in \{1,...,n\}, \forall k \in \{1,...,K\}$,

$$P(z_i^{(t)} = k|x_i, \theta_k^t) = \frac{\alpha_k^{(t)} p(x_i;\theta_k^{(t)})}{\sum_{l=1}^{K} \alpha_l^{(t)} p(x_i;\theta_l^{(t)})}$$

- *C-step:* $\forall i \in \{1,...,n\}$, $z_i^{(t)} = \underset{1\leq k\leq K}{\operatorname{argmax}}\ P(z_i = k|x_i, \theta_k^{(t)})$
- *M-step:* Compute $\forall k \in \{1,...,K\}$,

$$\begin{cases} \alpha_k^{(t+1)} = \frac{\sum_{i=1}^{n} \mathbb{1}_{\{z_i^{(t)}=k\}}}{n} \\ \mu_k^{(t+1)} = \frac{1}{\sum_{i=1}^{n} \mathbb{1}_{\{z_i^{(t)}=k\}}} \sum_{i=1}^{n} x_i \mathbb{1}_{\{z_i^{(t)}=k\}} \\ \Sigma_k^{(t+1)} = \frac{1}{\sum_{i=1}^{n} \mathbb{1}_{\{z_i^{(t)}=k\}}} \sum_{i=1}^{n} (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T \mathbb{1}_{\{z_i^{(t)}=k\}} \end{cases}$$

end

In the case of Gaussian Mixture Model, $\theta = ((\mu_k)_k, (\Sigma_k)_k)$ and $p(x_i|\theta_k) = \mathcal{N}(x_i; \mu_k, \Sigma_k)$.

3.3.3 Online Classification Expectation Maximization

In the case where the dataset becomes too large, or when the data are received sequentially, online versions of the precedent algorithms were introduced. For example, in Samé et al. [2007], the authors presented a stochastic version of the CEM (OCEM). These algorithms are known to perform pretty well, but most importantly, to run very fast compared to the classical EM algorithms, as they loop only once for each point, which should be pretty efficient.

This algorithm consists first of running the classical CEM on an initial dataset containing n_0 observations. In Samé et al. [2007], they analyze the impact of n_0, but they use uniform samples as initial dataset. Our intuition here is that a good initialization which samples well from the whole dataset will give better results for the OCEM than an

uniform sample. Then, they take care of all other points one by one, classify them and update the parameters using a stochastic gradient algorithm.

Algorithm 8: Online Classification Expectation-Maximization Algorithm

Initialization: n_0: number of initialization points
Sample n_0 points with the algorithm of your choice in S_0
Initialize μ, α, Σ
$t = n_0$
foreach $x \notin S_0$ **do**
- $t = t + 1$
- *E-step:* Compute $\forall k \in \{1, ..., K\}$,

$$P(z_{x,k}^{(t)} = 1 | x, \theta_k^{(t-1)}) = \frac{\alpha_k^{(t-1)} \mathcal{N}(x_i; \mu_k^{(t-1)}, \Sigma_k^{(t-1)})}{\sum_{l=1}^{K} \alpha_l^{(t-1)} \mathcal{N}(x_i; \mu_l^{(t-1)}, \Sigma_l^{(t-1)})}$$

- *C-step:* $k^* = \underset{1 \leq k \leq K}{\operatorname{argmax}} \; P(z_{x,k}^{(t)} = 1 | x, \theta_k^{(t-1)})$
- $z_{x,k}^{(t)} = \begin{cases} 1 & \text{if } k = k^* \\ 0 & \text{otherwise} \end{cases}$
- *M-step:* Compute $\forall k \in \{1, ..., K\}$,

$$\begin{cases} n_k^{(t)} = n_k^{(t-1)} + z_{x,k}^{(t)} \\ \alpha_k^{(t)} = \frac{1}{t}(z_{x,k}^{(t)} - \alpha_k^{(t-1)}) \\ \mu_k^{(t)} = \mu_k^{(t-1)} + \frac{z_{x,k}^{(t)}}{n_k^{(t)}}(x - \mu_k^{(t-1)}) \\ \Sigma_k^{(t)} = \Sigma_k^{(t-1)} + \frac{z_{x,k}^{(t)}}{n_k^{(t)}}\left(\left(1 - \frac{z_{x,k}^{(t)}}{n_k^{(t)}}\right)(x - \mu_k^t)(x - \mu_k^{(t)})^T - \Sigma_k^{(t-1)}\right) \end{cases}$$

end

3.3.4 Comparisons of EM Types Algorithms

In order to compare the OCEM with our subsampling as an initial set with the other EM algorithms, we created different scenarios such as overlapping clusters, well separated clusters, in high or low dimension. These tests are of course not exhaustive as we restrained ourselves on Gaussian mixtures. For each scenario, we generated 50 datasets and plotted a box plot of the time and of the adjusted Rand index (ARI) (Hubert and Arabie [1985]). The box extends between the lower and upper quartile, and an orange line represents the median. The adjusted Rand index allows us to compare a clustering with the ground truth clustering. It takes values between 0 and 1. If it equals 1, it means that the partitions are identical, up to permutations.

The scenarios which were generated are the following:

- **Scenario A:** In 2D with 10,000 points:

	μ	Σ	p
Cluster 1	(-5;-5)	0.5 I_2	0.5
Cluster 2	(0;0)	0.5 I_2	0.05
Cluster 3	(5;5)	0.5 I_2	0.4
Cluster 4	(-6;4)	0.5 I_2	0.05

Table 3.7: Scenario A

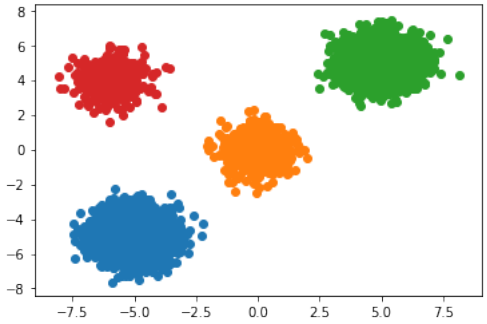


Figure 3.10: Scenario A

- **Scenario B:** In 2D with 10,000 points:

	μ	Σ	p
Cluster 1	(-5;-5)	0.5 I_2	0.5
Cluster 2	(0;0)	0.5 I_2	0.05
Cluster 3	(1;1)	0.5 I_2	0.4
Cluster 4	(-6;-2)	0.5 I_2	0.05

Table 3.8: Scenario B

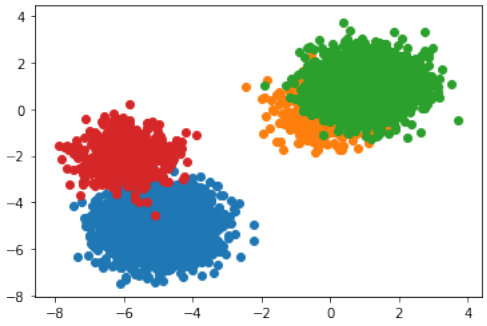


Figure 3.11: Scenario B

- **Scenario C:** In 2D with 10,000 points:

	μ	Σ	p
Cluster 1	(-5;-5)	$\begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$	0.5
Cluster 2	(0;0)	$\begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}$	0.05
Cluster 3	(1;4)	$\begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$	0.4
Cluster 4	(-6;-2)	$\begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix}$	0.05

Table 3.9: Scenario C

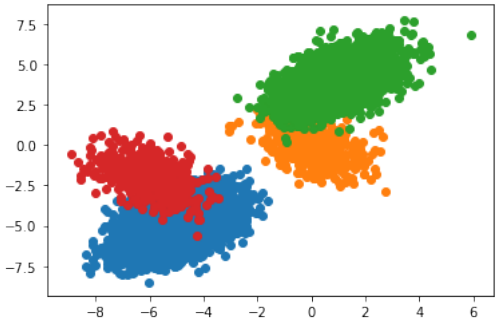


Figure 3.12: Scenario C

- **Scenario D:** In dimension 100 with 10,000 points:

	μ	Σ	p
Cluster 1	(-5;5;0;...)	I_2	0.5
Cluster 2	(0;0;0;...)	I_2	0.05
Cluster 3	(1;4;0;...)	I_2	0.4
Cluster 4	(-6;-2;0;...)	I_2	0.05

Table 3.10: Scenario D

In what follows, we present the results we obtained on the different scenarios. We compared the EM algorithm from scikit-learn (Pedregosa et al. [2011]) with CEM and OCEM that we implemented. The implementation of the EM algorithm from scikit-learn is probably more efficient than the one we would have done, and it can explain some difference in the execution time between the EM and the CEM. We denoted by "*Unif Sample*" the OCEM with an uniform subsample as initial dataset, "*Our Sample*" the OCEM with an initial subsample obtained with our algorithm, and "*DPP Sample*" the OCEM with an initial subsample obtained with a DPP using the toolbox DPPy.

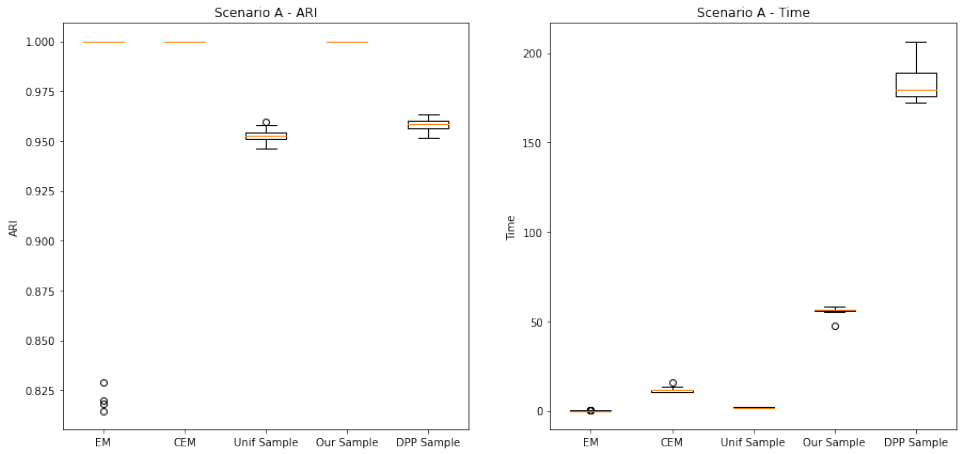


Figure 3.13: Results Scenario A

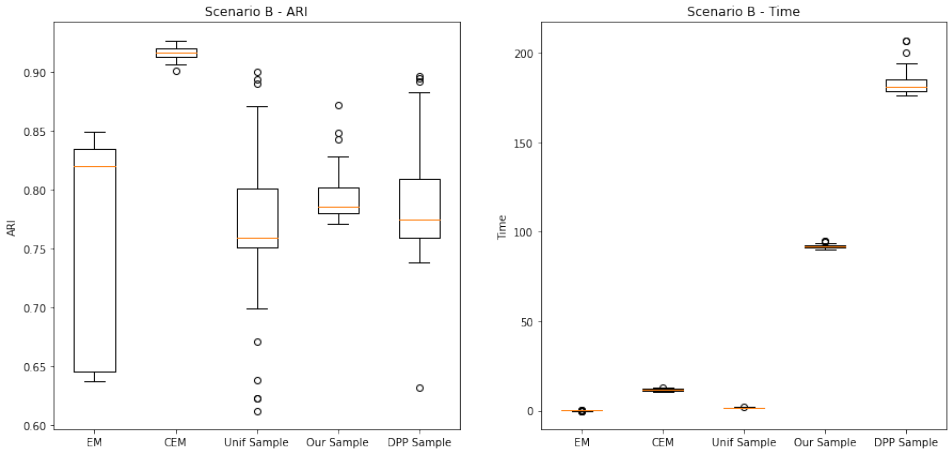


Figure 3.14: Results Scenario B

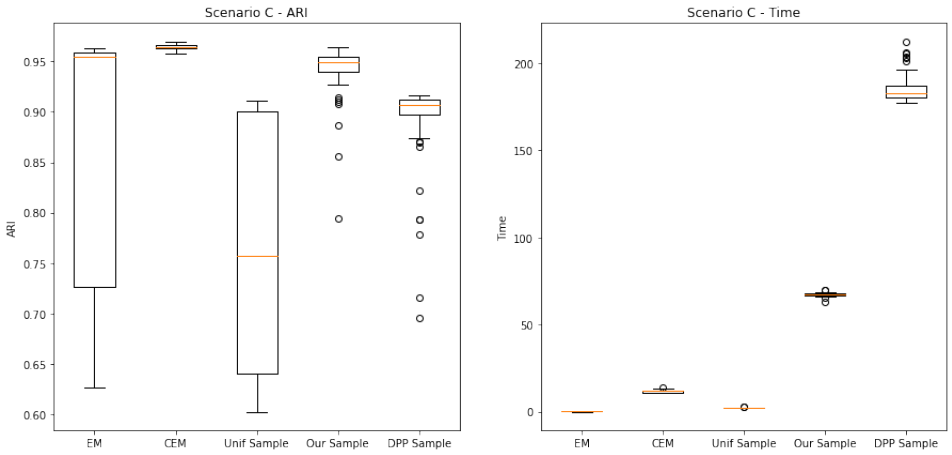


Figure 3.15: Results Scenario C

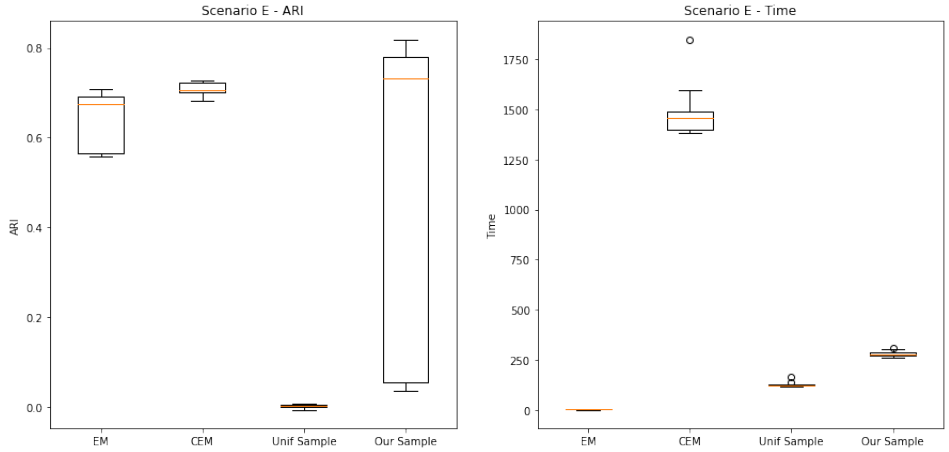


Figure 3.16: Results Scenario D

We observe that in terms of performance with the ARI indicator, our subsampling as an initial dataset followed by the OCEM performs quite well. Indeed, for the scenario A (3.7,3.13) where the clusters are totally untied, our method returns an ARI of 1, same as the EM and the CEM. The OCEM with uniform and DPP sample are not able to score an ARI of 1. Unfortunately, the execution time of our subsampling seems much bigger than with the other algorithms, but the implementation can most likely be improved with Cython in order to be more competitive. We observe more or less the same for the scenario B (3.8,3.14) and C (3.9,3.15) where the dimension is still 2. The ARI obtained with our initialization of OCEM is competitive with the EM and CEM and better than the uniform or the DPP initialization of OCEM. For the DPP subsampling, we were only able to test it on a computer with 16GB of ram as the kernel takes a lot of memory. In high dimension with scenario D (3.10,3.16), we still observe a good ARI for our subsampling initialization. Furthermore, we see that the time of execution is better than the CEM. However, we cannot make conclusions from that since the implementation of the CEM is not as efficient as if it were done with Cython as the EM of scikit-learn. This implementation probably just does not scale well with the dimension.

4 Application to Graphs

This section will first introduce the motivations of sampling on graphs, and in a second step, existing algorithms allowing to do that. We will also see that it is not easy to assess the quality of a subsample. Then, we will see how to adapt our algorithm to graph setting, before comparing it as an initialization subsample of the online classification EM algorithm (Zanghi et al. [2008]) with others clustering algorithms.

We will denote in what follows an undirected graph $G = (V, E)$ where V is a set of n vertices $(v_i)_{1 \leq i \leq n}$ and E is the set of edges of the graph, ie $v_i, v_j \in V$ are connected in the graph if $(v_i, v_j) \in E$. We will most of the time restrain ourselves to undirected graphs with no self loop.

4.1 Current Methods and Motivations

4.1.1 Motivations

Graphs are used in lots of areas such as social network analysis, bioinformatics, systems biology, networks... But some of these graphs are really huge and present lots of challenges to be able to deal with them. Some of them cannot even be stored in the memory of a computer. It implies that it is impossible to study the entirety of this type of graph at once. As noted in Hübler et al. [2008], it is needed in some fields such as routing protocols to run simulations algorithms on these large graphs. Unfortunately, when the size of the graph is too important, it can often be too computationally expensive and the runtime will often be really long as it will scale in polynomial time with the number of nodes. And when we want to find the characteristics of a graph, it is not directly possible when we cannot even store the graph in the memory.

A solution of these problems is to use better algorithms which can scale well with the size of the graph. But these improvements can unfortunately be at the cost of the precision because they will most likely require heuristics. The other solution is to find a subgraph through a subsample of the graphs, which will have the same characteristics of the full graph, and on which we will either be able to apply our expensive algorithms or to compute the needed characteristics.

In Leskovec and Faloutsos [2006] and in Hübler et al. [2008], the authors review, propose and compare methods to get a "representative" sample of the original graph. But comparing graphs is not trivial. And defining what is a good sample is also not obvious. In these articles, they rely on general properties such as the degree distribution or the clustering coefficient distribution in order to compare their sampling methods. But other methods exist which will be able to estimate a specific property of a graph.

Since the first papers on the topics of subsampling by Leskovec and Faloutsos and Hübler et al., other authors have studied the subsampling on graphs and have adapted it to others problems such as the evaluation of clustering techniques on a subgraph in Zhang et al. [2020] or have proposed new subsampling strategies which produce subgraphs representative of community network in the original graph as in Maiya and Berger-Wolf [2010].

In the next part, we will present some of these methods to which we will be able to compare our algorithm. For illustration purposes, we will display the results of the different methods on an initial graph which is a stochastic block model.

Definition 4.1. Stochastic Block Model: A stochastic block model is a random graph characterized by a number of nodes n, a partition of these nodes in Q different clusters, and a symmetric QxQ matrix P containing edge probabilities between the clusters. Then, the edges are sampled such that if $u \in C_i$ and $v \in C_j$, then there is an edge between u and v with probability P_{ij}.

So, we will use a stochastic block model with 200 nodes, and 3 clusters containing respectively 100, 60 and 40 nodes. In a next part, we will then compare the algorithms with our own, using concrete criteria.

4.1.2 Existing Methods

In Leskovec and Faloutsos [2006], they present several methods of subsampling adapted to large graphs such as *Random Node* sampling, *Random Edge* sampling or *Random Walk* sampling, which they found out at the time to work best with their criteria. Then, other algorithms were proposed, based on the same criteria or others.

4.1.2.1 Random Node Sampling

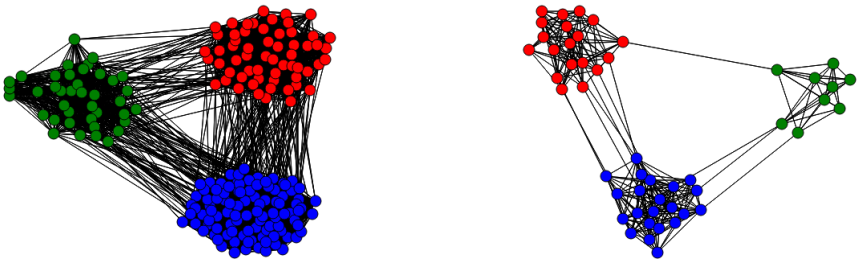
The easiest method, and maybe the more natural, is the so called *Random Node* sampling algorithm which simply consists in sampling random nodes uniformly, and then to use the induced graph as a subgraph, meaning that if we denote $S \subseteq V$ the subsample of vertices, then the induced subgraph is $\tilde{G} = (S, E[S])$ where

$$E[S] = \{(v_i, v_j) | v_i, v_j \in S, (v_i, v_j) \in E\}$$

In Stumpf et al. [2005], it was shown that this algorithm does not keep power-law distribution. However, Leskovec and Faloutsos have remarked that it works surprisingly well. It is indeed a really simple algorithm and we could therefore expect that it does not retain the properties well, but such is not the case.

Algorithm 9: Random Node Sampling

Result: Subgraph $\tilde{G} = (S, \tilde{E})$
Initialization: A graph $G = (V, E)$, d: the number of nodes of the subgraph
- Pick uniformly at random $u_1, \ldots, u_d \in V$
- $S = \{u_1, ..., u_d\}$
- Let $\tilde{E} = \emptyset$
foreach $e = (u, v) \in E$ **do**
  if $u \in S$ and $v \in S$ **then**
    $\tilde{E} = \tilde{E} \cup \{(u, v)\}$
end



(a) 200 nodes **(b)** Subgraph: 50 nodes

Figure 4.1: Example of *Random Node* sampling

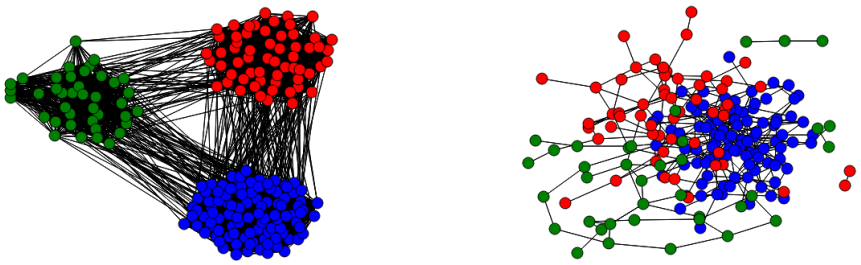
4.1.2.2 Random Edge Sampling

Another very simple method of sampling is the *Random Edge* sampling algorithm which consists in uniformly sampling edges and then keeping the nodes which are linked to the selected edges. Unfortunately, these types of graphs will not keep community structures as there will be fewer edges than with random nodes.

There are lots of variety of this type of subsampling strategy, such as *Random Node-Edge* sampling which consists in first picking uniformly a node and then uniformly picking an incident edge. But they globally do not work well to find general representative subsamples.

Algorithm 10: Random Edge Sampling

Result: Subgraph $\tilde{G} = (S, \tilde{E})$
Initialization: A graph $G = (V, E)$, d: the number of edges of the subgraph
- Pick uniformly at random $e_1, ..., e_d \in E$
- $\tilde{E} = \{e_1, ..., e_d\}$
- Let $S = \emptyset$
foreach $e \in \tilde{E}$ **do**
  $e = (u, v)$
  $S = S \cup \{u, v\}$
end



(a) 5006 edges **(b)** Subgraph: 300 edges

Figure 4.2: Example of *Random Edge* sampling

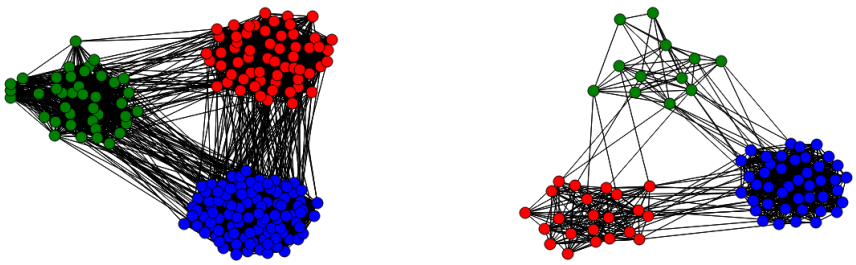
4.1.2.3 Random Walk

A well-known family of subsampling algorithms consists in applying a random walk from an initial uniformly picked-at-random node on the graph and in adding the visited

nodes in the subgraph. Of course, there may be problems if there are sinks for example. But we can resolve these e.g. by adding with some probability the possibility to do random jumps as in the Pagerank algorithm of Google (Page et al. [1999]). In Leskovec and Faloutsos [2006], they found that this method worked best to keep characteristics such as degree distribution or coefficient clustering distribution.

Algorithm 11: Random Walk with Random Jump

Result: Subgraph $\tilde{G} = (S, \tilde{E})$
Initialization: A graph $G = (V, E)$, n: the number of iterations, c: the random jump probability
- Pick uniformly a node $u \in V$
- Let $S = \{u\}$
for i=0 to n **do**
  $U \sim Unif([0, 1])$
  if $U < c$ **then**
    Pick uniformly a node $\tilde{u} \in V$
  else
    Pick $\tilde{u} \in N(u)$ with probability $\frac{1}{d_u}$
  end
  $S = S \cup \{\tilde{u}\}$
end
foreach $e = (u, v) \in E$ **do**
  if $u \in S$ and $v \in S$ **then**
    $\tilde{E} = \tilde{E} \cup \{(u, v)\}$
end



(a) 200 nodes **(b)** Subgraph: 77 nodes

Figure 4.3: Example of *Random Walk* sampling

4.1.2.4 Metropolis Hasting based Algorithms

Another approach, which has the advantage to be more flexible over the criteria we want to keep on the subgraph from the original graph, was proposed in [Hübler et al., 2008]. This approach is based on the well-known Metropolis-Hasting algorithm (Metropolis et al. [1953],Hastings [1970]) and was shown to have better results than the random walk. It consists in choosing a distance first, between the original graph and a subgraph. This distance can be for example based on the distribution of degrees. We will detail further which parameters we want to keep between the 2 graphs, and how to measure it. If we denote $\Delta(G, S)$ the chosen distance between the original graph and the sample S, then

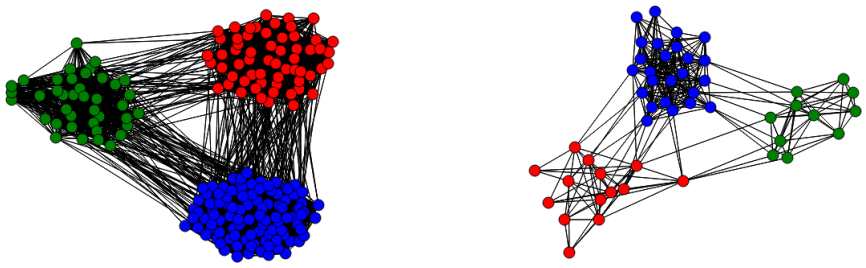
we want a sample following a density proportional to:

$$\rho^*(S) = \frac{1}{\Delta(G,S)^p}$$

where p is a parameter allowing us to sharp the distribution, or on the contrary to flatten it if we do not want to get stuck in local maxima. We can add a temperature which will decrease over time in order to explore at first all the distribution, and then to get to local maxima. Indeed, if the distance is small, it means that the sample is good, and the density will be high.

Algorithm 12: Metropolis Subgraph Sampling

Result: Subgraph $\tilde{G} = (S, \tilde{E})$
Initialization: A graph $G = (V, E)$, n: the number of iterations, d: the size of the sample, a distance function Δ, p
- $S_{current} \leftarrow$ uniformly at random from V
- $S_{best} = S_{current}$
for $i = 1$ to n **do**
  $v \sim U(S_{current})$
  $w \sim U((V \setminus S_{current}) \cup \{v\})$
  $S_{new} = (S_{current} \setminus \{v\}) \cup \{w\}$
  $\alpha \sim Unif([0,1])$
  if $\alpha < (\frac{\Delta(G,S_{current})}{\Delta(G,S_{new})})^p$ **then**
    $S_{current} = S_{new}$
    if $\Delta(G, S_{current}) < \Delta(G, S_{best})$ **then**
      $S_{best} = S_{current}$
end
foreach $e = (u, v) \in E$ **do**
  if $u \in S_{best}$ and $v \in S_{best}$ **then**
    $\tilde{E} = \tilde{E} \cup \{(u, v)\}$
end



(a) 200 nodes **(b)** Subgraph: 50 nodes

Figure 4.4: Example of *Metropolis Subgraph* Sampling

4.1.2.5 Expansion Factor based Algorithms

Then, other algorithms were proposed in order to keep other characteristics such as the community structure for example in Maiya and Berger-Wolf [2010]. In this paper, they propose to maximize the expansion factor:

Definition 4.2. Expansion Factor: The expansion factor measures the number of edges per node that point outside the sample ([Yang and Leskovec, 2015]). For a set $S \subseteq V$ of nodes, and the induced subgraph $\tilde{G} = (S, E[S])$:

$$g(S) = \frac{c_S}{|S|}$$

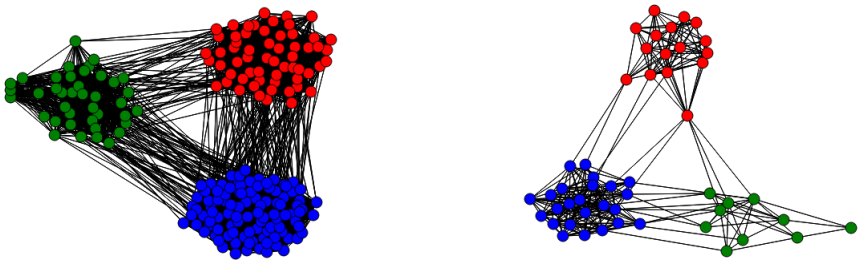
where $c_S = \{(u, v) \in E | u \in S, v \notin S\}$ corresponds to the number of edges on the boundary of S.

In order to do that, they propose the following greedy algorithm:

Algorithm 13: Snowball Expansion Sampler (XSN)

Result: Subgraph $\tilde{G} = (S, \tilde{E})$
Initialization: A graph $G = (V, E)$, d: the size of the sample
- $S \leftarrow \emptyset$
- $v \sim V$
- $S = S \cup \{v\}$
while $|S| \leq k$ **do**
  Select new node $v \in N(S)$ which maximized $|N(\{v\}) \setminus (N(S) \cup S)|$
  $S = S \cup \{v\}$
end
foreach $e = (u, v) \in E$ **do**
  if $u \in S$ and $v \in S$ **then**
    $\tilde{E} = \tilde{E} \cup \{(u, v)\}$
end

They also propose to apply the Metropolis algorithm (12) using a distance which will maximize the expansion factor such as: $\Delta(G, S) = \frac{|c_S|}{|V \setminus S|}$. We will use this algorithm instead of the XSN as we encountered some weird results when $|N(\{v\}) \setminus (N(S) \cup S)| = 0$. We will denote this algorithm the *MCMC Expansion Sampler* (XMC).



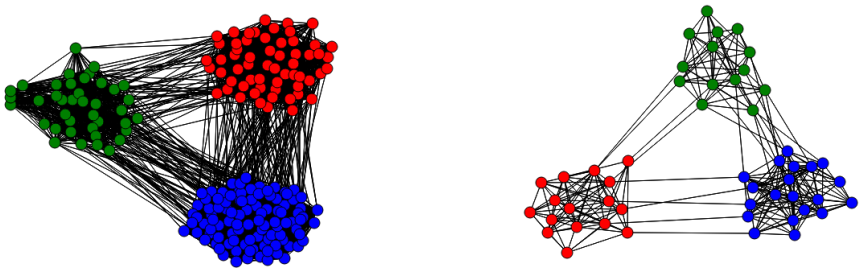
(a) 200 nodes **(b)** Subgraph: 50 nodes

Figure 4.5: Example of *MCMC Expansion Sampler* (XMC)

4.1.2.6 Determinantal Point Processes on Graphs

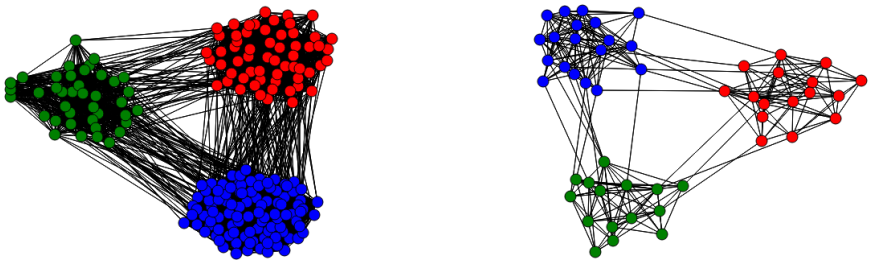
It is also possible to use determinantal point processes (section 3.1.2) to sample on graphs. Indeed, we only need a kernel matrix K or the corresponding matrix L which makes sense to apply on the graph. For example, we can try DPP with L being the

adjacency kernel ($\forall i, j \in V,\ L(i,j) = e^{-\|A_i - A_j\|^2}$), or L being the regularized Laplacian kernel (4.12).



(a) 200 nodes **(b)** Subgraph: 50 nodes

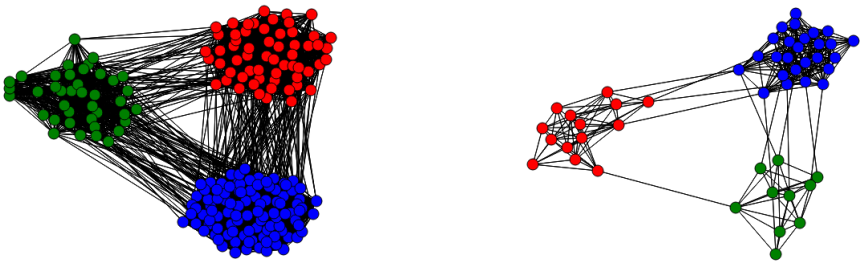
Figure 4.6: Example of *DPP* with adjacency kernel Sampling on Graphs



(a) 200 nodes **(b)** Subgraph: 50 nodes

Figure 4.7: Example of *DPP* with regularized Laplacian kernel Sampling on Graphs

In Tremblay et al. [2017], they proposed a method using the eingendecomposition of the Laplacian (4.9) in order to have the kernel K. More specifically, let $L = U\Lambda U^T$ which exists, since L is a real symmetric matrix by the spectral theorem. Then, let k be the number of nodes we want in our sample, we can take $K = Uh_{\lambda_k}(\Lambda)U^T$ with h_{λ_k} being such that $h_{\lambda_k}(\lambda) = 1$ if $\lambda \leq \lambda_k$. Then using any DPP sampler with marginal kernel K, we get our sample. They proposed also to use Wilson's algorithm with Wilson's marginal kernel when the eigendecomposition is intractable. We have to note however that the goal of subsampling in that paper is to recover a signal on the whole graph afterward, which is not really our case.



(a) 200 nodes **(b)** Subgraph: 50 nodes

Figure 4.8: Example of *DPP* Sampling on Graphs

4.1.3 Evaluation Criteria

In order to compare the algorithms, we need evaluation criteria which can be used to compare the subsamples. Unfortunately, there does not seem to be a consensus on which criteria to choose. However, we can still compare how some properties are kept in the subgraph. Indeed, in Leskovec and Faloutsos [2006], they decided to compare the quality of the subsamples by looking at the degree distribution and at the clustering coefficient distribution among others. Moreover, it can also be a criterion for community structure, because for some models such as the stochastic block models, the nodes in the same communities are supposed to have in average the same degrees.

It is interesting to keep the same degree distribution because firstly, it means that we have the same type of graphs. Secondly, it is really easy to compute as we can do it in $O(n \cdot d_{avg})$ using adjacency lists.

In order to compare distributions, we can use for example as in Leskovec and Faloutsos [2006] the Kolmogorov-Smirnov distance:

Definition 4.3. Kolmogorov-Smirnov distance: By denoting F_n the empirical cumulative distribution function of the initial dataset such that $\forall x,\ F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{X_i \leq x\}}$, and F'_k the cumulative distribution of the subsample. Then, the Kolmogorov-Smirnov distance is:

$$d(F_n, F'_k) = \sup_x |F_n(x) - F'_k(x)|$$

For example, if we want to compare the degree distributions of two samples, then we can take $\forall i \in V,\ X_i = d_i$. Of course, if we want to have significant results, we first have to normalize the distributions. Indeed, the subgraph will obviously have lower degrees, so the distance would always be equal to 1 if the degrees were not normalized.

The clustering coefficient (Watts and Strogatz [1998]) of a node is intuitively the proportion of triangles that exist with this node. More formally, it is:

$$\begin{aligned}
\forall u \in V,\ C(u) &= \frac{\sum_{v \neq w} \mathbb{1}_{\{(u,v)\in E\}} \mathbb{1}_{\{(u,w)\in E\}} \mathbb{1}_{\{(v,w)\in E\}}}{\sum_{v \neq w} \mathbb{1}_{\{(u,v)\in E\}} \mathbb{1}_{\{(u,w)\in E\}}} \\
&= \frac{\sum_{v \neq w} \mathbb{1}_{\{(u,v)\in E\}} \mathbb{1}_{\{(u,w)\in E\}} \mathbb{1}_{\{(v,w)\in E\}}}{\sum_{v \in V} \mathbb{1}_{\{(u,v)\in E\}} \sum_{w \neq v} \mathbb{1}_{\{(u,w)\in E\}}} \\
&= \frac{\sum_{v \neq w} \mathbb{1}_{\{(u,v)\in E\}} \mathbb{1}_{\{(u,w)\in E\}} \mathbb{1}_{\{(v,w)\in E\}}}{\sum_{v \in V} \mathbb{1}_{\{(u,v)\in E\}} (d_u - 1)} \\
&= \frac{\sum_{v \neq w} \mathbb{1}_{\{(u,v)\in E\}} \mathbb{1}_{\{(u,w)\in E\}} \mathbb{1}_{\{(v,w)\in E\}}}{d_u(d_u - 1)}
\end{aligned}$$

It is supposed to represent a local clustering. Then, we denote C_d the vector containing the average of C_v over all vectors of degree d, ie $\forall d,\ C_d = \frac{1}{d}\sum_{u \in V} C_u \mathbb{1}_{\{d_u = d\}}$. In order to compare this vector, I used as in [Hübler et al., 2008] the average absolute difference: $\frac{\sum_{d \in \mathcal{M}} |C_d - C'_d|}{|\mathcal{M}|}$, with $\mathcal{M}$ denoting the set of existing degrees.

To check whether or not we keep the correct community structures, we have first to define what is a good community structure. Yang and Leskovec wrote that, "intuitively, a

"good" community is cohesive, compact, and internally well connected while being also well separated from the rest of the network." But they concluded the article by mentioning that different definitions of network communities might be appropriate depending on the different networks. So, in order to compare how well the algorithms keep the community structures, we tried to use several criteria such as the separability, the density, the conductance or the modularity:

Definition 4.4. Separability (Yang and Leskovec [2015]): The separability captures the intuition that communities are well separated. For a set $S \subseteq V$ of nodes, and the induced subgraph $\tilde{G} = (S, E[S])$:

$$g(S) = \frac{|E[S]|}{c_S}$$

where $c_S = \{(u, v) \in E | u \in S, v \notin S\}$ is the number of edges on the boundary of S.

Definition 4.5. Density (Yang and Leskovec [2015]): The density captures the intuition that communities are well connected. For a set $S \subseteq V$ of nodes, and the induced subgraph $\tilde{G} = (S, E[S])$:

$$g(S) = \frac{2|E[S]|}{|S|(|S| - 1)}$$

Definition 4.6. Conductance (Yang and Leskovec [2015]): The conductance measures the fraction of edges pointing outside the cluster. For a set $S \subseteq V$ of nodes, and the induced subgraph $\tilde{G} = (S, E[S])$:

$$g(S) = \frac{c_S}{2|E[S]| + c_S}$$

Definition 4.7. Modularity (Blondel et al. [2008]): The modularity of a partition measures the density of links inside communities. For a graph G=(V,E)

$$Q = \frac{1}{2|E|} \sum_{i,j} (A_{ij} - \frac{k_i k_j}{2|E|}) \delta(c_i, c_j)$$

where $k_i = \sum_j A_{ij}$

Definition 4.8. Diameter: The diameter of a graph is the greatest distance between two vertices in the graph:

$$d = \max_{(u,v) \in V} d(u, v)$$

The distance for the diameter will be the length of the shortest path.

For the values which require c_S, we will take the average over ground truth clusters. Moreover, we will print in the table for these last criteria the average absolute difference as we did for the clustering coefficient.

4.1.4 Datasets

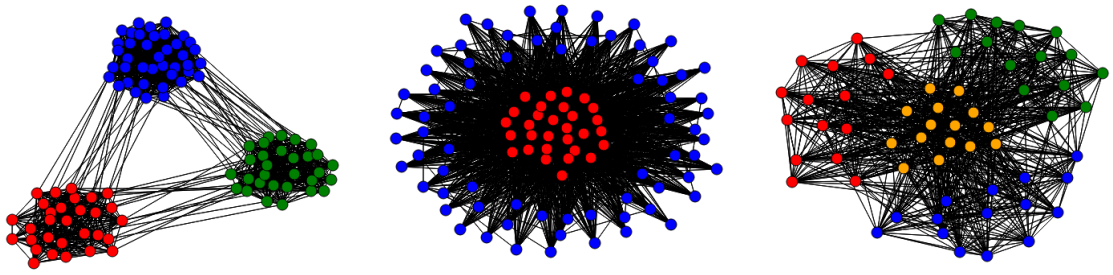
To compare these algorithms, we used real and synthetic datasets. The first idea was that our subsampling method should aim at keeping community structures, as in Maiya and Berger-Wolf [2010], in order to use it to select an initial subgraph for algorithm such

as *online CEM* (Zanghi et al. [2008]). So, we first decided to test the algorithms on synthetic dataset such as the stochastic block model (see definition 4.1). We tried several configurations of stochastic block models. For instance, taking the affiliation model which uses two parameters, p_{in} and p_{out} such that:

$$\begin{cases} \forall i \neq j,\ p_{ij} = p_{out} \\ \forall i,\ p_{ii} = p_{in} \end{cases}$$

With this model, we tried the classic community structure $p_{in} > p_{out}$ as well as an "inverted one" $p_{in} < p_{out}$ where the clusters represent for example influencers on social media. Meaning, there is a group which influences all other nodes. The influencers are not connected between them, but they are connected to all other nodes, which represent people following influencers. We also tried the "hub" model which requires $p_{in} > p_{out}$ and a matrix P of the form:

$$P = \begin{pmatrix} p_{in} & p_{out} & p_{out} & p_{in} \\ p_{out} & p_{in} & p_{out} & p_{in} \\ p_{out} & p_{out} & p_{in} & p_{in} \\ p_{in} & p_{in} & p_{in} & p_{in} \end{pmatrix}$$



(a) $p_{in} > p_{out}$ **(b)** $p_{in} < p_{out}$ **(c)** Hub

Figure 4.9: Example of affiliation models and hub

In order to compare the different algorithms, we used the following parameters:

Name	p_{in}	p_{out}	Number of clusters	Nodes by cluster
SBM1	0.8	0.02	2	100
SBM2	0.5	0.05	5	100
SBM3	0.02	0.8	2	100
SBM4	0.02	0.5	3	100
SBM-Hub	0.5	0.1	4	100

Table 4.1: Parameters of affiliation models

We also used a stochastic block model with a random matrix, forcing $p_{in} > p_{out}$, that we will denote SBM5.

As we were testing the algorithm, we first assessed the results using the number of samples in each cluster which were kept. Then, we used more advanced and quantitative

criteria such as those we described in section 4.1.3.

We decided also to test the algorithm on real datasets in order to compare it with benchmarks algorithms. We tried it on datasets of several sizes. Of course, when there are a few nodes, it is not really useful in practice to get a subgraph, but it still allows us to compare the algorithms on real datas. We used the following real datasets:

dataset	n	k	d_{avg}	d_{max}	CC	density
Blogs	194	1432	14.76	56	0.39	0.0763
Net Science	379	914	4.8	34	0.43	0.0128
C - Elegans Metabolic	453	2032	8.9	238	0.13	0.0199
Epinions	75879	405740	10.69	3044	0.065	0.00014

Table 4.2: Real Graphs with some statistics (n: number of nodes, k: number of edges, d_{avg}: average degree, d_{max}: maximum degree, CC: Clustering Coefficient, density: see 4.5)

The **Blogs** (Chiquet et al. [2007]) dataset is a graph where its nodes correspond to political blogs extracted in 2006 and classified manually as their political party. Two nodes are linked if there exists a link between these two blogs. The clusters correspond to the different French political parties and there are around 11 clusters. However, these clusters have to be taken carefully as they were made arbitrarily to make sense with the political parties. They do not correspond to any ground truth linked to stochastic block models.

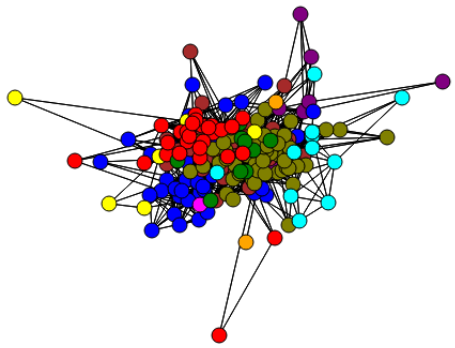


Figure 4.10: Graph blogs (Chiquet et al. [2007])

The **Network Science Collaboration Network** (Rossi and Ahmed [2015]) is a network made in collaboration by researchers. The nodes represent authors, and there are edges between two nodes if they are co-authoring a paper. Here, contrary to the blogs dataset where a kind of manual clustering was possible by labeling the nodes with the political parties, nobody seems to have clustered it manually. So, if we want to compare the conservation of different clusters, we have to cluster the graph ourselves without any ground truth.

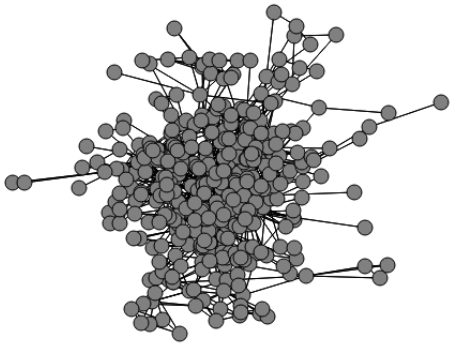


Figure 4.11: Network Science Collaboration Network (Rossi and Ahmed [2015])

The **C. elegans Metabolic Network** (Kunegis [2017]) is a metabolic network. Edges represent interactions between metabolic functions.

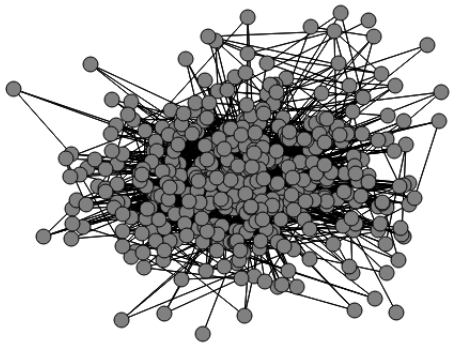


Figure 4.12: C. elegans Metabolic Network (Kunegis [2017])

The **Epinions** dataset is a graph describing who trusts whom based on the general consumer review site Epinions.com in which users can decide whom to trust. This graph is much bigger than the other and we decided to use it in order to see how the method of subsampling scales.

4.1.5 Comparison of Sampling Methods

We will compare some of the algorithms we found in the literature that we described in the previous section 4.1.2. So, we will print the results obtained with the most naive method, the *random node sampling* (RN), the *random walk with random jump* (RWRJ), *Metropolis Hastings* with clustering coefficient distance (MHwC), the *MCMC Expansion Sampler* (XMC), the *Determinantal Point Process* (DPP) graph sampling algorithm based on [Tremblay et al., 2017] and the *Determinantal Point Process* graph sampling with the adjacency kernel (DPPAdj) and the regularized kernel (DPPReg).

We will denote the degree as *d*, the clustering coefficient as *CC*, the modularity as *Mod*, the separability as *Sep* and the conductance as *Cond*. We took averages over 10 or 100 runs following the execution time of the algorithms.

We will show the results here on the datasets SBM2, SBM4 and SBM-Hub. The results on the others datasets can be seen in appendix A1.

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.12	0.17	0.33	0.04	0.05	0.72	2.29
RWRJ	0.12	0.18	0.24	0.16	0.12	0.71	2.69
MHwC	0.15	0.14	0.37	0.08	0.12	0.77	2.66
XMC	0.13	0.17	0.37	0.05	0.06	0.7	2.24
DPP	0.13	0.17	0.33	0.03	0.04	0.73	2.16
DPPAdj	0.21	0.14	0.9	0.14	0.27	0.74	3.9
DPPReg	0.26	0.13	0.6	0.23	0.2	0.71	2.9

Table 4.3: Results on the dataset SBM2

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.13	0.11	0.48	0.04	0.02	0.85	0.77
RWRJ	0.13	0.11	0.41	0.07	0.02	0.84	0.75
MHwC	0.17	0.09	0.5	0.04	0.03	0.88	0.81
XMC	0.14	0.11	0.5	0.03	0.02	0.86	0.78
DPP	0.13	0.11	0.5	0.04	0.02	0.87	0.8
DPPAdj	0.2	0.1	0.5	0.14	0.05	0.95	0.95
DPPReg	0.23	0.09	0.65	0.09	0.08	0.96	0.96

Table 4.4: Results on the dataset SBM4

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.26	0.16	0	0.04	0.87	0.82	2.71
RWRJ	0.3	0.16	0	0.06	1.11	0.84	2.69
MHwC	0.31	0.14	0	0.05	2.15	0.87	2.79
XMC	0.27	0.16	0	0.04	0.83	0.83	2.72
DPP	0.23	0.17	0	0.03	0.45	0.82	2.58
DPPAdj	0.35	0.14	0.3	0.09	1.96	0.94	5.58
DPPReg	0.32	0.14	0.4	0.11	1.53	0.93	5.63

Table 4.5: Results on the dataset SBM-Hub

The *Random Walk* algorithm is supposed to be better than the *Random Node* one according to Leskovec and Faloutsos. We observe that it is not always the case, but it looks better to keep the clustering coefficient.

The *Metropolis Hasting* algorithm is supposed to beat the last two according to Hübler et al.. As per my observations, it seems quite close. However, it seems to estimate clustering coefficients much better, which was expected as the objective density is the distance between clustering coefficients. If we wanted to have better results for the degree, for example, we could use the distance associated to the degree distribution in the density.

According to Maiya and Berger-Wolf, the *MCMC Expansion Sampler* is supposed to keep community structures. So, it should have good results on the density, the modularity, the separability or the clustering coefficient. It is pretty much the case even if it does not look stable.

To the best of our knowledge, the determinantal point processes based algorithms were never compared with the others algorithms on these criteria. And they seem to work quite well, especially the algorithm from Tremblay et al..

Finally, the algorithms are pretty close and it is pretty hard to say that one is better than another. They all have their strengths and their weaknesses, and each one is better at keeping some criteria than others.

4.2 Adaptation of the Algorithm on Graphs

4.2.1 Using graphs settings

As we saw in section 2.2, there are several possibilities to define the different parameters of our algorithm. And it must be adapted to different settings. So, we cannot, *a priori*, use the same parameters as in $\mathbb{R}^d$, and we have to modify the algorithm in order to suit the data structure of graphs.

4.2.1.1 Kernel on Graphs

Firstly, we have several choices for the kernel on the graph. We can choose for example the Laplacian kernel, the normalized Laplacian kernel, or the regularized Laplacian kernel [Avrachenkov et al., 2017b], which can be expensive to compute as we will most likely need to do an eigendecomposition. However, as stated by Smola and Kondor, it is possible to make approximated computation using the fact that the Laplacian L should be sparse in real graphs.

Definition 4.9. Laplacian (Chung and Graham [1997]): Let us denote D the diagonal matrix containing the degrees, and A the adjacency matrix, then the Laplacian is defined as: $L = D - A$.
The Normalized Laplacian is defined as: $\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

Definition 4.10. Laplacian/Heat/Diffusion Kernel (Kondor and Lafferty [2002]): The Laplacian kernel is defined as:

$$K = \exp(-L)$$

Definition 4.11. Normalized Laplacian Kernel (Smola and Kondor [2003]): The Normalized Laplacian Kernel is defined as:

$$K = \exp(-\tilde{L})$$

Definition 4.12. Regularized Laplacian Kernel (Smola and Kondor [2003]): The Regularized Laplacian Kernel is defined as:

$$K = (I + L)^{-1}$$

These kernels are related to the diffusion with the heat equation, but do not really provide intuitive distance on the graphs. However, we can still analyse whether or not they are proximity measures. In Chebotarev and Shamis [2005], they studied functions that express proximity which are not necessarily metrics. To do that, they introduced proximity measures.

Definition 4.13. Proximity Measure (Chebotarev and Shamis [2005]): A proximity measure on a finite set V is a function $K : V \times V \longrightarrow \mathbb{R}$ which satisfies the following triangle inequalities:

$$\begin{cases} \forall x, y, z \in V,\ K(x,y) + K(x,z) - K(y,z) \leq K(x,x) \\ \text{if } z = y \text{ and } y \neq x, \text{ the inequality is strict} \end{cases}$$

Moreover, it is a Σ-proximity if $\forall x \in V,\ \sum_{y \in V} K(x,y) = \Sigma$.

These measures have interesting properties in that they are symmetric, and that they satisfy the egocentrism property, i.e. $\forall x, y \in V,\ x \neq y,\ K(x,x) > K(x,y)$ if there are Σ-proximities. In Avrachenkov et al. [2017a], they studied several kernels and noted for example that the heat kernel (4.10) is not generally a proximity. But it can be one locally, if we take a small t and $K(t) = \exp(-tL)$. They have the same result for the normalized kernel (4.11). However, the regularized Laplacian kernel (4.12) is a good proximity measure which can be interesting to define our random field.

We can also choose the more intuitive kernel based on the adjacency distance:

$$\forall i, j \in V,\ K(i,j) = e^{-\|A_i - A_j\|_2^2}$$

This distance actually counts the number of neighbours which are different between the nodes i and j. K is indeed a kernel on $\mathbb{R}^n$ as it is the Gaussian kernel. This one is much cheaper to compute, especially if we work with sparse structure. The complexity of computing the distance would then be $O(d_{max})$.

4.2.1.2 Convolution on Graph

Then, the natural way of doing the convolution is to sum over the neighbours. The advantage is that it is computationally cheaper than computing the ball with a distance, and it can be implemented very efficiently with the good data structures such as sparse matrix. So the score is computed the following way:

$$\forall i \in V,\ f(i) = \sum_j f_0(j) K(i,j) \mathbb{1}_{\{(i,j) \in E\}} = \sum_j f_0(j) K(i,j) A_{ij}$$

But we can also do the sum over all the nodes in the graph. It would be of course more expensive to compute.

4.2.1.3 Neighborhood on Graphs

Last but not least, we have to choose well the neighborhood in order to sample meaningful points. Indeed, we cannot just take the local maximum over neighbours, because it would return disconnected nodes, which is not suited to graphs. So, the first idea has been to keep a node if its score belongs to the p% best among its neighbours, where p is a threshold that we give as an input.

4.2.1.4 Results

At first, to choose among the possibilities of doing the algorithm, we looked at whether or not the subsample seemed to return sample of each community in stochastic block

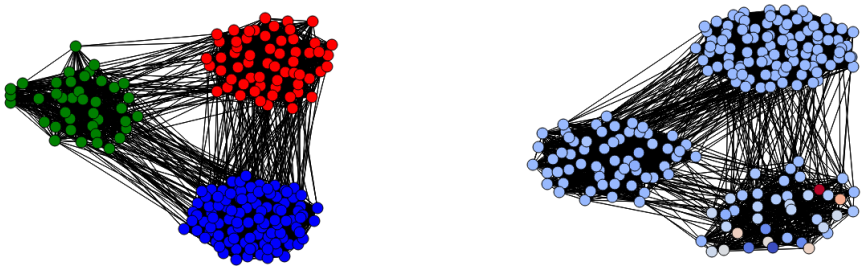
model graphs. It is a naive way of doing, but we were first looking at an algorithm which could return a subsample for clustering algorithms such as in Zanghi et al. [2008]. We noticed that all the configurations seemed to return such samples for the affiliation model with $p_{in} > p_{out}$. But, we also wanted to be able to deal with the case $p_{in} < p_{out}$. And in this case, most of them did not return good samples with this criterion. We finally noticed that it acts as expected only when we made the sum over all the nodes.

4.2.2 Using $\mathbb{R}^d$

As we previously saw, it seems to maintain the community structure better when we perform the convolution on all the nodes.

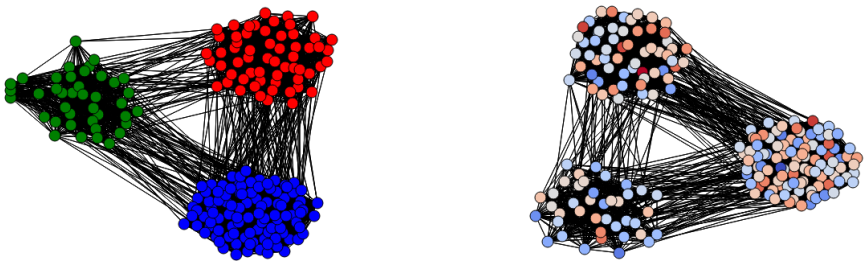
However, we noticed that with the convolution on the neighbours, especially in the stochastic block model where $p_{in} < p_{out}$, the values of the scores of each cluster seemed to be different in average. In other words, it seems to separate the scores of the clusters.

Originally, the algorithm was supposed to create a Gaussian random field, meaning that close nodes should have close values. And it seems to work, but not when we take the neighborhood using the neighbours in the graph. Indeed, each cluster seems to take a value, in average, which means that the clusters with the highest values should predominate.



(a) 200 nodes **(b)** Scores f

Figure 4.13: Scores on each node when taking the convolution over neighbours



(a) 200 nodes **(b)** Scores f

Figure 4.14: Scores on each node when taking the convolution over all nodes

To counter that, we thought of using a neighborhood which is consistent with the way we created the random field, meaning that we should use the same distance used in the kernel. For example, if we used the kernel with adjacency distance:

$$\forall i, j \in V,\ K(i,j) = e^{-\|A_i - A_j\|_2^2}$$

We should then use as neighborhood the ball of a certain radius using this distance. It is almost the same algorithm as (3). This intuition was actually good and the algorithm seems to return nodes from all communities, even when $p_{in} < p_{out}$. On the other hand, it is more computationally expensive.

4.2.3 Comparison of Algorithms

To sum up, the algorithms are:

Algorithm 14: Convolution over neighbours (GRF)

Result: Subsample S
Initialization: A graph $G = (V, E)$, a kernel K, a proportion p
- $\forall i \in V,\ f_0(i) \sim \mathcal{N}(0, 1)$
- $\forall i \in V,\ f(i) = \sum_{j \in V} f_0(j) K(i, j) \mathbb{1}_{\{(i,j) \in E\}}$
- $S = \emptyset$
foreach $i \in V$ **do**
  Denote $X_1, ..., X_{d_j}$ the sorted $(f(j))_{j \in N(i)}$
  if $f(i) \geq X_{\lfloor \frac{j}{p} \rfloor}$ **then**
    $S = S \cup \{i\}$
end

Algorithm 15: Convolution over all nodes (GRFAll)

Result: Subsample S
Initialization: A graph $G = (V, E)$, a kernel K, a proportion p
- $\forall i \in V,\ f_0(i) \sim \mathcal{N}(0, 1)$
- $\forall i \in V,\ f(i) = \sum_{j \in V} f_0(j) K(i, j)$
- $S = \emptyset$
foreach $i \in V$ **do**
  Denote $X_1, ..., X_{d_j}$ the sorted $(f(j))_{j \in N(i)}$
  if $f(i) \geq X_{\lfloor \frac{j}{p} \rfloor}$ **then**
    $S = S \cup \{i\}$
end

Algorithm 16: Neighborhood in $\mathbb{R}^n$ (GRFR)

Result: Subsample S
Initialization: A graph $G = (V, E)$, a kernel K, a proportion p, a radius b
- $\forall i \in V,\ f_0(i) \sim \mathcal{N}(0, 1)$
- $\forall i \in V,\ f(i) = \sum_{j \in V} f_0(j) K(i, j) \mathbb{1}_{\{(i,j) \in E\}}$
- $S = \emptyset$
foreach $i \in V$ **do**
  if $\forall j \in B(i, b),\ f(j) < f(i)$ **then**
    $S = S \cup \{i\}$
end

4.2.3.1 Complexity

The fastest algorithm we proposed is algorithm 14 as it only needs to compute the kernel between neighbours. Using sparse structure for matrix, this can be computed for all nodes in $O(nd_{max}^2)$ with the adjacency distance, d_{max} being the maximum degree of nodes. The convolutions can then be computed in $O(nd_{max})$ as we only take care of the neighbours. Finally, we need to sort the values of the convolutions, which can be made in $O(d_{max}\log(d_{max}))$. So overall, the complexity is in $O(nd_{max}^2)$. For the two other algorithms, we need to run through all the nodes a second time, in order to compute the kernel between all points for algorithm 15, or to filter the points belonging to a ball in algorithm 16. This raises a complexity of $O(n^2 d_{max})$.

Compared to the other existing methods that we reviewed previously in section 4.1.2, the complexity is worse than the *Random Node* or *Random Edge* sampling as these algorithms are very fast but also pretty naive. We expect to obtain better results. On the other hand, the complexity of the *Metropolis Hastings* based algorithm is according to Hübler et al. $O(nd_{avg} + R(\sigma(G)) + \#it(d_{avg}n' + R(\sigma(S_{current}))))$ with n' the size of the subsample and $R(\sigma(G))$ is the complexity of computing the distance on the graph. The distance that we used in our test was the clustering coefficient one, and it can be computed in the worst case in $O(n^3)$ which gives an overall complexity of $O(nd_{avg} + n^3 + \#it(d_{avg}n' + n'^3))$. For the *MCMC expansion sampler*, the complexity to compute the distance is cheaper as we only need to compute the number of neighbours of the set. It can be done naively in $O(n'd)$ if for each point of the subsample, we test for all the neighbours to see whether or not they belong to the subsample (using a hash table with set in python for example). The overall complexity is then $O(nd_{avg} + \#it\ d_{avg}n')$ which is slightly better than ours.

4.2.3.2 Results on datasets

We used the same protocol as in section (4.1.5) on the three previously presented algorithms. We added the results of algorithm (12), as it is supposed to be one of the best subsampler, in order to compare them.

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
GRF	0.16	0.16	1.04	0.25	0.24	0.38	0.75
GRFAll	0.15	0.15	0.63	0.16	0.05	0.74	2.43
GRFR	0.14	0.17	0.56	0.15	0.25	0.48	0.84
MHwC	0.15	0.14	0.37	0.08	0.12	0.77	2.66

Table 4.6: Results of our Algorithms on the dataset SBM2

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
GRF	0.15	0.11	0.5	0.02	0.02	0.51	0.43
GRFAll	0.16	0.1	0.52	0.09	0.05	0.91	0.87
GRFR	0.13	0.12	0.2	0.08	0.08	0.58	0.42
MHwC	0.17	0.09	0.5	0.04	0.03	0.88	0.81

Table 4.7: Results of our Algorithms on the dataset SBM4

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
GRF	0.32	0.16	0.5	0.38	8	0.44	0.08
GRFAll	0.28	0.15	0.04	0.06	1.35	0.87	3.39
GRFR	0.17	0.16	0	0.04	0.87	0.82	2.71
MHwC	0.31	0.14	0	0.05	2.15	0.87	2.79

Table 4.8: Results of our Algorithms on the dataset SBM-Hub

Comparing our algorithms with the ones presented before, they seem to have about the same kind of results. They are not really stable, and are sometimes better, sometimes worse than these algorithms. However, the GRFR seems to work better than the two others overall.

Then, we also tried it on the epinion dataset (4.2) which is a huge one compared to the previous datasets.

On this dataset, we encountered some difficulties linked to the size and to the configuration of the graph itself. Indeed, as this graph is really huge, we have to adapt the different algorithms taking advantage of a sparse representation of the adjacency matrix, in order to subsample on it, and even to compute the statistics. Moreover, this graph is not very dense, and therefore, even when keeping points only when they have the maximum values over their neighbours, it keep too many points. The smaller sample we succeeded to draw with our algorithm (14) had 26,847 nodes over the 75,879 initially, which is huge and useless as there are lots of disconnected nodes. This is of course a huge drawback of the algorithm. To prevent that from happening, we can remove all the points which do not have any neighbours in the subsample. It allows us to reduce the subsample size.

With a proportion $p = 0.2$ as input, we obtained around 30,000 points in the first sample. After removing all the disconnected nodes with 0 neighbours, we had around 7,000 points. And with this subsample, we obtained the following results:

Algorithms	Degree	Clustering Coefficient	Density
GRF	0.65	0.014	17.1
RN	0.69	0.012	1,302.39
XMC	0.7	0.014	1,327.72

Table 4.9: Results of our Algorithms on the dataset Epinion

We only tried algorithm 14 because it is the only one that we proposed which scales well with big datasets, as we can take advantage of the sparse structure of the graph in its implementation. Therefore, we compared it with the random node sampling algorithm (4.1.2.1) which is of course still efficient on huge graphs, and the Metropolis expansion sampler (4.1.2.5). On table 4.9, we can see that our algorithm is comparable to the two others in terms of degree and clustering coefficient, even slightly better. But on the density, we see that our method returns the best result.

4.2.3.3 Conclusion

We saw that the algorithm is pretty interesting as it can scale well on big datasets and have results which are comparable to other existing techniques. Moreover, if the graphs are dense enough, the algorithm should return a connected set of nodes. However, it remains hard to say if the computation time is worth it, compared to simple methods such as random node sampling. Furthermore, there are drawbacks such as the difficulty to plan the number of nodes which will be returned, like the version of the algorithm in section 3.2.

4.3 Application on Online Type of Expectation Maximization Algorithms

Without a clear consensus, this is not easy to define what a good community structure is. Thus, lots of techniques were proposed in order to find these types of structures. For example, popular methods are spectral clustering using algorithms such as K-Means. Another efficient method is the Louvain algorithm proposed in Blondel et al. [2008] which aims at maximizing the modularity (4.7) and which is pretty efficient with a complexity of $O(n\log(n))$. On the other hand, these methods work terribly as soon as there are complicated clusters such as star-shaped for example. To find these types of clusters, mathematical models exist which aim at describing graphs and at finding clusters in it. These techniques often rely on EM type of algorithms, and are heavier to run, and thus harder to use on real graphs, but they can also find more intricate communities.

4.3.1 Variational EM

A way of modeling graphs in order to find clusters is to use random graphs such as stochastic block models (4.1). This model is called "mixture model for graphs" and supposes that vertices belong to one of Q classes with probability α_q $\forall q \in \{1, ..., Q\}$. Let us denote $Z_{iq} \sim Ber(\alpha_q)$ the indicator variable which indicates whether the node i belongs to the class q or not. Then, we can denote π_{ql} the probability that a vertex from the class q is linked with a vertex from class l. If we note X_{ij} the indicator variables of the edges, then we also suppose that there are conditionally independent given the classes of the nodes i and j, and we have in the case of undirected graph:

$$\begin{cases} X_{ij}|\{Z_{iq}Z_{jl} = 1\} \sim Ber(\pi_{ql}) & \text{if } i \neq j \\ X_{ii} = 0 \end{cases}$$

Then, in order to apply the EM algorithm (5), we need to compute $\log p(X, Z|\theta)$ where $\theta = ((\alpha_q)_{1\leq q\leq Q}, (\pi_{ql})_{1\leq q,l\leq Q})$.

$$\log p(X, Z|\theta) = \sum_i \sum_q \log(\alpha_q) Z_{iq} + \frac{1}{2} \sum_{i\neq j} \sum_{q,l} Z_{iq}Z_{jl} \log(\pi_{ql}^{X_{ij}}(1 - \pi_{ql})^{1-X_{ij}})$$

Unfortunately, we can not apply directly the EM algorithm as $p(Z|X)$ is not tractable. Therefore, Daudin et al. proposed a variational approach that aims at optimizing a lower

bound of $p(X)$, which is:

$$J(R_X) = \log p(X) - KL(R_X||p(\cdot|X))$$

It is indeed a lower bound as the Kullback-Leibler divergence is non negative, and is equal to 0 if and only if $R_X = p(\cdot|X)$. To approximate $p(\cdot|X)$, we also have to choose a class of distribution. In Daudin et al. [2008], they simply choose the mean field approximation among multinomial of parameters $\tau_i = (\tau_{iq})_{1\leq q\leq Q}$:

$$R_X(Z) = \prod_i \prod_q \tau_{iq}^{z_{iq}}$$

Then the algorithm consists of maximizing alternatively with respect to $(\tau_i)_i$ and with respect to α and π. The formula are derived in Daudin et al. [2008] and are:

$$\begin{cases} \tau_{iq} \propto \alpha_q \prod_{j\neq i} \prod_l \log\left(\pi_{ql}^{X_{ij}}(1-\pi_{ql})^{1-X_{in}}\right)^{\tau_{jl}} \\ \alpha_q = \frac{1}{n}\sum_i \tau_{iq} \\ \pi_{ql} = \frac{\sum_{i\neq j}\tau_{iq}\tau_{jl}X_{ij}}{\sum_{i\neq j}\tau_{iq}\tau_{jl}} \end{cases}$$

And the algorithm is:

Algorithm 17: Variational Expectation-Maximization Algorithm

while $|\theta^{(t+1)} - \theta^{(t)}| \leq \epsilon$ **do**

- *M-step:* Compute $\forall q, l \in \{1, ..., Q\}$,

$$\begin{cases} \alpha_q = \frac{1}{n}\sum_i \tau_{iq} \\ \pi_{ql} = \frac{\sum_{i\neq j}\tau_{iq}\tau_{jl}X_{ij}}{\sum_{i\neq j}\tau_{iq}\tau_{jl}} \end{cases}$$

- *V-step:* Compute $\forall i \in \{1, ..., n\}, \forall q \in \{1, ..., Q\}$,

$$\tau_{iq} \propto \alpha_q \prod_{j\neq i}\prod_l \log\left(\pi_{ql}^{X_{ij}}(1-\pi_{ql})^{1-X_{in}}\right)^{\tau_{jl}}$$

end

This algorithm works pretty well to estimate stochastic block models, but the complexity is in $O(n^2)$ which can be too big for huge graphs.

4.3.2 Online Classification EM

Therefore, Zanghi et al. proposed an online algorithm which aims at estimating the MixNet model on a graph by maximizing the complete likelihood online with a stochastic approximation.

This algorithm is divided in two steps and relies on the following decomposition of the

likelihood:

$$L_C^m(X^m, Z^m(\theta^{m-1}), \theta) = L_C^{m-1}(X^{m-1}, Z^{m-1}, \theta) + \sum_q z_{mq}\Big(\log(\alpha_q) + \sum_l \sum_{j \neq m} z_{jl} \log(\pi_{ql}^{X_{mj}}(1-\pi_{ql})^{(1-X_{mj})})\Big)$$

with X^m the adjacency matrix of the graph containing m nodes and:

$$Z^{m-1}(\theta) = \underset{z}{\operatorname{argmax}}\ L_C(X^{m-1}, Z, \theta)$$

The first step then consists in maximizing the second term with respect to z_m, and the second step consists in maximizing with respect to θ. As for the regular online classification EM, we can initialize the algorithm on a subset and deal with the rest of the points online. Moreover, it is possible to revisit each node in order to continue improve the likelihood.

Algorithm 18: Online Classification Expectation-Maximization Algorithm

Initialization: n_0: number of initialization vertices
Sample n_0 nodes with the algorithm of your choice in S_0
Initialize z, α, π
$k = n_0$
foreach $m \notin S_0$ **do**
- $k = k + 1$
- *C-step:* $q^* = \underset{1 \leq k \leq K}{\operatorname{argmax}}\ \log(\alpha_q) + \sum_l \sum_{j \neq m} z_{jl} \log(\pi_{ql}^{X_{mj}}(1-\pi_{ql})^{(1-X_{mj})})$
- $z_{mq} = \begin{cases} 1 & \text{if } q = q^* \\ 0 & \text{otherwise} \end{cases}$
- *M-step:* Compute $\forall q, l \in \{1, ..., Q\}$,

$$\begin{cases} n_q^{(k)} = n_q^{(k-1)} + z_{mq} \\ n_{ql}^{(k)} = n_{ql}^{(k-1)} + \sum_{j \neq m} z_{mq} z_{jl} X_{mj} \\ \alpha_q^{(k)} = \frac{n_q^{(k)}}{k} \\ \pi_{ql}^{(k)} = \frac{n_q^{(k)}}{n_q^{(k)} n_l^{(k)}} \\ \pi_{qq}^{(k)} = \frac{2n_{qq}^{(k)}}{n_q^{(k)}(n_q^{(k)}-1)} \end{cases}$$

end

This algorithm is also in $O(n^2)$ but Zanghi et al. mentioned that the speed ratio between this algorithm and the variational EM is a linear function of the number of nodes (approximately $1.5n$).

4.3.3 Comparison

To see if our subsampling as initialization of OCEM algorithm improves over a simple uniform sampling, we created as in section 3.3.4 several scenarios. And for each one of

them, we generated 50 datasets and plotted box plots of the adjusted Rand index as well as the execution time. For the different scenarios, we used stochastic block models with the same parameters as in table 4.1.

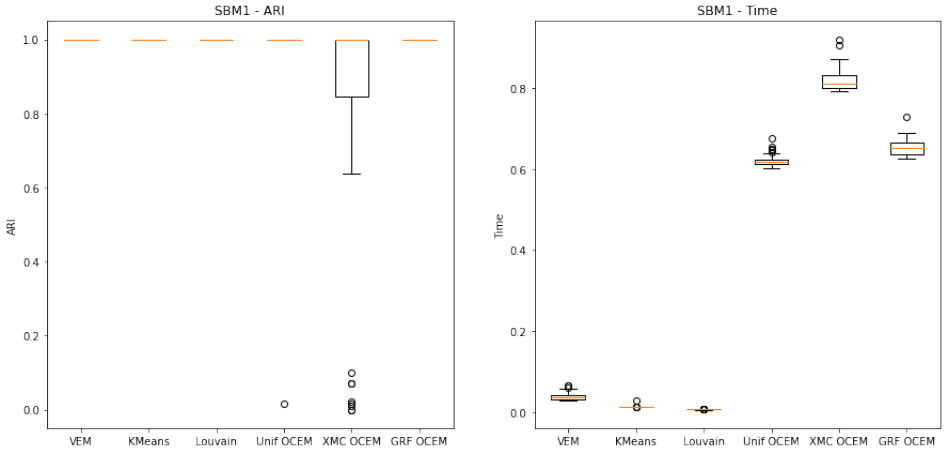


Figure 4.15: Results SBM1

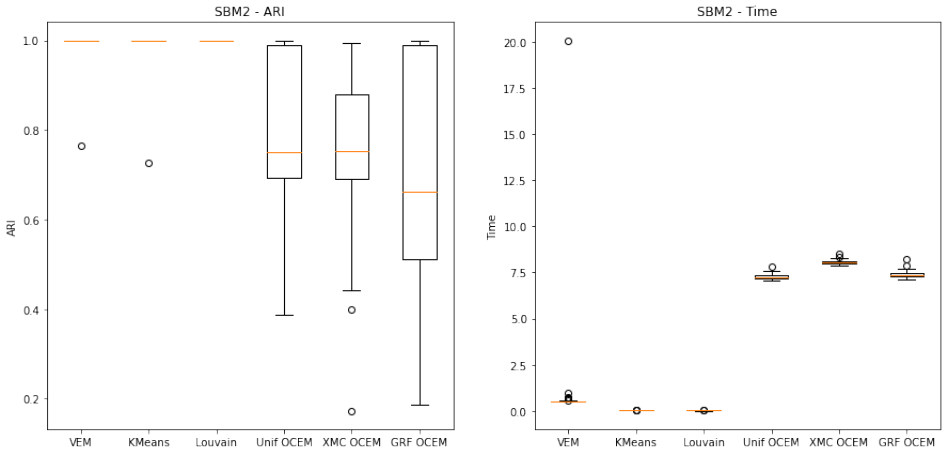


Figure 4.16: Results SBM2

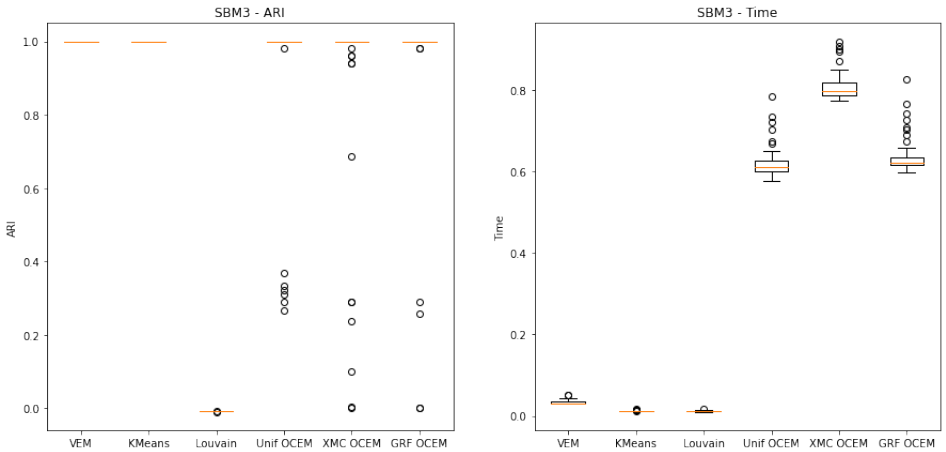


Figure 4.17: Results SBM3

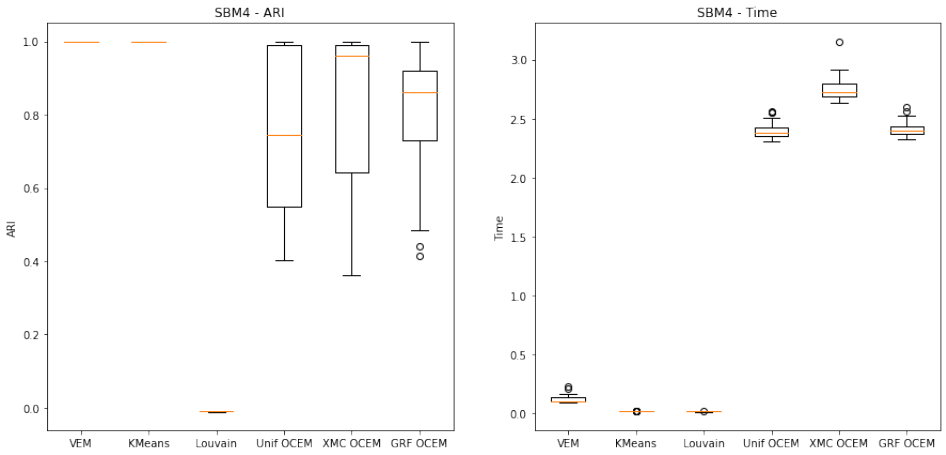


Figure 4.18: Results SBM4

We first observe that the execution time of the OCEM is much bigger than the VEM whereas it should be the contrary, as we saw previously. This is probably due to the implementations which were made in python using Numba (Lam et al. [2015]) in order to optimize and to parallelize whenever it was possible. Unfortunately, the OCEM algorithm relies much more on loops which are pretty slow in python. Perhaps an implementation based on Cython would have been better.

On the results side, we see on figure 4.15 and 4.16 that on classical community structures with $p_{in} > p_{out}$, the usual algorithms Louvain, KMeans and the VEM work perfectly which is not the case of the OCEM. Moreover, the OCEM with our subsampling initialization has an ARI which is inconveniently unstable, and the median is below the two others initialisations.

For the more complicated clustering, with $p_{in} < p_{out}$, we see on figures 4.17 and 4.18 that the Louvain algorithm does not perform sufficiently well. It is the illustration of what we previously mentioned: the Louvain algorithm works well only on classical community structures. This is expected as it relies on maximizing the modularity (4.7) which measures the density of links inside communities. Of course in this case, maximizing the modularity cannot give the right clustering as the density of links inside the clusters is small. On the OCEM algorithm, we observe comparable results. The XMC initialization seems to work better, and our initialization seems to be more stable instead, and better than the uniform initialization.

4.4 Clustering on Graphs

4.4.1 Idea

As we noticed that the scores seemed to be different in average for the different clusters, we started to investigate this lead in order to cluster using the values of f. Firstly, if we change the initial distribution f_0 as $\forall i,\ f_0(i) \sim \mathcal{N}(1, \frac{1}{2})$ for example, we can compute the law of f:

$$\forall i \in V,\ f(i) = \sum_{j \in V} f_0(j) K(i,j) A_{ij}$$

So, as the $(f_0(j))_j$ are independent, we have:

$$\forall i \in V,\ f(i) \sim \mathcal{N}(\sum_{j \in V} K(i,j)A_{ij}, \frac{1}{2}\sum_{j \in V} K(i,j)^2 A_{ij})$$

If we take $K(i,j) = 1\ \forall i,j$, then:

$$\forall i \in V,\ f(i) \sim \mathcal{N}(d_i, \frac{d_i}{2})$$

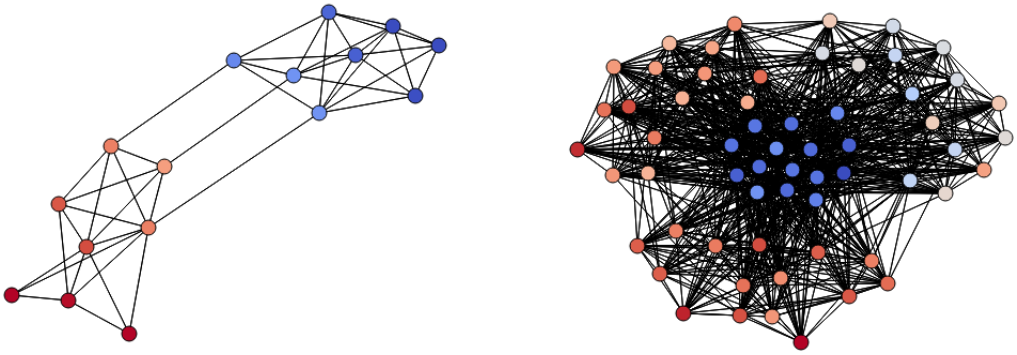
So, we see that if the clusters have in average different degrees, they should have values of f centered around their degrees. But, whenever the clusters have the same degrees, the scores of the points of different clusters should overlap.

Then, we tried to normalize the sum, and to iterate it:

$$\forall k \geq 0, \forall i \in V,\ f_{k+1}(i) = \frac{1}{d_i}\sum_{j \in V} f_k(j)A_{ij}$$

Doing that, we have $\forall i \in V,\ f_1(i) \sim \mathcal{N}(1, \frac{1}{2d_i})$.

For a stochastic block model with 2 clusters of the same size and with $p_{in} > p_{out}$, and for a hub with 4 clusters, we obtained the following results:



(a) Scores obtained on a SBM with $p_{in} > p_{out}$ and 2 equal clusters

(b) Scores obtained on a hub

Figure 4.19: Scores on each node

We observe on the figure 4.19a that the values of f in each cluster seem to be of the same magnitude, but are different between clusters. And for the hub (4.19b), we observe pretty much the same thing. The cluster of the middle contains values very different from the others. And among the three clusters surrounding the middle one, we see two types of clusters: two which have similar values, and the last one having different values than the rest. There seems to be an overlap of the scores of two clusters in this case which can be problematic if we want to apply a clustering algorithm directly on these scores.

Then we plotted these scores over the different iterations for the SBM with $p_{in} > p_{out}$:

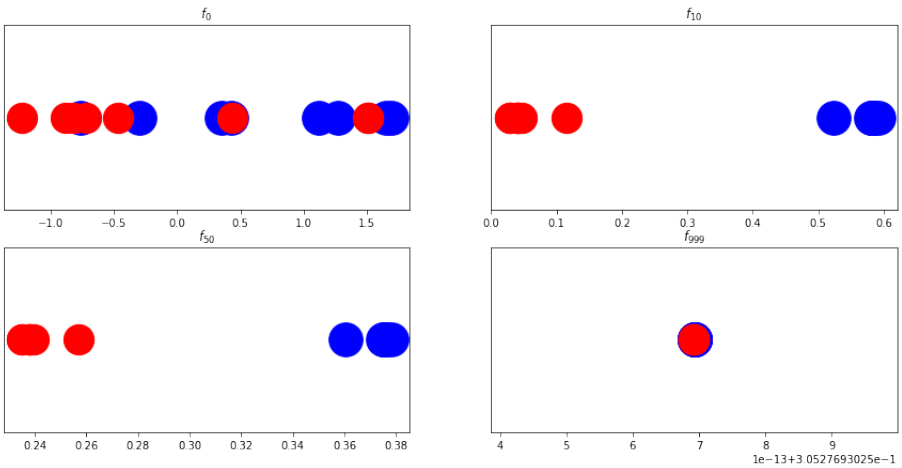


Figure 4.20: Scores obtained over iterations on a SBM with 2 equal clusters

We observe on the figure 4.20 that the values of the clusters seem to go apart, and then converge towards the same value. We can actually understand the convergence phenomena by rewriting the score with matrices. Indeed, computing the average for each node is equivalent to compute:

$$\begin{aligned}\forall k \geq 0,\ X^{(k+1)} &= (D^{-1}A)X^{(k)} = (D^{-1}A)^{k+1}X^{(0)} \\ &\sim \mathcal{N}((D^{-1}A)^{k+1}\mu, (D^{-1}A)^{k+1}\Sigma((D^{-1}A)^{k+1})^T)\end{aligned}$$

with D being a diagonal matrix containing the degrees of the nodes, A being the adjacency matrix, and $X^{(0)} \sim \mathcal{N}(\mu, \Sigma)$, with for example $\mu = \gamma 1_n$ and $\Sigma = \sigma^2 I_n$. Then we have $X_i^{(k+1)} = f_{k+1}(i)$ and the matrix $P = D^{-1}A$ is a stochastic matrix by row. So, if μ is proportional to 1_n, then $(D^{-1}A)\mu = \mu$. And P is especially a random walk matrix, which is a transition matrix of a Markov chain. Therefore, we know that:

$$\lim_{k\to\infty}(D^{-1}A)^k = \begin{pmatrix}\pi \\ \vdots \\ \pi\end{pmatrix}$$

where $\forall i \in \{1, \dots, n\},\ \pi_i = \frac{d_i}{\sum_j d_j}$ and π is the stationary distribution of the associated Markov Chain.
Thus, we have:

$$\lim_{k\to\infty} X^{(k)} = \begin{pmatrix}\pi \\ \vdots \\ \pi\end{pmatrix} X^{(0)}$$

So, we understand why the values tend to squeeze. And we can assume that the separation between clusters is due to a second order term. We know that the first eigenvalue of P is 1_n and that $1_n = P1_n$ as P is a stochastic matrix, and that $\pi = \pi P$ as π is the stationary distribution. Therefore, we have the following decomposition:

$$\forall k \geq 0,\ P^k = 1\pi + \sum_{i=1}^{n} \lambda_i^k v_i u_i^T$$

with $(v_i)_{2\leq i\leq n}$ the right eigenvectors, $(u_i)_{2\leq i\leq n}$ the left eigenvectors and $(\lambda_i)_{1\leq i\leq n}$ the eigenvalues such that $1 = \lambda_1 \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. And we have that the rate of convergence is in $O(|\lambda_2|^n)$ if P is also irreducible, aperiodic, and has eigenvalues with different modulus by Perron Frobenius theorem (see for example Brémaud [2013] or Backåker [2012]).

By experimenting, in plotting the values given by $\lambda_2^n v_2 u_2^T X^{(0)}$, we observed that this term seems to separate the clusters:

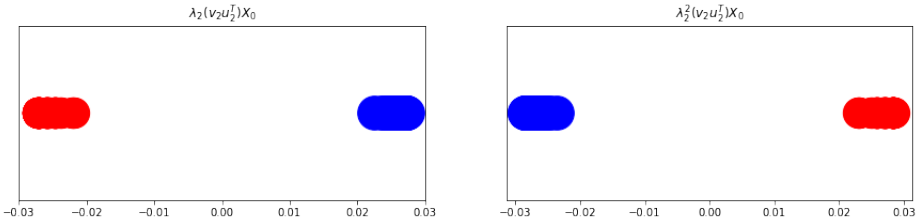


Figure 4.21: Scores obtained with only the second term

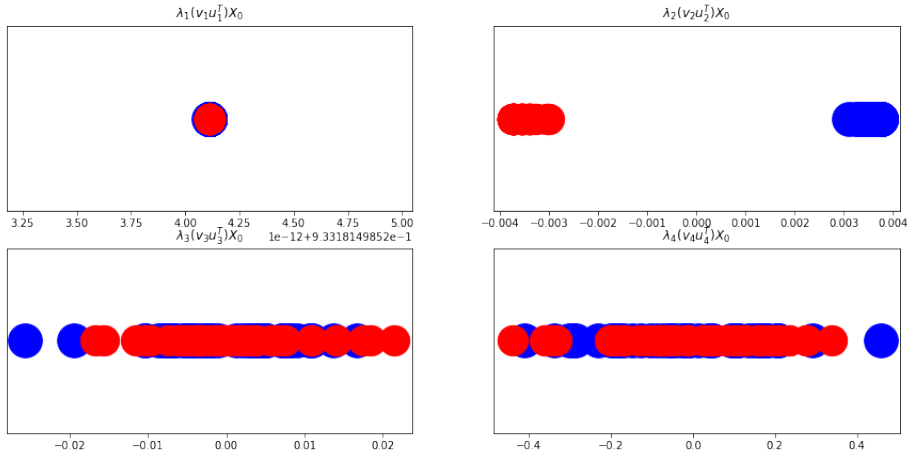


Figure 4.22: Scores with different terms

Then, we plotted the scores over iterations of the hub:

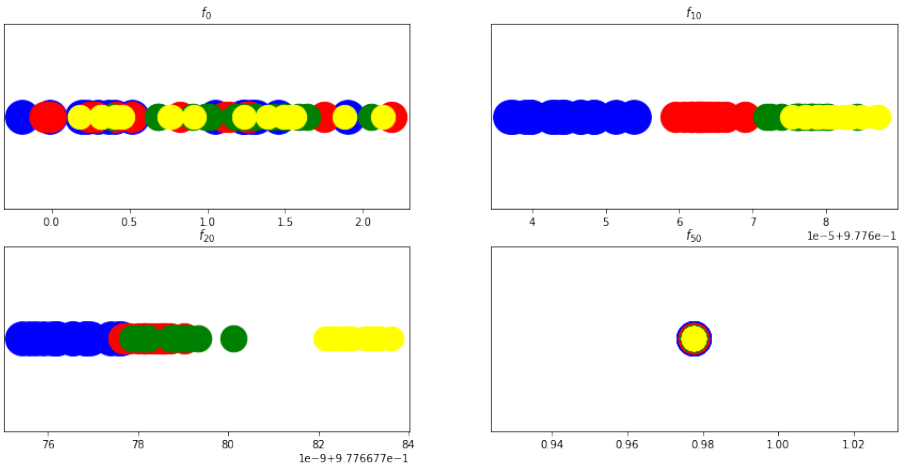


Figure 4.23: Scores obtained over iterations on a Hub

We observe on the figure 4.23 that unfortunately, the scores of the different clusters seem to overlap each other. So, the clustering on the scores becomes impossible. And it is actually the case even for affiliation models whenever there are more than 3 clusters. To prevent that, we tried to initialize the algorithm in higher dimension. Meaning for example instead of drawing $f_0(i) \sim \mathcal{N}(1, \frac{1}{2})$, we can draw a Gaussian vector in dimension 2 or beyond, when there are more clusters: $f_0(i) \sim \mathcal{N}(1_2, \frac{1}{2}I_2)$. Then, the scores will be in greater dimension and we hope that the clusters will be as separated as in 1D because they have more degrees of freedom. We obtain these types of results:

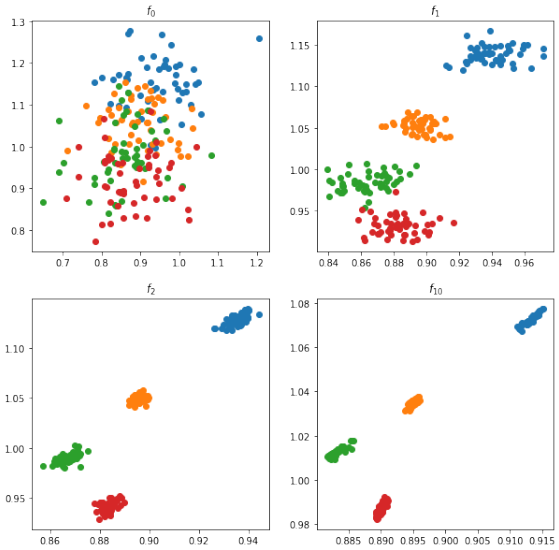


Figure 4.24: Scores in 2D over several iterations

It seems to work beautifully. Of course, we observe the same convergence behavior. Furthermore, we observe that there are more significant eigenvalues and eigenvectors which matters for the clustering. More precisely, there seems to be k important terms for k clusters:

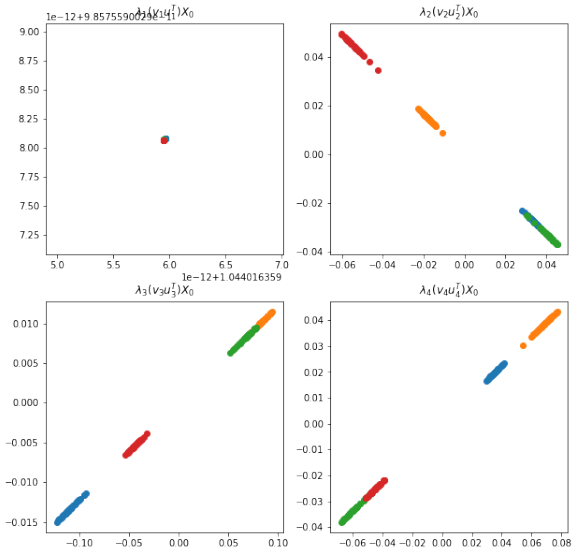


Figure 4.25: Scores in 2D with different terms

Interestingly enough, we see that the term of the second eigenvalue separates the clusters less than the third and the fourth.

To understand the behaviour of the separation of the scores, we noticed the similarity with the spectral clustering (see for example Von Luxburg [2007]). Indeed, one of the way of exercising spectral clustering is to use as embedding the random walk Laplacian which is:

$$L_{rw} = D^{-1}L = I - D^{-1}A = I - P$$

We see here that our transition matrix appears in the Laplacian. Moreover, if we note $(\mu_i)_i$ the eigenvalues of L_{rw}, then we have that: $\forall i,\ \mu_i = 1 - \lambda_i$.

The spectral clustering method consists of computing this Laplacian matrix, and then to pick the first k eigenvectors, before clustering the points using an algorithm such as K-Means on the rows. Indeed, the idea is that in the ideal case when the k clusters are not linked, the eigenvalue 0 is of multiplicity k and the corresponding eigenvectors are indicator vectors with 1 on components which belong to the corresponding cluster. Therefore, keeping only these k eigenvectors as columns, we see that the rows will contain only a 1 in the column of the cluster which contains it. When we are not in the ideal case, we hope to have vectors close to these indicator vectors. This part explains why the k largest eigenvectors of P seem to separate the values of the different clusters.

However, the spectral clustering works only on communities, i.e. when $p_{in} > p_{out}$, and not in the case of $p_{in} < p_{out}$. With our method, we surprisingly observe that it seems also to work on the inverted case. Indeed, we obtain these types of results:

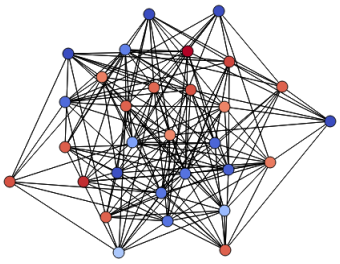


Figure 4.26: Scores on an "inverted" SBM

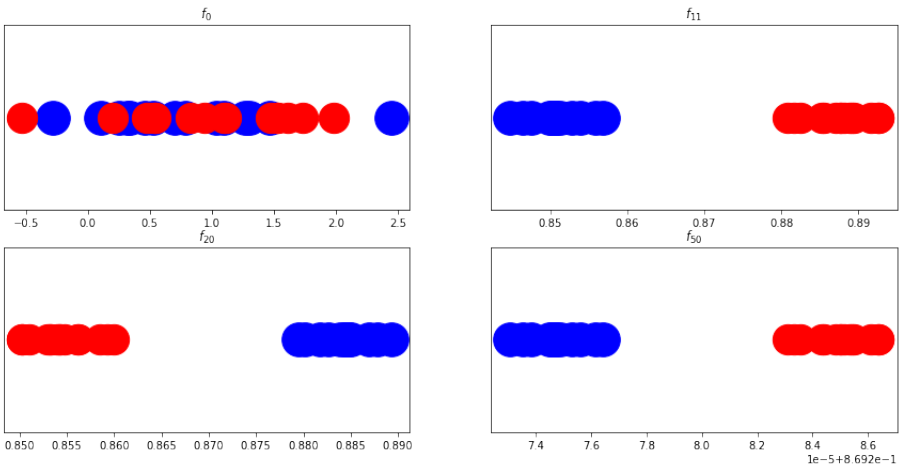


Figure 4.27: Scores over iterations on an "inverted" SBM

To have an intuitive explanation of why it works, we can get back to the first expression (4.4.1). As we take the mean of the previous scores overs the neighbours, we see that if a group of nodes are connected to the same nodes, then they will tend to take the same value. In the case of communities, the nodes are very densely connected by cluster. So, the means of the different clusters will take a certain value. Whenever the different communities have the same number of nodes and the same degrees in averages, then the score will be probably different thanks to the random initialization. But it may not be the case. In the case of "inverted" SBM, the means will be over the points from the other clusters. But there will be the same phenomenon of average. On the other hand, the scores of the clusters will swap between each others, as the average of a node will be taken mainly over the nodes of the other cluster. This is what we observe in figure 4.27.

4.4.2 Comparison

One of the main interests of this algorithm is that the iterations are composed only of averages, which are really easy and fast to compute with a sparse structure. Each iteration can indeed be done in $O(nd_{max})$ as it only requires the neighbours of each point. Furthermore, by computing the variance of the score at each step, we see that this one seems to strictly decrease. We could therefore use it as a criterion to stop the iterations. For example, if the variance diminishes below a threshold, then we can stop the algorithm. This can be pretty fast.

We compared this algorithm with spectral clustering using the random walk Laplacian and K-Means, Louvain and the variational EM algorithm on the same SBM scenarios of section 4.3.3. We added a scenario SBM2.1 with different proportions for the clusters: the corresponding clusters contain in this case respectively 100, 75, 25, 150 and 50 nodes. We also added a scenario SBM4.1 with the proportions: 75, 200, 25 nodes. And finally, we added a second hub scenario with the proportions 100, 150, 50, 50.

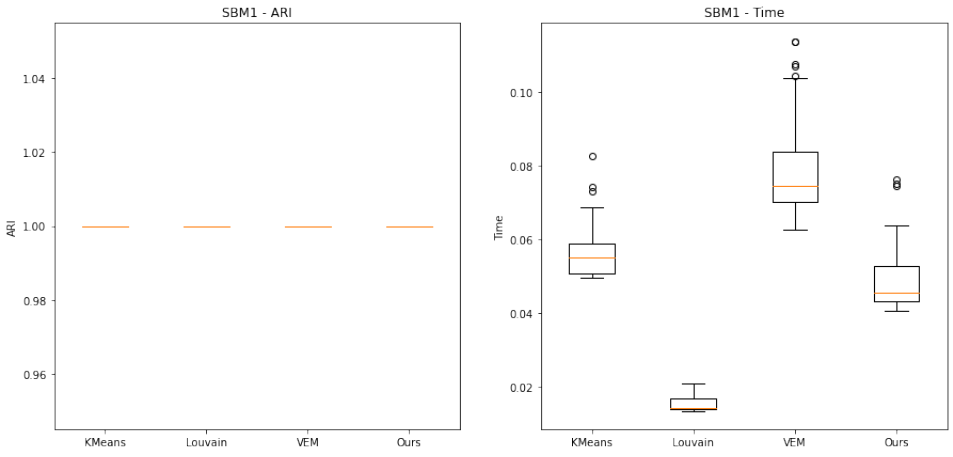


Figure 4.28: Results SBM1

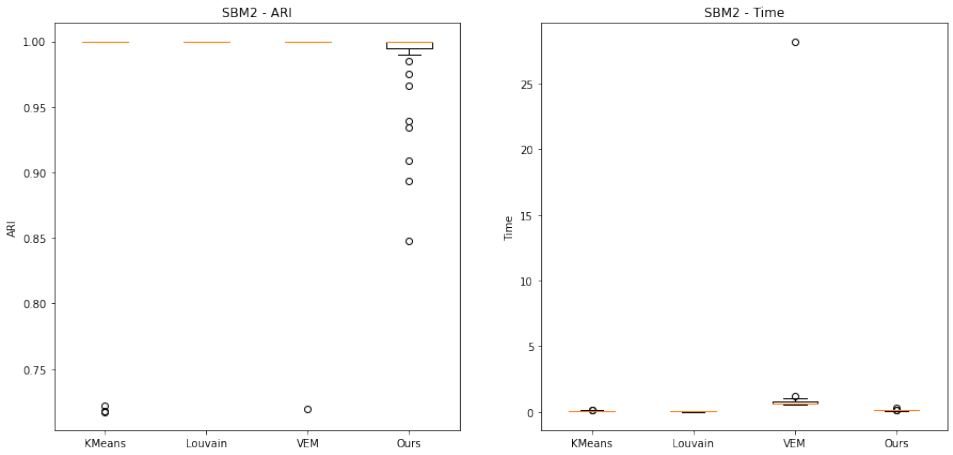


Figure 4.29: Results SBM2

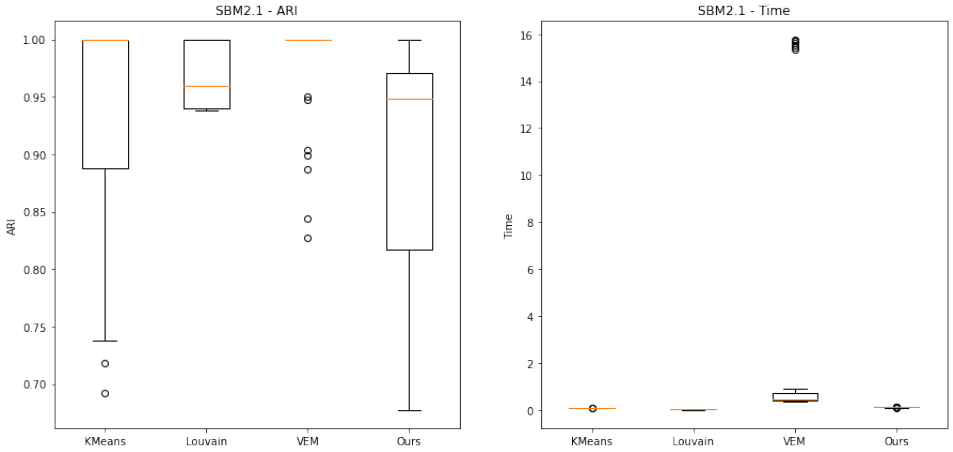


Figure 4.30: Results SBM2.1

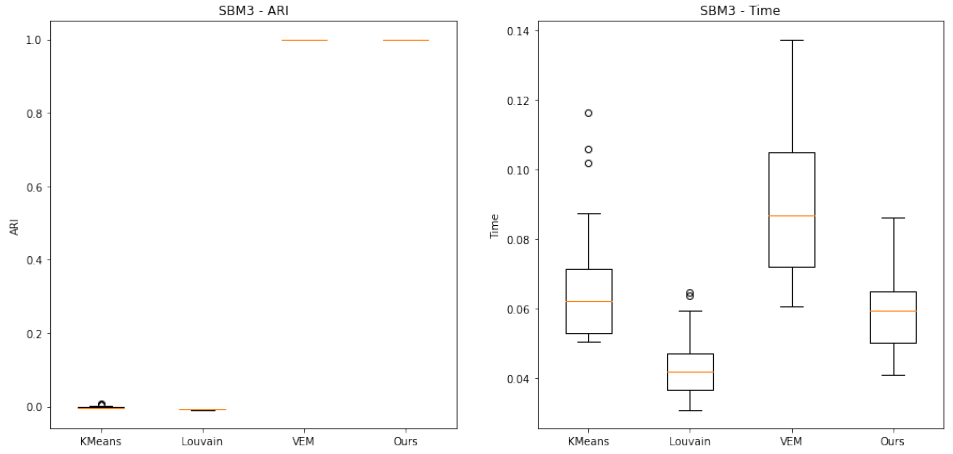


Figure 4.31: Results SBM3

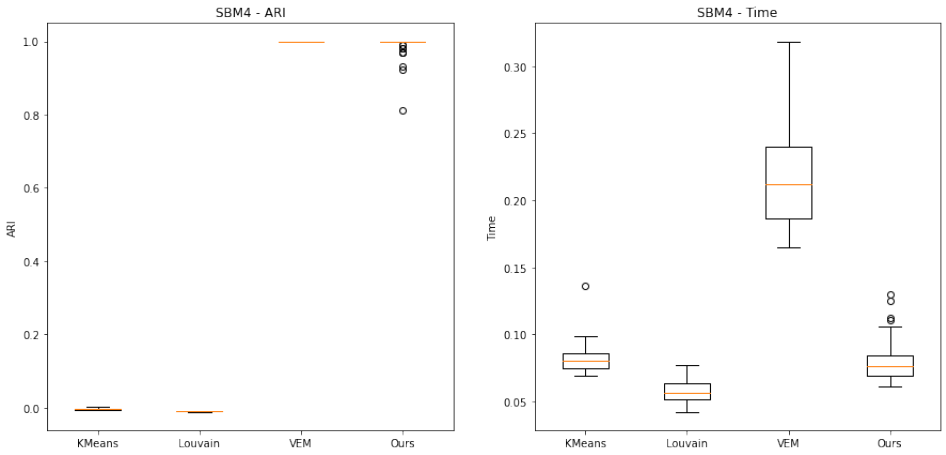


Figure 4.32: Results SBM4

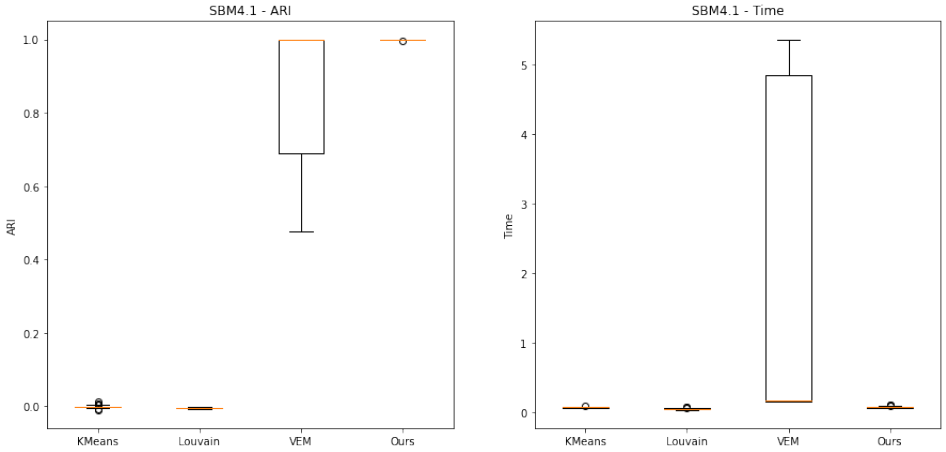


Figure 4.33: Results SBM4.1

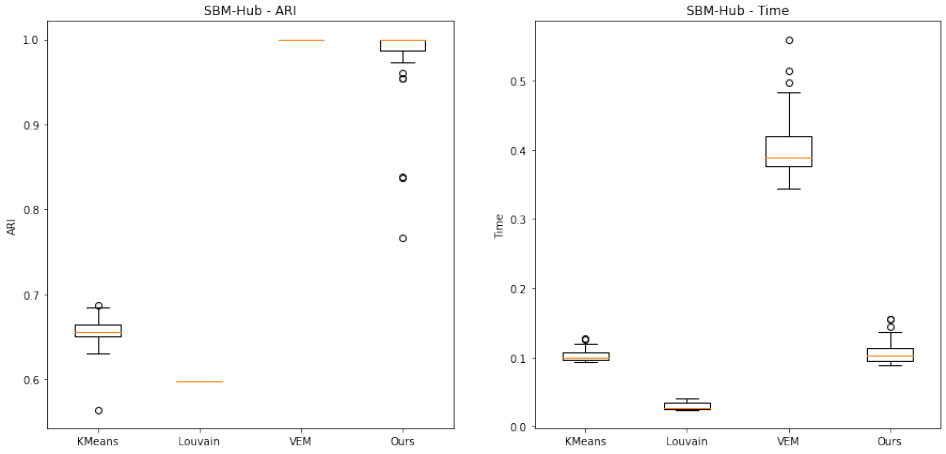


Figure 4.34: Results SBM-Hub

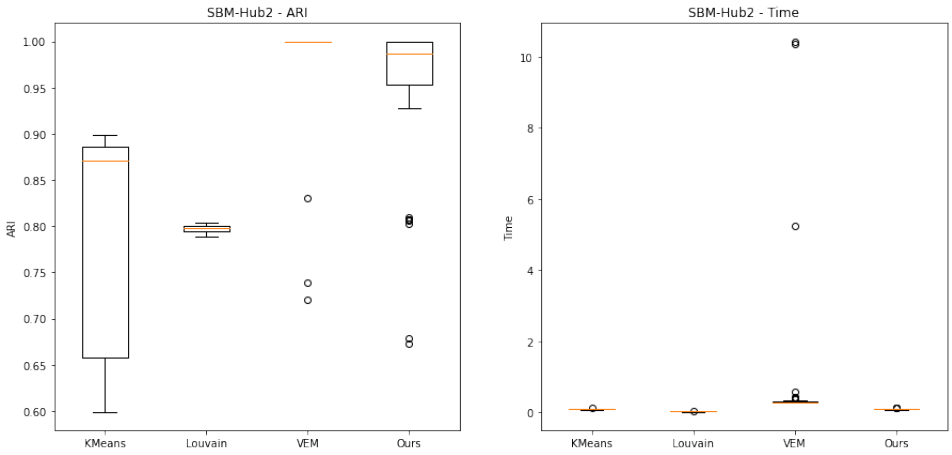


Figure 4.35: Results SBM-Hub2

We observe that the results provided by our algorithm are very similar to the ones provided by the variational EM algorithm. The adjusted Rand scores of our method are a little below when the proportions of the clusters are different (4.30, 4.35). In the case of $p_{in} > p_{out}$ (4.28, 4.29), the results are very close to the others 3 algorithms. When $p_{in} < p_{out}$ (4.31, 4.32), we see that as expected, the spectral clustering and Louvain are terrible, and our algorithm performs almost similarly as the variational EM. For the hubs (4.34, 4.35), our algorithm is slightly worse than the variational EM but is better than the two others. Finally, in terms of time, our method is almost as fast as the spectral clustering which is promising as the implementation used numpy matrix operations, and was not further optimized.

5 Conclusion

We saw through this work that the proposed algorithm (3) in Euclidean space such as $\mathbb{R}^d$ is quite promising. Firstly, it has a better complexity than determinantal point processes sampling algorithm, and it also seems to provide better initialization sample for the Online Classification EM algorithm. On the other hand, as it was implemented, it is currently slower than the classical Classification EM algorithm on the whole dataset. There is still work to be done in the implementation in order to see whether it can be improved or not. Moreover, the choice of the parameters for a good subsampling seems rather complicated or even intractable in some cases, for example when there are several areas with data on different scales. Finally, the probability of a sample seems to be complicated to compute, and it is therefore a drawback for the use of coreset analysis for example. This last point seems to be the main advantage of the DPP sampling, because the probability of sampling a set can be computed in closed form, and can therefore be used for coreset analysis, despite the huge cost of the algorithm.

In the graph setting, we saw that the complexity was good enough compared to other algorithms, and the results on the different statistics which are retained in the subsample are comparable to others. Unfortunately, we do not see it shine on a specific statistic either. Moreover, the results when using our subsamples as initialization sets of the Online Classification algorithm were bad.

However, following the work to study the algorithm of subsampling on graphs, we also tried a clustering algorithm which seems promising as it is supposed to be fast, and it seems to cluster stochastic block models graphs well.

References

Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

Nima Anari, Shayan Oveis Gharan, and Alireza Rezaei. Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes. In *Conference on Learning Theory*, pages 103–115, 2016.

Konstantin Avrachenkov, Pavel Chebotarev, and Dmytro Rubanov. Kernels on graphs as proximity measures. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 27–41. Springer, 2017a.

Konstantin Avrachenkov, Pavel Chebotarev, and Dmytro Rubanov. Kernels on graphs as proximity measures, 2017b.

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

Fredrik Backåker. The google markov chain: convergence speed and eigenvalues, 2012.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Jon Louis Bentley. Decomposable searching problems. Technical report, Carnegie-Mellon Univiversity Pittsburgh PA Department of Computer Science, 1978.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

Alexei Borodin and Eric M Rains. Eynard–mehta theorem, schur process, and their pfaffian analogs. *Journal of statistical physics*, 121(3-4):291–317, 2005.

Pierre Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer Science & Business Media, 2013.

David R Burt, Carl Edward Rasmussen, and Mark van der Wilk. Convergence of sparse variational inference in gaussian processes regression. *Journal of Machine Learning Research*, 21(131):1–63, 2020.

Gilles Celeux and Gérard Govaert. A classification em algorithm for clustering and two stochastic versions. *Computational statistics & Data analysis*, 14(3):315–332, 1992.

Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)*, 37(2):200–212, 1990.

P Yu Chebotarev and EV Shamis. On a duality between metrics and σ-proximities. *arXiv preprint math/0508183*, 2005.

Julien Chiquet, Pierre Barbillon, and Timothée Tabouy. Political blogosphere network prior to 2007 french presidential election. http://julien.cremeriefamily.info/missSBM/reference/frenchblog2007.html#source, 2007.

Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

J-J Daudin, Franck Picard, and Stéphane Robin. A mixture model for random graphs. *Statistics and computing*, 18(2):173–183, 2008.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Michal Derezinski, Daniele Calandriello, and Michal Valko. Exact sampling of determinantal point processes with sublinear time preprocessing. In *Advances in Neural Information Processing Systems*, pages 11546–11558, 2019.

Dan Feldman, Matthew Faulkner, and Andreas Krause. Scalable training of mixture models via coresets. In *Advances in neural information processing systems*, pages 2142–2150, 2011.

Mike Gartrell, Victor-Emmanuel Brunel, Elvis Dohmatob, and Syrine Krichene. Learning nonsymmetric determinantal point processes. In *Advances in Neural Information Processing Systems*, pages 6718–6728, 2019.

Guillaume Gautier, Guillermo Polito, Rémi Bardenet, and Michal Valko. DPPy: DPP Sampling with Python. *Journal of Machine Learning Research - Machine Learning Open Source Software (JMLR-MLOSS)*, 2019. Code at http://github.com/guilgautier/DPPy/ Documentation at http://dppy.readthedocs.io/.

Jean Ginibre. Statistical ensembles of complex, quaternion, and real matrices. *Journal of Mathematical Physics*, 6(3):440–449, 1965.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

J Ben Hough, Manjunath Krishnapur, Yuval Peres, Bálint Virág, et al. Determinantal processes and independence. *Probability surveys*, 3:206–229, 2006.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2 (1):193–218, 1985.

Christian Hübler, Hans-Peter Kriegel, Karsten Borgward, and Zoubin Ghahramani. Metropolis algorithms for representative subgraph sampling. In *2008 Eighth IEEE International Conference on Data Mining*, 2008.

Eric D Kolaczyk and Gábor Csárdi. *Statistical analysis of network data with R*, volume 65. Springer, 2014.

Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete input spaces. 2002.

Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *ICML*, 2011.

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *arXiv preprint arXiv:1207.6083*, 2012.

Jérôme Kunegis. Caenorhabditis elegans network dataset – KONECT, 2017. URL http://konect.cc/networks/arenas-meta.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

Claire Launay, Bruno Galerne, and Agnès Desolneux. Exact sampling of determinantal point processes without eigendecomposition. *Journal of Applied Probability*, 2020.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, 2006.

Chengtao Li, Stefanie Jegelka, and Suvrit Sra. Fast dpp sampling for nystr\" om with application to kernel methods. *arXiv preprint arXiv:1603.06052*, 2016.

Odile Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 7(1):83–122, 1975.

Arun S. Maiya and Tanya Y. Berger-Wolf. Sampling community structure, 2010.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Jack Poulson. High-performance sampling of generic determinantal point processes. *Philosophical Transactions of the Royal Society A*, 378(2166):20190059, 2020.

Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization, 2015. URL http://networkrepository.com.

Allou Samé, Christophe Ambroise, and Gérard Govaert. An online classification em algorithm based on the mixture model. *Statistics and Computing*, 17(3):209–218, 2007.

Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003.

Michael P.H. Stumpf, Carsten Wiuf, and Robert M. May. Subnets of scale-free networks are not scale-free: Sampling properties of networks. *Proceedings of the National Academy of Sciences*, 2005.

Nicolas Tremblay, Pierre-Olivier Amblard, and Simon Barthelmé. Graph sampling with determinantal processes. *arXiv:1703.01594 [cs, stat]*, 2017.

Nicolas Tremblay, Simon Barthelmé, and Pierre-Olivier Amblard. Determinantal point processes for coresets. *arXiv preprint arXiv:1803.08700*, 2018.

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

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H Chi, and Jennifer Gillenwater. Practical diversified recommendations on youtube with determinantal point processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2165–2173, 2018.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

Hugo Zanghi, Christophe Ambroise, and Vincent Miele. Fast online graph clustering via erdös-renyi mixture. *Pattern recognition*, 41(12):3592–3599, 2008.

Jianpeng Zhang, Yulong Pei, George Fletcher, and Mykola Pechenizkiy. Evaluation of the sample clustering process on graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Appendix

A1 Comparison of Sampling Methods

We compare here the different algorithms on their abilities to keep the properties stated in section 4.1.3 on the graphs of section 4.1.4.

As a reminder, we denoted the degree as *d*, the clustering coefficient as *CC*, the modularity as *Mod*, the separability as *Sep* and the conductance as *Cond*. And we took averages over 10 or 100 runs following the execution time of the algorithms.

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.17	0.26	0.22	0.04	0.11	0.49	1.32
RWRJ	0.23	0.27	0.19	0.2	0.43	0.6	4.06
MHwC	0.24	0.22	0.2	0.09	0.04	0.62	2.04
XMC	0.17	0.27	0.33	0.07	0.12	0.45	1.24
DPP	0.19	0.26	0.16	0.04	0.08	0.59	1.89
DPPAdj	0.22	0.24	0.43	0.09	0.28	0.84	2.8
DPPReg	0.23	0.24	0.27	0.12	0.18	0.69	3.01
GRF	0.27	0.23	0.26	0.07	0.03	0.56	0.59
GRFAll	0.21	0.24	0.37	0.08	0.04	0.58	1.43
GRFR	0.23	0.24	0.12	0.06	0.02	0.43	0.47

Table A1.1: Results on the dataset SBM1

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.12	0.17	0.33	0.04	0.05	0.72	2.29
RWRJ	0.12	0.18	0.24	0.16	0.12	0.71	2.69
MHwC	0.15	0.14	0.37	0.08	0.12	0.77	2.66
XMC	0.13	0.17	0.37	0.05	0.06	0.7	2.24
DPP	0.13	0.17	0.33	0.03	0.04	0.73	2.16
DPPAdj	0.21	0.14	0.9	0.14	0.27	0.74	3.9
DPPReg	0.26	0.13	0.6	0.23	0.2	0.71	2.9
GRF	0.16	0.16	1.04	0.25	0.24	0.38	0.75
GRFAll	0.15	0.15	0.63	0.16	0.05	0.74	2.43
GRFR	0.14	0.17	0.56	0.15	0.25	0.48	0.84

Table A1.2: Results on the dataset SBM2

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.18	0.02	0	0.04	0.02	0.82	1.31
RWRJ	0.18	0.02	0	0.05	0.02	0.84	1.33
MHwC	0.19	0.01	0	0.03	0.05	1	1.98
XMC	0.18	0.02	0	0.03	0.02	0.84	1.3
DPP	0.16	0.02	0	0.03	0.02	0.89	1.5
DPPAdj	0.2	0.02	0	0.05	0.02	0.92	1.66
DPPReg	0.24	0.01	0.03	0.09	0.04	0.96	1.8
GRF	0.38	0.03	0.21	0.52	0.45	1.7	0.66
GRFAll	0.23	0.02	0.05	0.15	0.03	0.12	0.84
GRFR	0.16	0.03	0	0.07	0.08	0.5	0.54

Table A1.3: Results on the dataset SBM3

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.13	0.11	0.48	0.04	0.02	0.85	0.77
RWRJ	0.13	0.11	0.41	0.07	0.02	0.84	0.75
MHwC	0.17	0.09	0.5	0.04	0.03	0.88	0.81
XMC	0.14	0.11	0.5	0.03	0.02	0.86	0.78
DPP	0.13	0.11	0.5	0.04	0.02	0.87	0.8
DPPAdj	0.2	0.1	0.5	0.14	0.05	0.95	0.95
DPPReg	0.23	0.09	0.65	0.09	0.08	0.96	0.96
GRF	0.15	0.11	0.5	0.02	0.02	0.51	0.43
GRFAll	0.16	0.1	0.52	0.09	0.05	0.91	0.87
GRFR	0.13	0.12	0.2	0.08	0.08	0.58	0.42

Table A1.4: Results on the dataset SBM4

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.17	0.24	0	0.03	0.08	0.84	2.52
RWRJ	0.18	0.25	0.01	0.08	0.16	0.81	2.86
MHwC	0.2	0.23	0	0.07	0.08	0.87	1.93
XMC	0.16	0.24	0	0.03	0.07	0.85	2.41
DPP	0.21	0.24	0	0.05	0.08	0.87	2.2
DPPAdj	0.24	0.21	0.4	0.11	0.33	0.94	4.18
DPPReg	0.28	0.21	0.35	0.08	0.44	0.97	4.58
GRF	0.66	0.2	0.15	1.04	1	1	1
GRFAll	0.19	0.23	0.07	0.04	0.07	0.9	0.96
GRFR	0.2	0.25	0	0.07	0.34	0.03	0.3

Table A1.5: Results on the dataset SBM5

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.26	0.16	0	0.04	0.87	0.82	2.71
RWRJ	0.3	0.16	0	0.06	1.11	0.84	2.69
MHwC	0.31	0.14	0	0.05	2.15	0.87	2.79
XMC	0.27	0.16	0	0.04	0.83	0.83	2.72
DPP	0.23	0.17	0	0.03	0.45	0.82	2.58
DPPAdj	0.35	0.14	0.3	0.09	1.96	0.94	5.58
DPPReg	0.32	0.14	0.4	0.11	1.53	0.93	5.63
GRF	0.32	0.16	0.5	0.38	8	0.44	0.08
GRFAll	0.28	0.15	0.04	0.06	1.35	0.87	3.39
GRFR	0.17	0.16	0	0.04	0.87	0.82	2.71

Table A1.6: Results of our Algorithms on the dataset SBM-Hub

Algos	d	CC	Diameter	Density	Mod	Sep	Cond
RN	0.22	0.27	0.25	0.22	0.24	0.51	0.35
RWRJ	0.19	0.23	0.14	1.63	0.35	0.77	0.35
MHwC	0.19	0.2	0.32	1.15	0.51	0.92	0.33
XMC	0.23	0.24	0.3	0.9	0.27	2.2	0.44
DPP	0.2	0.24	0.06	0.83	0.5	0.55	0.64
DPPAdj	0.3	0.28	0.36	0.37	0.61	0.74	0.72
DPPReg	0.38	0.29	0.56	0.48	0.4	0.87	0.82
GRF	0.38	0.3	0.73	0.73	0.26	0.91	0.82
GRFAll	0.38	0.29	0.44	0.82	0.24	0.67	0.52
GRFR	0.16	0.16	0.05	1.46	0.43	0.28	0.13

Table A1.7: Results on the dataset Blogs (Chiquet et al. [2007])

Algorithms	Degree	Clustering Coefficient	Diameter	Density
RN	0.43	0.26	0.53	0.02
RWRJ	0.14	0.25	0.6	4.4
MHwC	0.37	0.14	0.86	3.31
XMC	0.39	0.24	0.87	3.08
DPP	0.31	0.23	0.71	0.67
DPPAdj	0.51	0.25	0.92	1.22
DPPReg	0.73	0.26	0.97	0.63
GRF	0.46	0.26	0.94	0.88
GRFAll	1	0.26	1	1
GRFR	0.21	0.12	0.23	0.72

Table A1.8: Results on the dataset Net Science (Rossi and Ahmed [2015])

Algorithms	Degree	Clustering Coefficient	Diameter	Density
RN	0.33	0.05	0.29	0.1
RWRJ	0.49	0.03	0.33	4.6
MHwC	0.45	0.03	0.36	1.39
XMC	0.45	0.05	0.47	0.34
DPP	0.45	0.04	0.26	1.93
DPPAdj	0.81	0.05	0.84	0.76
DPPReg	0.86	0.05	0.94	0.66
GRF	0.66	0.05	0.53	0.3
GRFAll	0.77	0.05	0.89	1
GRFR	0.42	0.03	0.15	2.88

Table A1.9: Results on the dataset Metabolic (Kunegis [2017])