SparseBM: A Python Module for Handling Sparse Graphs with Block Models

Gabriel Frisch, Jean-Benoist Leger, Yves Grandvalet

SparseBM: A Python Module for Handling Sparse Graphs with Block Models

Gabriel Frisch[1], Jean-Benoist Leger[1], and Yves Grandvalet[1]

[1] *Université de technologie de Compiègne, CNRS, Heudiasyc (Heuristics , and Diagnosis of Complex Systems), CS 60 319 - 60 203 Compiègne Cedex*

Abstract

The stochastic and latent block models are clustering and coclustering tools that are commonly used for network analyses, such as community detection or collaborative filtering. We present a variational inference algorithm for the stochastic block model and the latent block model for sparse graphs, which leverages on the sparsity of edges to scale up to a very large number of nodes. This algorithm is implemented in **SparseBM**, a Python module that takes advantage of the hardware speed up provided by graphics processing units (GPU).

1 Introduction

Dense graphs are usually represented by adjacency matrices as illustrated in Figure 1. When the average degree of vertices is low, most elements of the adjacency matrix are zero; the matrix is sparse. Such type of graph is commonly found in datasets generated from social networks or collaborative systems. For instance, the Movielens-25M dataset [Harper and Konstan, 2015] can be model by a bipartite network made of 120,000 users and 60,000 movies vertices with an average degree of 112 or by a biadjacency matrix with a sparsity rate of 97.7%. In such a context, the size of the adjacency matrix poses a computational problem for handling the model, be it the stochastic block model (SBM) [Holland et al., 1983] or the latent block model (LBM) [Govaert and Nadif, 2008, Nadif and Govaert, 2010].

These generative models for random graphs rely on mixtures, assuming that the observations are generated from finite mixture components in rows and columns. They have found applications in many areas such as text analysis [Selosse et al., 2020], genomic analysis [Aubert et al., 2016], ecology [Bar-Hen et al., 2020], collaborative filtering [Corneli et al., 2020], or political analysis [Latouche et al., 2011, Wyse and Friel, 2012]. These probabilistic models provide a co-clustering analysis of the nodes of a graph that can be compared, among others, spectral methods [Dhillon, 2001, Kluger et al., 2003], mutual information methods [Dhillon et al., 2003], modularity based methods [Labiod and Nadif, 2011] or non-negative matrix tri-factorization [Ding et al., 2006].

Though adjacency lists are routinely used to represent sparse graphs in a compact way, the packages developed for SBM and LBM [Leger, 2016, Bhatia et al., 2017] rely on computations on dense adjacency matrices to benefit from the computational efficiency offered by matrix calculus. In this article, we show how to efficiently conduct inference with the SBM and LBM in very large sparse graphs, using a computational representation based on an adjacency list instead of an adjacency

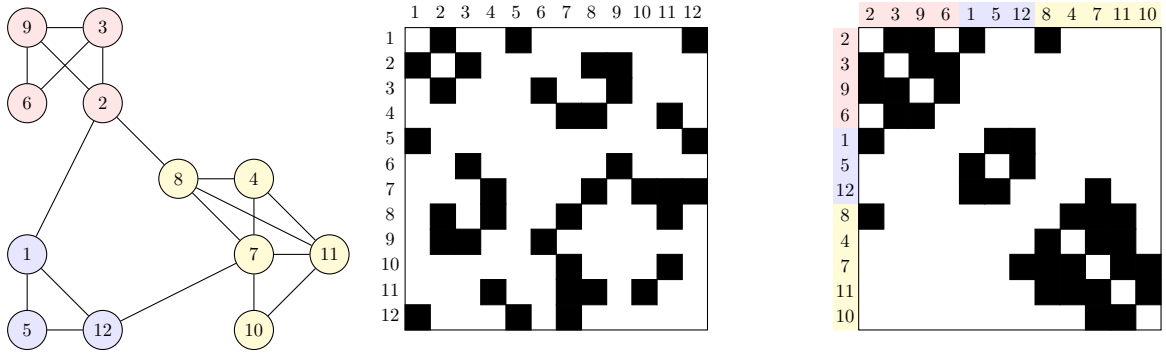


Figure 1: A binary graph on the left and its adjacency matrix on the center. The matrix on the right is the adjacency matrix reorganized according to the node clustering infered by the stochastic block model.

matrix. This inference optimized for sparse graphs is implemented in **SparseBM**, a Python module that also takes advantage of the hardware speed up provided by graphics processing units (GPUs). Our contributions allow the analysis of graphs whose size is beyond the reach of current SBM and LBM implementations.

The article is organized as follows; we present the mathematical foundation of the stochastic and latent block models in Section 2. Section 3 describes the original variational inferences and the ones we propose to reduce the complexity for sparse graphs. We provide an overview of the functionalities of the **SparseBM** module through the various examples of Section 4. Section 5 then reports experiments on synthetic datasets that show that our computational tricks are relevant to analyze sparse matrices.

2 Stochastic and latent block models

2.1 Notation

Let n_1 be the number of vertices of a graph and n_2 the number of vertices of the second set when considering a bipartite graph. Sums and products relative to the first and second set of vertices (if bipartite) will be indexed respectively by i and j, and the classes of these two types of vertices will be indexed by $q \in \{1, \dots, k_1\}$ and $l \in \{1, \dots, k_2\}$. The bounds of summations or products will be implicit, for example $\sum_i$ will be a shorthand for $\sum_{i=1}^{n_1}$ and $\prod_{ijql}$ for $\prod_{i=1}^{n_1} \prod_{j=1}^{n_2} \prod_{q=1}^{k_1} \prod_{l=1}^{k_2}$. We use set-builder notation to describe sets that are defined by a predicate, rather than explicitly enumerated, a colon separator in sums and products specifying this domain. For example, $\sum_{ijql:\, i \neq j,\, X_{ij}=1}$ is the quadruple sum on i, j, q, and l, such as the indices i and j are not equal and $X_{ij} = 1$, that is, an edge is present from vertex i to vertex j.

2.2 Stochastic block model

The binary stochastic block model (SBM) is a probabilistic model that classifies the vertices of a graph. It is typically used to model the relationships (represented by edges) between homogeneous objects (represented by vertices). For instance, a social network can be represented with a graph,

possibly directed, in which the vertices are people and the edges are their interactions. Each edge from vertex i to vertex j is associated to a random variable X_{ij} that codes for its presence: $X_{ij} = 1$ if an edge is present and $X_{ij} = 0$ otherwise. The SBM assumes a partition of the vertices that corresponds to a strong structure of the $(n_1 \times n_1)$ adjacency matrix $\boldsymbol{X}$ in homogeneous blocks. This block structure is unveiled by reordering the rows and columns of $\boldsymbol{X}$ according to their class index; for k_1 classes, the reordering reveals $k_1 \times k_1$ homogeneous blocks in the adjacency matrix. The partition is governed by the latent variable $\boldsymbol{U}$, the $n_1 \times k_1$ indicator matrix of classes ($U_{iq} = 1$ if vertex i belongs to class q and $U_{iq} = 0$ otherwise). The class indicator of vertex i will be denoted $\boldsymbol{U}_i$. The SBM makes several assumptions on the dependencies:

Vertex classes are independent and identically distributed The latent variables $\boldsymbol{U}_i$ are independent and follow a multinomial distribution $\mathcal{M}(1; \boldsymbol{\alpha})$, where $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_{k_1})$ are the mixing proportions of vertices:

$$\begin{aligned} \mathsf{P}(\boldsymbol{U}; \boldsymbol{\alpha}) &= \prod_i \mathsf{P}(\boldsymbol{U}_i; \boldsymbol{\alpha}) \\ \mathsf{P}(U_{iq} = 1; \boldsymbol{\alpha}) &= \alpha_q \quad , \end{aligned}$$

with $\boldsymbol{\alpha} \in S_{(k_1-1)} = \{\boldsymbol{\alpha} \in \mathbb{R}_+^{k_1} | \sum_q \alpha_q = 1\}$.

Given the vertex classes, the edge presences are independent and identically distributed Given the vertex classes $\boldsymbol{U}$, the edge presences $\boldsymbol{X}$ are independent and follow a Bernoulli distribution of parameter $\boldsymbol{\pi} = (\pi_{ql}; q = 1, ..., k_1; l = 1, ..., k_1)$: the probability of the presence of edge X_{ij} depends only on the classes of the two vertices i and j.

$$\begin{aligned} \mathsf{P}(\boldsymbol{X} | \boldsymbol{U}; \boldsymbol{\pi}) &= \prod_{ij} \mathsf{P}(X_{ij} | \boldsymbol{U}_i, \boldsymbol{U}_j; \boldsymbol{\pi}) \\ \mathsf{P}(X_{ij} = 1 | U_{iq} U_{jl} = 1; \boldsymbol{\pi}) &= \pi_{ql} \quad . \end{aligned}$$

To summarize, the parameters of the SBM are $\theta = (\boldsymbol{\alpha}, \boldsymbol{\pi})$ and the probability mass function of $\boldsymbol{X}$ can be written as:

$$\mathsf{P}(\boldsymbol{X}; \theta) = \sum_{\boldsymbol{U} \in I} \left(\prod_{iq} {\alpha_q}^{U_{iq}} \right) \left(\prod_{jl} {\alpha_l}^{U_{jl}} \right) \left(\prod_{iqjl} \phi(X_{ij}; \pi_{ql})^{U_{iq} U_{jl}} \right) \quad ,$$

where $\phi(X_{ij}; \pi_{ql}) = \pi_{ql}^{X_{ij}} (1 - \pi_{ql})^{1 - X_{ij}}$ is the mass function of a Bernoulli variable, and where I denotes the set of all possible partitions of the n_1 vertices into k_1 groups.

2.3 Latent Block Model

The binary latent block model (LBM) can be seen as an extended binary SBM that co-classifies the vertices of a bipartite graph. It is typically used to model the relationships between two types of homogeneous objects, represented by nodes of type (1) and nodes of type (2). The LBM forms a double partition with k_1 groups in the set of vertices of type (1) and k_2 groups in the set of vertices of type (2). For instance, a recommendation system can be represented with a bipartite graph in which type-(1) vertices are people, type-(2) vertices are items, and edges represent purchases. Each edge from type-(1) vertex i to type-(2) vertex j is associated to a random variable X_{ij} coding for

its presence. The partitions of the two sets of vertices are governed by the latent variables $\boldsymbol{U}$ and $\boldsymbol{V}$, $\boldsymbol{U}$ being the $n_1 \times k_1$ indicator matrix of the classes of type-(1) vertices, and $\boldsymbol{V}$ being the $n_2 \times k_2$ indicator matrix of the classes of type-(2) vertices. The class indicator of type-(1) vertex i will be denoted $\boldsymbol{U}_i$, and similarly, the class indicator of type-(2) vertex j will be denoted $\boldsymbol{V}_j$. The $(n_1 \times n_2)$ binary matrix $\boldsymbol{X}$ can be seen as the biadjacency matrix of the bipartite graph. The LBM makes assumptions similar to those of the SBM on dependencies:

Independent and identically distributed vertex classes of the two partitions The latent variables $\boldsymbol{U}$ and $\boldsymbol{V}$ are independent and follow respectively the multinomial distributions $\mathcal{M}(1;\boldsymbol{\alpha})$ and $\mathcal{M}(1;\boldsymbol{\beta})$, where $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_{k_1})$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_{k_2})$ are the mixing proportions of vertices of the sets (1) and (2):

$$\mathsf{P}(\boldsymbol{U}, \boldsymbol{V}) = \prod_i \mathsf{P}(\boldsymbol{U}_i;\boldsymbol{\alpha}) \prod_j \mathsf{P}(\boldsymbol{V}_j;\boldsymbol{\beta})$$
$$\mathsf{P}(U_{iq} = 1;\boldsymbol{\alpha}) = \alpha_q \text{ and } \mathsf{P}(V_{jl} = 1;\boldsymbol{\beta}) = \beta_l \ ,$$

with $\boldsymbol{\alpha} \in S_{(k_1-1)}$ and $\boldsymbol{\beta} \in S_{(k_2-1)}$.

Given vertex classes, independent and identically distributed block entries in the biadjacency matrix Given $\boldsymbol{U}$ and $\boldsymbol{V}$, the classes of vertices, the biadjacencies X_{ij} are independent and follow a Bernoulli distribution of parameter $\boldsymbol{\pi} = (\pi_{ql}; q = 1, ..., k_1; l = 1, ..., k_2)$: all elements of a block follow the same probability distribution.

To summarize, the parameters of the LBM are $\theta = (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi})$ and the probability mass function of $\boldsymbol{X}$ can be written as:

$$\mathsf{P}(\boldsymbol{X};\theta) = \sum_{(\boldsymbol{U},\boldsymbol{V}) \in I \times J} \left(\prod_{iq} \alpha_q{}^{U_{iq}} \right) \left(\prod_{jl} \beta_l{}^{V_{jl}} \right) \left(\prod_{ijql} \phi(X_{ij};\pi_{ql})^{U_{iq}V_{jl}} \right) \ ,$$

where $\phi(X_{ij};\pi_{ql}) = \pi_{ql}^{X_{ij}}(1-\pi_{ql})^{1-X_{ij}}$ is the mass function of a Bernoulli variable, and where I (resp. J) denotes the set of all possible partitions of the n_1 type-(1) vertices (resp. n_2 type-(2) vertices) into k_1 (resp. k_2) groups.

3 Estimation procedure

3.1 Computationally efficient variational inference for sparse graphs

The generative modelling the SBM and LBM can be split into a set of unobserved latent variables and a set of observed variables consisting of $\boldsymbol{X}$ only. An observation of $\boldsymbol{X}$ is referred to as incomplete data, and an observation of $\boldsymbol{X}$ together with the latent variables is referred to as complete data.

Given the incomplete data, the objective is to infer the model parameters θ via maximum likelihood $\widehat{\theta} = \arg\max_\theta \mathsf{P}(\boldsymbol{X};\theta)$. When applying the Expectation Maximization (EM) algorithm to the SBM or to the LBM to maximize $\mathsf{P}(\boldsymbol{X};\theta)$, the computation of the complete log-likelihood at the E-step requires the posterior distribution of the latent variables, which is intractable, because the search space of the latent variables is combinatorially too large [Brault and Mariadassou, 2015].

This problem is well known in the context of co-clustering; for both SBM and LBM, some methods [Celeux and Diebolt, 1985, Keribin et al., 2015] rely on a stochastic E-step with Monte Carlo sampling,

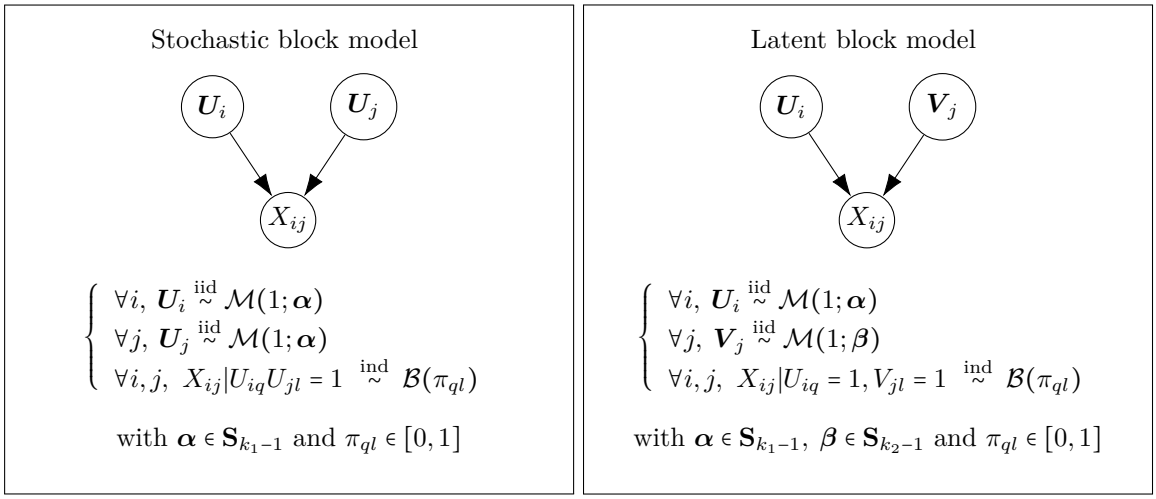


Figure 2: Summary of the standard stochastic block model (left) and latent block model (right) with binary data.

but these strategies are not suited to large-scale problems. We follow the variational reformulation of the problem that is more efficient in high dimension. The variational EM (VEM) [Jordan et al., 1999, Jaakkola, 2000] introduces q_γ, a restricted set of parametric distributions defined over the latent variables, and maximizes the following lower bound on the log-likelihood of the incomplete data:

$$\mathcal{J}(q_\gamma, \theta) = \log \mathsf{P}(\boldsymbol{X};\theta) - KL(q_\gamma \parallel \mathsf{P}(\cdot|\boldsymbol{X};\theta)) \ , \tag{1}$$

where KL stands for the Kullback-Leibler divergence and q_γ denotes the variational distribution over the latent variables. The criterion $\mathcal{J}(q_\gamma, \theta)$ can be rewritten as as the sum of a negative "energy" and the entropy of q_γ:

$$\mathcal{J}(q_\gamma, \theta) = \mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}, \cdot\,; \theta)] + \mathcal{H}(q_\gamma) \ , \tag{2}$$

where $\mathbb{E}_{q_\gamma}$ is the expectation with respect to the variational distribution and $\mathcal{H}(q_\gamma)$ is the entropy of the variational distribution. The variational distribution q_γ is restricted to belong to a set of distributions that lead to a tractable computation of the criterion of Equation 2. Here, as is usually done in variational inference, the conditional independence of the latent variables is assumed; this is known as the "mean-field approximation" [Parisi, 1988].

3.1.1 Variational inference of the stochastic block model

The mean-field approximation applied to the stochastic block model leads to the following form of the variational distribution over the latent variable $\boldsymbol{U}$:

$$q_\gamma = \prod_i \mathcal{M}(1;\boldsymbol{\tau}_i)$$

where $\boldsymbol{\tau}_i \in \mathbf{S}_{k_1-1}$ are the parameters of the variational multinomial distributions. Using the conditional independence of the latent variable, the criterion $\mathcal{J}(q_\gamma, \theta)$ is expanded as:

$$\mathcal{J}(q_\gamma, \theta) = \mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}|\boldsymbol{U};\theta)] + \mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{U};\boldsymbol{\alpha})] + \mathcal{H}(q_\gamma) \ ,$$

where

$$\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}|\boldsymbol{U};\theta)] = \sum_{ijql:\, i\neq j} \tau_{iq}\tau_{jl}(X_{ij}\log\pi_{ql} + (1-X_{ij})\log(1-\pi_{ql})) \tag{3}$$

$$\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{U};\boldsymbol{\alpha})] = \sum_{iq} \tau_{iq}\log\alpha_q$$

$$\mathcal{H}(q_\gamma) = -\sum_{iq} \tau_{iq}\log\tau_{iq} \ .$$

As Equation 3 involves a sum on all the non-diagonal elements of the adjacency matrix $\boldsymbol{X}$, the computation of the criterion $\mathcal{J}(q_\gamma,\theta)$ is of complexity $\mathcal{O}({n_1}^2{k_1}^2)$ where n_1 is the number of vertices and k_1 is the number of classes. When the considered graph is large, this complexity becomes problematic: the adjacency matrix may not fit in memory and/or the computation time may be prohibitive. However, Equation (3) can be rewritten by summing only the non-zero elements of the adjacency matrix, lowering the complexity to $\mathcal{O}(\#\{ij : X_{ij}=1\}{k_1}^2)$ where $\#\{ij : X_{ij}=1\}$ is the number of non-zero entries in $\boldsymbol{X}$:

$$\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}|\boldsymbol{U};\theta)] = \sum_{ijql:\, i\neq j,\, X_{ij}=1} \tau_{iq}\tau_{jl}(\log\pi_{ql} - \log(1-\pi_{ql}))$$
$$+ \sum_{ql}\log(1-\pi_{ql})\left(\left(\sum_i \tau_{iq}\right)\left(\sum_j \tau_{jl}\right) - \sum_i \tau_{iq}\tau_{il}\right) \ .$$

We give the parameter estimates as defined for the original VEM inference in Algorithm 1 and for the VEM inference optimized for sparse graphs in Algorithm 2. The memory complexity of the algorithm, originally in $\mathcal{O}({n_1}^2)$, is reduced to $\mathcal{O}(\#\{ij : X_{ij}=1\})$.

Algorithm 1: VEM - Original version

Data: Adjacency matrix $\boldsymbol{X}$

Inititialize $\boldsymbol{\tau}$, $\boldsymbol{\alpha}$, $\boldsymbol{\pi}$

while $\mathcal{J}^{(t)} - \mathcal{J}^{(t-1)} > atol$ **do**

  repeat

    $Q_{il} = \sum_{j:\, j\neq i} \tau_{jl}X_{ij}$

    $R_l = \sum_{i:\, i\neq j} \tau_{jl}$

    $\tau_{iq} \propto \alpha_q \prod_l \pi_{ql}^{Q_{il}}(1-\pi_{ql})^{R_l - Q_{il}}$

  until *convergence*;

  $\alpha_q = \frac{1}{n_1}\sum_i \tau_{iq}$

  $\pi_{ql} = \frac{\sum_{ij:\, i\neq j} \tau_{iq}\tau_{jl}X_{ij}}{\sum_{ij:\, i\neq j} \tau_{iq}\tau_{jl}}$

Algorithm 2: VEM - Sparse graph

Data: Sparse adjacency matrix $\boldsymbol{X}$

Inititialize $\boldsymbol{\tau}$, $\boldsymbol{\alpha}$, $\boldsymbol{\pi}$

while $\mathcal{J}^{(t)} - \mathcal{J}^{(t-1)} > atol$ **do**

  repeat

    $Q_{il} = \sum_{j:\, j\neq i, X_{ij}=1} \tau_{jl}$

    $\tau_{iq} \propto \alpha_q \prod_{jl:\, j\neq i}(1 - \pi_{ql})^{\tau_{jl}} \prod_l \left(\frac{\pi_{ql}}{(1-\pi_{ql})}\right)^{Q_{il}}$

  until *convergence*;

  $\alpha_q = \frac{1}{n_1}\sum_i \tau_{iq}$

  $\pi_{ql} = \frac{\sum_{ij:\, i\neq j, X_{ij}=1} \tau_{iq}\tau_{jl}}{(\sum_i \tau_{iq})(\sum_j \tau_{jl}) - \sum_i \tau_{iq}\tau_{il}}$

3.1.2 Variational inference of the latent block model

The mean-field approximation applied to the latent block model leads to the following form of the variational distribution over the latent variables $\boldsymbol{U}$ and $\boldsymbol{V}$:

$$q_\gamma = \prod_i \mathcal{M}\left(1;\boldsymbol{\tau}_i^{(U)}\right) \prod_j \mathcal{M}\left(1;\boldsymbol{\tau}_j^{(V)}\right) \ ,$$

where $\boldsymbol{\tau}_i^{(U)}$ and $\boldsymbol{\tau}_j^{(V)}$ are respectively the parameters of the variational multinomial distributions over the latent variables $\boldsymbol{U}$ and $\boldsymbol{V}$. Using the conditional independence of the latent variable, the criterion $\mathcal{J}(q_\gamma, \theta)$ is expanded as:

$$\mathcal{J}(q_\gamma, \theta) = \mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}|\boldsymbol{U}, \boldsymbol{V}; \theta)] + \mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{U}; \boldsymbol{\alpha})] + \mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{V}; \boldsymbol{\beta})] + \mathcal{H}(q_\gamma) \ ,$$

where

$$\begin{aligned}
\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}|\boldsymbol{U}, \boldsymbol{V}; \theta)] &= \sum_{ijql} \tau_{iq}^{(U)} \tau_{jl}^{(V)} (X_{ij} \log \pi_{ql} + (1 - X_{ij}) \log(1 - \pi_{ql})) \qquad (4) \\
\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{U}; \boldsymbol{\alpha})] &= \sum_{iq} \tau_{iq}^{(U)} \log \alpha_q \\
\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{V}; \boldsymbol{\beta})] &= \sum_{jl} \tau_{jl}^{(V)} \log \beta_l \\
\mathcal{H}(q_\gamma) &= -\sum_{iq} \tau_{iq}^{(U)} \log \tau_{iq}^{(U)} - \sum_{jl} \tau_{jl}^{(U)} \log \tau_{jl}^{(V)} \ .
\end{aligned}$$

Equation 4 is rewritten analogously to Equation 3, reducing the computational complexity of $\mathcal{J}(q_\gamma, \theta)$ from $\mathcal{O}(n_1 n_2 k_1 k_2)$ to $\mathcal{O}(\#\{ij : X_{ij} = 1\} k_1 k_2)$:

$$\begin{aligned}
\mathbb{E}_{q_\gamma}[\log \mathsf{P}(\boldsymbol{X}|\boldsymbol{U}, \boldsymbol{V}; \theta)] = &\sum_{ijql:\, X_{ij}=1} \tau_{iq}^{(U)} \tau_{jl}^{(V)} (\log \pi_{ql} - \log(1 - \pi_{ql})) \\
&+ \sum_{ql} \log(1 - \pi_{ql}) \left(\sum_i \tau_{iq}^{(U)} \right) \left(\sum_j \tau_{jl}^{(V)} \right) \ .
\end{aligned}$$

We give the parameter estimates as defined for the original VEM inference in Algorithm 3 and for the VEM inference for sparse graphs in Algorithm 4.

Algorithm 3: VEM - Original version

Data: Adjacency matrix $\boldsymbol{X}$

Inititialize $\boldsymbol{\tau}^{(U)}$, $\boldsymbol{\tau}^{(V)}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\pi}$

while $\mathcal{J}^{(t)} - \mathcal{J}^{(t-1)} > atol$ **do**

  repeat

    $Q_{il} = \sum_j \tau_{jl}^{(V)} X_{ij}$

    $R_l = \sum_j \tau_{jl}^{(V)}$

    $\tau_{iq}^{(U)} \propto \alpha_q \prod_l \pi_{ql}^{Q_{il}} (1-\pi_{ql})^{R_l - Q_{il}}$

    $S_{jq} = \sum_i \tau_{iq}^{(U)} X_{ij}$

    $T_q = \sum_i \tau_{iq}^{(U)}$

    $\tau_{jl}^{(V)} \propto \beta_l \prod_q \pi_{ql}^{S_{jq}} (1-\pi_{ql})^{T_q - S_{jq}}$

  until *convergence*;

  $\alpha_q = \frac{\sum_i \tau_{iq}^{(U)}}{n_1}$

  $\beta_l = \frac{\sum_j \tau_{jl}^{(V)}}{n_2}$

  $\pi_{ql} = \frac{\sum_{ij} \tau_{iq}^{(U)} \tau_{jl}^{(V)} X_{ij}}{\sum_{ij} \tau_{iq}^{(U)} \tau_{jl}^{(V)}}$

Algorithm 4: VEM - Sparse graph

Data: Sparse adjacency matrix $\boldsymbol{X}$

Inititialize $\boldsymbol{\tau}^{(U)}$, $\boldsymbol{\tau}^{(V)}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\pi}$

while $\mathcal{J}^{(t)} - \mathcal{J}^{(t-1)} > atol$ **do**

  repeat

    $Q_{il} = \sum_{j:\, X_{ij}=1} \tau_{jl}^{(V)}$

    $\tau_{iq}^{(U)} \propto$

    $\alpha_q \prod_{jl} (1-\pi_{ql})^{\tau_{jl}^{(V)}} \prod_l \frac{\pi_{ql}^{Q_{il}}}{(1-\pi_{ql})^{Q_{il}}}$

    $S_{jq} = \sum_{i:\, X_{ij}=1} \tau_{iq}^{(U)}$

    $\tau_{jl}^{(V)} \propto$

    $\beta_l \prod_{iq} (1-\pi_{ql})^{\tau_{iq}^{(U)}} \prod_q \frac{\pi_{ql}^{S_{jq}}}{(1-\pi_{ql})^{S_{jq}}}$

  until *convergence*;

  $\alpha_q = \frac{\sum_i \tau_{iq}^{(U)}}{n_1}$

  $\beta_l = \frac{\sum_j \tau_{jl}^{(V)}}{n_2}$

  $\pi_{ql} = \frac{\sum_{ij:X_{ij}=1} \tau_{iq}^{(U)} \tau_{jl}^{(V)}}{\sum_i \tau_{iq}^{(U)} \sum_j \tau_{jl}^{(V)}}$

3.2 Initialization

The optimization process does not ensure convergence towards a global optimum of the criterion $\mathcal{J}(q_\gamma, \theta)$. EM-like algorithms are known to be sensitive to initialization, particularly when applied to models with discrete latent spaces, and may get stuck into unsatisfactory local maxima [Biernacki et al., 2003, Baudry and Celeux, 2015].

A simple heuristic consists in training for a few iterations from several random initializations, and pursuing optimization for the solutions with highest value of the variational criterion [see, e.g., small EM for mixtures Baudry and Celeux, 2015]. Another approach is to rely on cheaper clustering methods, such as k-means or spectral clustering, to initialize the algorithm [Shireman et al., 2015]. These methods bring out good estimates but spend a great deal of computing and memory resources when the data matrices get bigger. Some existing methods such as online k-means [MacQueen, 1967] are adapted to handle large matrices and could be used. However for simplicity reasons, the initialization procedure implemented in **SparseBM** is limited to multiple random initializations.

3.3 Selection of the number of classes

The Integrated Completed Likelihood criterion (ICL), inspired by the Bayesian Information Criterion, was originally proposed to select a relevant number of classes for mixture models [Biernacki et al., 2000]. It was extended to select an appropriate number of classes in the SBM [Daudin et al., 2008] and in the LBM [Keribin et al., 2012].

The ICL criterion for the standard SBM reads:

$$\begin{aligned} ICL_{SBM}(k_1) &= \log \int \mathsf{P}(\boldsymbol{X}, \boldsymbol{U}|\theta; k_1)\mathsf{P}(\theta; k_1)\mathrm{d}\theta \\ &\approx \max_{\theta} \log \mathsf{P}(\boldsymbol{X}, \boldsymbol{U}; \theta) - \frac{{k_1}^2}{2}\log\left(n_1(n_1-1)\right) - \frac{k_1-1}{2}\log n_1 + o(\log n_1) \ , \end{aligned}$$

with $\mathsf{P}(\theta; k_1)$ the prior distribution of parameters as set by Daudin et al. [2008].

The ICL criterion for the standard LBM reads:

$$\begin{aligned} ICL_{LBM}(k_1, k_2) &= \log \int \mathsf{P}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{V}|\theta; k_1, k_2)\mathsf{P}(\theta; k_1, k_2)\mathrm{d}\theta \\ &\approx \max_{\theta} \log \mathsf{P}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{V}; \theta) - \frac{k_1 k_2}{2}\log\left(n_1 n_2\right) \\ &\quad - \frac{k_1-1}{2}\log n_1 - \frac{k_2-1}{2}\log n_2 + o(\log n_1) + o(\log n_2) \ , \end{aligned}$$

with $\mathsf{P}(\theta; k_1, k_2)$ the prior distribution of parameters as modeled by Keribin et al. [2012]. In practice, as the log-likelihood maximum can not be computed, its variational approximation is used. By taking into account the latent variables, ICL is clustering-oriented, whereas BIC or AIC are driven by the faithfulness to the distribution of $\boldsymbol{X}$ [Biernacki et al., 2000].

Being dependent on the log-likelihood, the ICL criterion is also sensitive to the VEM solution, and thus to its initialization, which usually leads to an irregular ICL behavior during the exploration of the number of groups. To get a smoother ICL response, **SparseBM** implements a procedure, known as "split and merge" or "forward and backward" [Tabouy, 2019], that relies on the two alternated strategies to "split" and "merge" groups. Starting from a trained model with k_1 groups, the split strategy explores all models obtained by splitting one of the k_1 groups and keeps the best model estimation in terms of ICL. The split strategy brings out models with more and more groups until no model improves upon the best ICL criterion found so far, and thus for a few iterations. In our implementation the number of groups considered should not exceeds $min(1.5 \cdot n_q^{best}, n_q^{best} + 10)$ with n_q^{best} being the number of groups of the best model found so far in the split strategy. The merge strategy then starts backward, from the model with the highest number of groups, and explores all models obtained by merging two groups. It generates new model estimations with a decreasing number of groups until merging becomes pointless (e.g., from a SBM with only two groups). The split and merge procedure is repeated until no best model estimations comes out for a few iterations (two in the implementation we propose).

4 Block clustering with SparseBM

SparseBM is a Python module that implements the Bernoulli latent block model and stochastic block model variational inference, optimized for large and sparse graphs. The estimation procedure is fully written with tensor expressions to easily leverage parallel computing. The module can optionally make use of the **CuPy** library that provides GPU accelerated computing. As **CuPy** and **NumPy** share the same interface, only one agnostic code is implemented. The **SparseBM** module is distributed through the PyPI repository (`https://pypi.org/project/sparsebm/`) and the documentation is available at `https://sparsebm.readthedocs.io/`.

4.1 Installation guidelines

As the module is available through PyPI repository, it can be installed with the package installer **pip**:

```
pip install sparsebm
```

To leverage GPU accelaration, the **CuPy** module must be installed with **pip** or **anaconda** or directly with the extra argument when installing SparseBM:

```
pip install sparsebm[gpu]
```

For users that do not have GPU, we advise the free serverless Jupyter notebook environment provided by Google Colab (`https://colab.research.google.com/`) where the **Cupy** module is already installed and ready to use with one GPU.

4.2 SparseBM: A Python interface

The main features exposed to the user are:

- `generate_SBM_dataset` and `generate_LBM_dataset`, two functions optimized to generate large and sparse graphs using either the SBM or the LBM;
- `SBM` and `LBM`, two classes implementing the stochastic block model and latent block model inference optimized for sparse graphs and using the multiple random initialisations strategy;
- `ModelSelection` a class implementing the model selection algorithm based on split-merge strategy and making use of the SBM or LBM for inference.

In the following sections, we give more details and provide examples of the use of these algorithm.

Sparse network generation: network generation avoids the manipulation of dense matrices by creating the adjacency matrix $\boldsymbol{X}$ block by block.

The function `generate_SBM_dataset` generates a network from the SBM with a specified number of nodes n_1, a number of classes k_1, class proportions ($\boldsymbol{\alpha} \in \mathbf{S}_{k_1-1}$), and array of connection probabilities ($\boldsymbol{\pi} \in [0,1]^{k_1 \times k_1}$) between classes. The argument `symmetric` indicates wether the adjacency matrix is symmetric, when clustering an undirected graph. The generated sparse adjacency matrix $\boldsymbol{X}$ (from class `scipy.sparse.coo_matrix`) and the generated indicator matrix of the latent classes $\boldsymbol{U}$ are returned in a dictionary at keys "data" and "cluster_indicator".

```
>>> from sparsebm import generate_SBM_dataset
>>> import numpy as np
>>>
>>> connection_probabilities = np.array(
...     [
...         [0.1, 0.036, 0.012, 0.0614],
...         [0.036, 0.074, 0.0, 0.0],
...         [0.012, 0.0, 0.11, 0.024],
...         [0.0614, 0.0, 0.024, 0.086],
```

```
...     ]
... )
>>>
>>> dataset = generate_SBM_dataset(
...     number_of_nodes=10 ** 3,
...     number_of_clusters=4,
...     connection_probabilities=connection_probabilities,
...     cluster_proportions=np.array([0.25, 0.25, 0.25, 0.25]),
...     symmetric=True,
... )
>>> graph = dataset["data"]
>>> cluster_indicator = dataset["cluster_indicator"]
```

If no argument is given to `generate_SBM_dataset`, a random affiliation graph [Matias and Miele, 2017] is generated:

```
>>> from sparsebm import generate_SBM_dataset
>>> dataset = generate_SBM_dataset()
```

A similar function called `generate_LBM_dataset` generates a bipartite network following the LBM and returns a dictionary that contains the adjacency matrix and the indicator matrices of the row and column latent classes.

Stochastic block model: the SBM is encapsulated in the `SBM` class that inherits from the `sklearn.base.BaseEstimator` that is the base class for all estimators in scikit-learn. A number of classes k_1 should be specified with the parameter `n_clusters`, otherwise the default value 5 is used. If the **Cupy** module is installed, the class uses the GPU with the largest memory available. The parameter `use_gpu` can disable this behaviour and the parameter `gpu_index` can enforce the use of a specific GPU.

The class implements the random initializations strategy that corresponds to the execution of `n_iter_early_stop` EM steps on `n_init` random initializations, followed by iterations until the convergence of the criterion for the `n_init_total_run`-best preliminary results; `n_iter_early_stop`, `n_init` and `n_init_total_run` are parameters of the class.

The convergence of the criterion $\mathcal{J}(q_\gamma, \theta)$ is declared when

$$\mathcal{J}^{(t)}(q_\gamma, \theta) - \mathcal{J}^{(t-5)}(q_\gamma, \theta) \leq (atol + rtol \cdot |\mathcal{J}^{(t)}(q_\gamma, \theta)|) \ ,$$

with $atol$ = 1e−4 and $rtol$ = 1e−10 being respectively the absolute tolerance and the relative tolerance.

```
>>> from sparsebm import SBM
>>> model = SBM(
...     n_clusters=4,
...     max_iter=10000,
...     n_init=100,
...     n_init_total_run=10,
...     n_iter_early_stop=10,
...     rtol=1e-10,
```

```
...     atol=1e-4,
...     verbosity=1,
...     use_gpu=True,
...     gpu_index=0,
... )
```

The class implements a `fit` method to learn from the adjacency matrix of a graph, being either sparse (class `scipy.sparse`) or not (class `numpy.array`):

```
>>> model.fit(graph)

---------- START RANDOM INITIALIZATIONS ----------
100 of 100 Initializations:  [100% ] [ Elapsed Time: 0:00:03 ]
---------- START TRAINING BEST INITIALIZATIONS ----------
10 of 10 Runs:  [100% ] [ Elapsed Time: 0:00:01 ]
```

When successfully inferred, the class proportions $\boldsymbol{\alpha}$ of the SBM, the array $\boldsymbol{\pi}$ of connection probabilities and the labels of the classes are available by the model properties `group_membership_probability`, `group_connection_probabilities` and `labels`. The Integrated Completed Loglikelihood can be computed with the method `get_ICL`. The inferred labels can be compared with the true ones using the adjusted rand index [Hubert and Arabie, 1985] that computes a similarity measure between two clusterings:

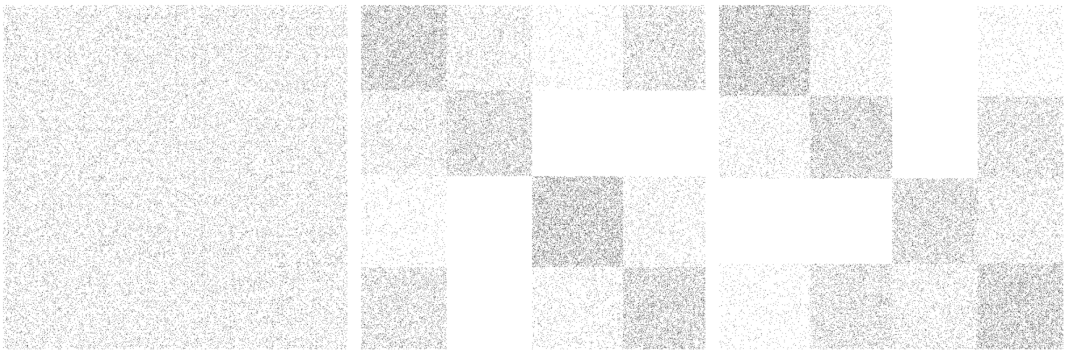
```
>>> from sparsebm.utils import ARI
>>> ari = ARI(cluster_indicator.argmax(1), model.labels)
>>> print("Adjusted Rand index is {:.2f}".format(ari))
>>> print("ICL is {:.4f}".format(model.get_ICL()))

Adjusted Rand index is 1.00
ICL is -74473.8386
```

The function `reorder_rows` reorders the rows of a sparse matrix enabling an easy visualization (see Figure 3) of the adjacency matrix reordered according to the estimated or true classes:

```
>>> from sparsebm.utils import reorder_rows
>>> reorder_rows(graph, np.argsort(model.labels))
>>> graph = graph.transpose()
>>> reorder_rows(graph, np.argsort(model.labels))
>>> graph = graph.transpose()
```

Latent block model: the `LBM` class encapsulates the latent block model and its random initialisation procedure. Its usage is similar to the `SBM` class and we refer the reader to the documentation of the **SparseBM** module or examples for more details. To measure the agreement between co-clustering partitions, the module proposes an implementation of the co-clustering adjusted rand index (CoARI) [Robert et al., 2020], which is an extension of the adjusted rand index for co-clustering.



(a) (b) (c)

Figure 3: Adjacency matrix of a network with $n_1 = 1000$ nodes generated by a SBM. The size of black pixels representing edges is enlarged for visualization reasons: (a) original adjacency matrix, (b) adjacency matrix reordered according to the true classes, (c) adjacency matrix reordered according to the classes returned by inference. Note that the permutation of classes observed between (b) and (c) is irrelevant for clustering purposes.

Model selection: the `ModelSelection` class encapsulates the model selection algorithm based on the split and merge strategy. The argument `model_type` specifies the model to use and `n_clusters_max` specifies the upper bound on the number of groups the algorithm can explore. The split strategy stops when the number of classes is greater than $min(1.5 \cdot nnq\_best, nnq\_best + 10, n\_clusters\_max)$ with nnq_best being the number of classes of the best model found so far during the split strategy. The merge strategy stops when the minimum relevant number of classes is reached. The split and merge strategy alternates until no best model is found for two iterations.

The argument `plot` specifies if an illustration is displayed to the user during the learning process (see Figure 4).

```
>>> from sparsebm import ModelSelection
>>> sbm_model_selection = ModelSelection(
...     model_type="SBM",
...     n_clusters_max=30,
...     plot=True,
...     use_gpu=True,
...     gpu_index=None,
... )
```

To learn from a sparse network, the class implements the `fit` method and returns the best model found.

```
>>> sbm_selected = sbm_model_selection.fit(graph, symmetric=True)
>>> number_of_clusters = dataset['cluster_indicator'].shape[1]
>>> print(f"Best ICL is {sbm_selected.get_ICL():.4f}")
>>> print(f"The original number of classes was {number_of_clusters}")
>>> print(f"The model selection picked {sbm_selected.n_clusters} classes")
```

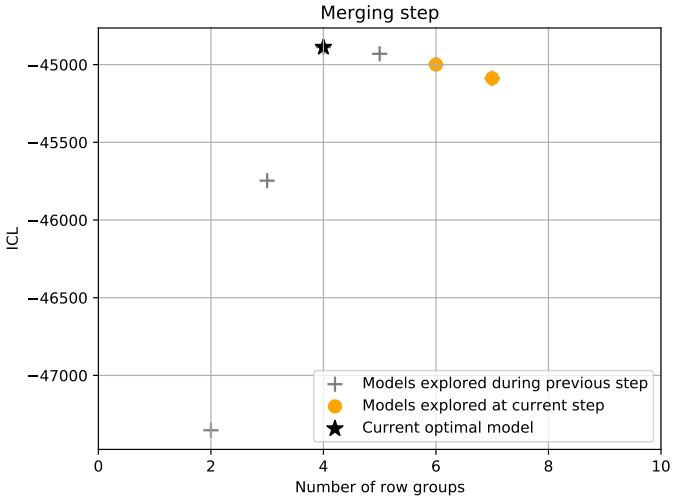


Figure 4: Illustration displayed during model selection with merge and split strategy for a SBM.

```
Best ICL is -44162.1115
The original number of classes was 4
The model selection picked 4 classes
```

Sckikit-learn integration: the SBM and LBM implemented in **Sparsebm** use the **Scikit-learn** interface style; models are thus compatible with the pipelines, model selection, and evaluation metrics. We illustrate the integration with **Scikit-learn** with a gridsearch algorithm to select the best number of classes. In this example, the `GridSearchCV` instance receives the `SBM` model and runs the algorithm with the numbers of classes specified. The model are compared together with the Integrated Completed Likelihood criterion implemented in the `SBM` model. A number of jobs to run in parallel is specified with the argument `n_jobs`; the **SparseBM** module is using all GPUs available in the system. The number of jobs in parallel should never be higher that the number of GPUs in the system.

```
>>> from sparsebm import SBM
>>> import sklearn
>>> from sklearn import metrics
>>>
>>> graph = dataset["data"]
>>> clusters_index = dataset["cluster_indicator"].argmax(1)
>>> number_of_nodes = graph.shape[0]
>>>
>>> model = SBM(verbosity=0)
>>> train = test = np.arange(number_of_nodes)
>>> n_clusters = [1, 2, 3, 4, 5, 6, 7, 8]
>>> grid = sklearn.model_selection.GridSearchCV(
```

```
...     estimator=model,
...     n_jobs=4,
...     param_grid={"n_clusters": n_clusters},
...     cv=[[train, test]],
...     verbose=1,
... )
>>> print("Start grid search algorithm")
>>> grid.fit(graph, symmetric=True)
>>> ari = metrics.adjusted_rand_score(
...     clusters_index, grid.best_estimator_.labels
... )
>>> print(
...     "Best number of classes is {} according to ICL".format(
...         grid.best_params_["n_clusters"]
...     )
... )

Start grid search algorithm
Fitting 1 folds for each of 8 candidates, totalling 8 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   8 out of   8 | elapsed:   22.4s finished
Best number of classes is 4 according to ICL
Adjusted Rand Index is 0.97891220269305
```

4.3 SparseBM: A command line interface

The **SparseBM** module comes with a command line interface to run the LBM and SBM inference and to generate networks. The SBM/LBM model selection algorithm to chose the best number of classes according to the ICL criterion is available. The command `sparsebm` must be followed by the positional argument `sbm` or `lbm` or `modelselection` or `generate` to use respectively the stochastic block model inference or the latent block model inference or the model selection algorithm or to generate a network with one of these models.

Latent block model: `sparsebm lbm` command line returns a JSON file that contains the two partitions and the estimated parameters of the model. The usage of the command is detailed below:

```
sparsebm lbm --help

usage: sparsebm lbm [-h] [-k1 N_ROW_CLUSTERS] [-k2 N_COLUMN_CLUSTERS]
                    [-o OUTPUT] [-sep SEP] [-niter MAX_ITER] [-ninit N_INIT]
                    [-early N_ITER_EARLY_STOP] [-ninitt N_INIT_TOTAL_RUN]
                    [-t TOL] [-v VERBOSITY] [-gpu USE_GPU] [-idgpu GPU_INDEX]
                    ADJACENCY_MATRIX

optional arguments:
  -h, --help            show this help message and exit
```

```
mandatory arguments:
  ADJACENCY_MATRIX      List of edges in CSV format
  -k1 N_ROW_CLUSTERS, --n_row_clusters N_ROW_CLUSTERS
                        number of row clusters
  -k2 N_COLUMN_CLUSTERS, --n_column_clusters N_COLUMN_CLUSTERS
                        number of row clusters

output:
  -o OUTPUT, --output OUTPUT
                        File path for the json results.

optional arguments:
  -sep SEP, --sep SEP   CSV delimiter to use. Default is ','
  -niter MAX_ITER, --max_iter MAX_ITER
                        Maximum number of EM step
  -ninit N_INIT, --n_init N_INIT
                        Number of initializations that will be run
  -early N_ITER_EARLY_STOP, --n_iter_early_stop N_ITER_EARLY_STOP
                        Number of EM steps to perform for each initialization.
  -ninitt N_INIT_TOTAL_RUN, --n_init_total_run N_INIT_TOTAL_RUN
                        Number of the best initializations that will be run
                        until convergence.
  -t TOL, --tol TOL     Tolerance of likelihood to declare convergence.
  -v VERBOSITY, --verbosity VERBOSITY
                        Degree of verbosity. Scale from 0 (no message
                        displayed) to 3.
  -gpu USE_GPU, --use_gpu USE_GPU
                        Specify if a GPU should be used.
  -idgpu GPU_INDEX, --gpu_index GPU_INDEX
                        Specify the gpu index if needed.
```

Stochastic block model: `sparsebm sbm` command line returns a JSON file that contains the partition and the estimated parameters of the model. A summary of the usage of the command is given:

```
sparsebm sbm --help
```

```
usage: sparsebm sbm [-h] [-sep SEP] [-o OUTPUT] [-k N_CLUSTERS] [-s SYMMETRIC]
                    [-niter MAX_ITER] [-ninit N_INIT]
                    [-early N_ITER_EARLY_STOP] [-ninitt N_INIT_TOTAL_RUN]
                    [-t TOL] [-v VERBOSITY] [-gpu USE_GPU] [-idgpu GPU_INDEX]
                    ADJACENCY_MATRIX
```

Model selection: `sparsebm modelselection sbm` or `sparsebm modelselection lbm` command line returns a JSON file that contains the partition (two if LBM is used) and the estimated parameters of the best model found.

```
sparsebm modelselection --help

usage: sparsebm modelselection [-h] -t TYPE [-gpu USE_GPU] [-idgpu GPU_INDEX]
                               [-s SYMMETRIC] [-p PLOT] [-o OUTPUT]
                               ADJACENCY_MATRIX
```

Graph generation: `sparsebm generate` command line returns a JSON file that contains the partition and a CSV file that contains the adjacency list of the graph. A summary of the usage of the command is given:

```
sparsebm generate --help

positional arguments:
  {sbm,lbm}   model to generate data with
    sbm       use the stochastic block model to generate data
    lbm       use the latent block model to generate data
```

Example: with the two following commands, a network is generated and trained with a SBM using the model selection algorithm:

```
sparsebm generate sbm
sparsebm modelselection edges.csv -t=sbm

---------- START Graph Generation ----------
25 of 25 Generating block:  [100% ]
Groups and params saved in ./groups.json
Edges saved in ./edges.csv
Spliting
         Explore models from 1 classes
         ...
Merging
         Explore models from 5 classes
         ...
Best icl is -53481.1475
Model has been trained successfully.
Value of the Integrated Completed Loglikelihood is -53481.1475
The model selection picked 3 classes
Results saved in results.json
```

5 Experiments and discussion

5.1 Benefit of our inference optimized for sparse graphs.

We compare our inference optimized for sparse graphs to the original inference designed for dense graphs. We provide here experiments on the latent block model only, similar results can be obtained for the stochastic block model.

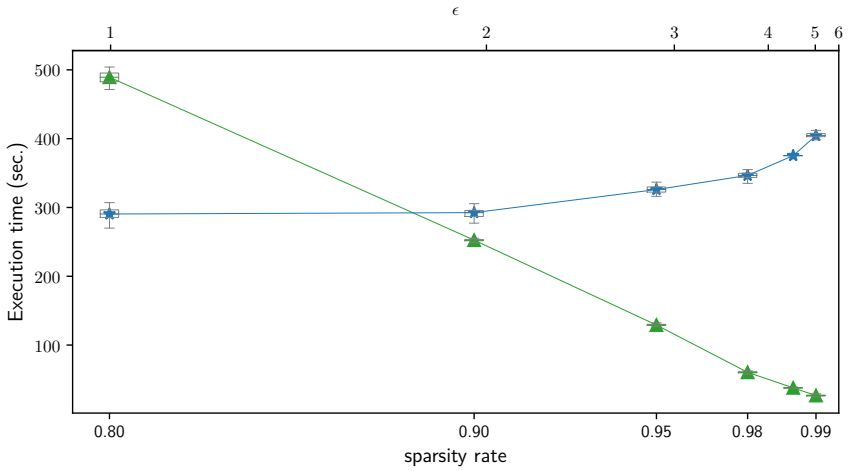


Figure 5: Median computation times for inferring the parameters of a latent block model as a function of the sparsity of the bipartite graph (size $10\,000 \times 5\,000$); ▲ is for the algorithm optimized for sparse graph; ⋆ is for the original algorithm.

5.1.1 Fixed graph size, varying sparsity

A network is generated following an LBM with $n_1 = 10\,000$ nodes of type (1) equally divided in three classes and $n_2 = 5\,000$ nodes of type (2) equally divided in four classes, with parameters

$$\boldsymbol{\alpha} = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\beta} = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\pi} = 2^{-\epsilon} \cdot \begin{pmatrix} 1 & 1/4 & 1/4 & 1/2 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/2 & 1/4 & 1/2 & 1/2 \end{pmatrix}, \tag{5}$$

where $\epsilon \in \{1, \cdots, 6\}$ defines the sparsity level of the graph. For each value of ϵ, a network is generated using these model parameters; the size of the generated networks is fixed, and their sparsity increases with ϵ.

The model parameters are estimated for each network using the original variational inference and the one optimized for sparse graphs (Algorithms 3 and 4, respectively). This process is repeated 100 times for each graph size.

The medians of the computation times are presented in Figure 5 as a function of the sparsity of the graph (that is, one minus the ratio of actual edges to the $n_1 \times n_2$ edges of the complete bipartite graph). The execution times reported here correspond to the overall estimation protocol, that is, (i) 20 EM steps from 100 random initializations, followed by (ii) iterations until convergence of the criterion for the 10 best results reached after these 20 initial steps (see Section 4). The architecture used is a NVIDIA DGX Server with a Tesla V100-SXM2-32GB GPU.

The execution times of the original inference (⋆ in Figure 5) are nearly constant, except for high sparsity levels, where the difficulty of estimation is increased, requiring more EM steps to reach convergence. For our inference optimized for sparse graphs (▲ in Figure 5), the quasi-linear

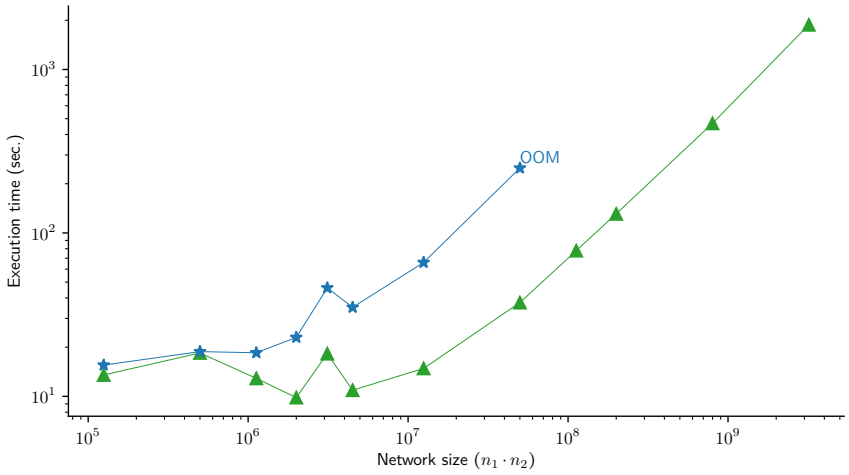


Figure 6: Median computation times for inferring the parameters of a latent block model as a function of the graph size $n_1 \times n_2$ (fixed sparsity rate of 98.76%); ▲ is for the algorithm optimized for sparse graph, ⋆ is for the original algorithm.

trend of execution times in relation to the sparsity rate gives an experimental confirmation of the $\mathcal{O}(\#\{ij : X_{ij} = 1\}k_1k_2)$ computational complexity.

5.1.2 Fixed graph sparsity, varying size

A second series of network is generated following the LBM, using the parameters of the previous experiment, except that ϵ is now fixed to 5, leading to a sparsity rate of 98.76%, and that the sizes of the bipartite graph, n_1 and n_2, vary. The model parameters are estimated for each network using the original variational inference and the one optimized for sparse graphs. The random initialization strategy and the hardware architecture used are as in the previous experiments.

The medians of the computation times are reported in Figure 6 as a function of the size of the bipartite graph $n_1 \times n_2$. Using the original inference (⋆ in Figure 6), the GPU is out of memory (OOM) with graphs bigger than $10\,000 \times 5\,000$ due to the $\mathcal{O}(n_1 n_2)$ memory complexity of the algorithm. The inference implemented in **SparseBM** (▲ in Figure 6) can be applied to much bigger graphs as its memory complexity is in $\mathcal{O}(\#\{ij : X_{ij} = 1\}$ and gets some execution times scaling linearly with the size of the graphs.

5.2 Comparing SparseBM with existing R packages.

We compare the LBM inference from **SparseBM**, **Blockcluster** [Bhatia et al., 2017] and **Blockmodels** [Leger, 2016], using the previous experimental setup with varying graph size.

For a fair comparison between packages, the architecture used is an Intel Xeon Gold 6138 CPU (2.00GHz) with 16 GB RAM (**Blockcluster** and **Blockmodels** are not designed for GPU). Due to this limited computation power, we lighten the previous optimization protocol: we still use 100 random initializations, but they are only updated for 10 EM steps (a single step for **Blockmodels** as

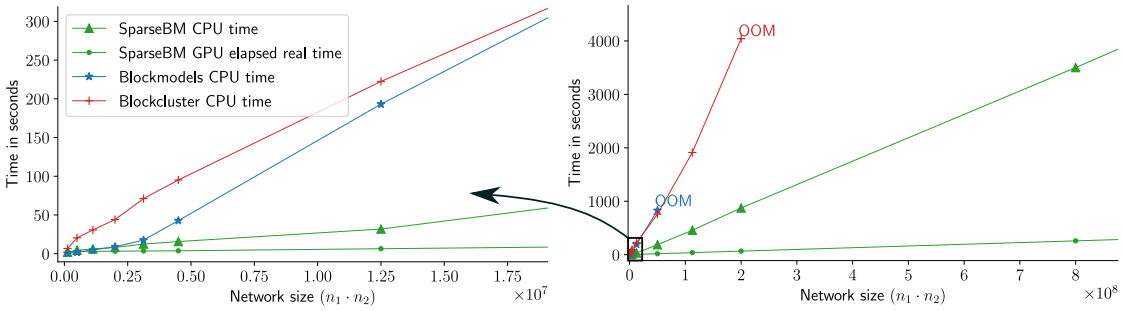


Figure 7: Median computation times (CPU), using existing implementations for inferring the parameters of a latent block model as a function of the graph size $n_1 \times n_2$ with a fixed sparsity rate of 98.76%; ▲ is for **SparseBM**, ⋆ is for **Blockmodels** and + is for **Blockcluster**. The median of the computation real elapsed time using **SparseBM** with GPU is also displayed (•) for reference. The graphic on the left zooms in on smaller networks.

this number is hard-coded); then, only the (single) best initialization is selected to pursue until the convergence of the criterion.

The medians on a hundred repetitions of the execution times are reported in Figure 7. The algorithms from **Blockmodels** and **Blockcluster** are saturating the RAM memory with networks of sizes respectively $(15\,000 \times 7\,500)$ and $(40\,000 \times 20\,000)$, while the implementation of **SparseBM** allows bigger networks as shown in Section 5.1. Note that the limited memory footprint of **SparseBM** provided by the sparse reformulation of the inference is essential to reach the low computation times (real elapsed time) with GPU (• in Figure 7). Indeed, using LBM on large networks would not be possible otherwise due to the very limited memory size available in common GPUs.

We verify that the solutions obtained by the different packages are of comparable accuracy by calculating their similarity with the true generated coclustering. To measure this similarity, we use the coclustering adjusted rand index scores (CoARI) [Robert et al., 2020], whose median values are reported in Table 1. The scores increase for bigger networks as the inference problems gets easier; the scores for **SparseBM** and **Blockcluster** are similar, and we suppose that the poorer performance of **Blockmodels** is mainly due to the lighter initialization procedure.

	CoARI measured with packages		
Network size ($n_1 \cdot n_2$)	**SparseBM**	**Blockcluster**	**Blockmodels**
1.25×10^5	0.05	0.05	0.05
5.00×10^5	0.11	0.11	0.09
1.13×10^6	0.18	0.18	0.12
2.00×10^6	0.26	0.26	0.15
3.13×10^6	0.33	0.32	0.18
4.50×10^6	0.41	0.41	0.20
1.25×10^7	0.68	0.68	0.25
5.00×10^7	0.93	0.93	0.30
1.13×10^8	0.98	0.98	OMM
2.00×10^8	1.00	1.00	OMM

Table 1: Median of the coclustering adjusted rand index (CoARI, a similarity measure between two coclusterings), using existing implementations, as a function of the graph size $n_1 \times n_2$ with a fixed sparsity rate of 98.76%.

6 Conclusion

SparseBM is a Python module for estimating Bernoulli block models in large and sparse networks, relying on the stochastic and latent block models. After a brief review of the mathematical foundations of these models, we present the details of the calculations that are used in this package to reduce the complexity of the original formulation of the variational inference. These computation tricks enable the modeling of large sparse networks for which computational and memory requirements prohibit the use of the original approach.

We present the command line interface and the **Scikit-learn** compatible Python API of the module through examples, and we conduct experiments on synthetic datasets showing that this inference is computationally efficient, enabling to analyze many more networks in a given computation time, and more importantly, much larger sparse networks than the ones that can be handled by current packages. In future releases of the **SparseBM** module, we plan to extend the models to other probability distributions that may result in sparse graphs, such as the zero-inflated Poisson.

References

Julie Aubert, Sophie Schbath, and Stephane Robin. Latent block model for metagenomic data. European Conference on Computational Biology (ECCB 2016), Workshop "Recent Computational Advances in Metagenomics (RCAM)", September 2016. URL `https://hal.archives-ouvertes.fr/hal-01661258`.

Avner Bar-Hen, Pierre Barbillon, and Sophie Donnet. Block models for multipartite networks. applications in ecology and ethnobiology, 2020.

Jean-Patrick Baudry and Gilles Celeux. EM for mixtures. *Statistics and Computing*, 25(4):713–726, 2015. doi: 10.1007/s11222-015-9561-x.

Parmeet Singh Bhatia, Serge Iovleff, and Gérard Govaert. Blockcluster: An r package for model-based co-clustering. *Journal of Statistical Software*, 76(9):1–24, 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i09. URL `https://www.jstatsoft.org/v076/i09`.

Christophe Biernacki, Gilles Celeux, and Gérard Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):719–725, 2000. doi: 10.1109/34.865189.

Christophe Biernacki, Gilles Celeux, and Gérard Govaert. Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. *Computational Statistics & Data Analysis*, 41:561–575, 01 2003. doi: 10.1016/S0167-9473(02)00163-9.

Vincent Brault and Mahendra Mariadassou. Co-clustering through latent bloc model: A review. *Journal de la Société Française de Statistique*, 156(3):120–139, 2015. URL `http://journal-sfds.fr/article/view/474/448`.

G. Celeux and J. Diebolt. The SEM algorithm: A probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Computational Statistics Quarterly*, 2:73–82, 1985.

Marco Corneli, Charles Bouveyron, and Pierre Latouche. Co-clustering of ordinal data via latent continuous random variables and not missing at random entries. *Journal of Computational and Graphical Statistics*, 2020. doi: 10.1080/10618600.2020.1739533. URL `https://hal.archives-ouvertes.fr/hal-01978174`.

Jean-Jacques Daudin, Franck Picard, and Stephane Robin. A mixture model for random graphs. *Statistics and Computing*, 18(2):173–183, 2008. doi: 10.1007/s11222-007-9046-7. URL `https://hal.archives-ouvertes.fr/hal-01197587`.

Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 269–274, aug 2001. doi: 10.1145/502512.502550.

Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 89—-98, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137370. doi: 10.1145/956750.956764.

Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 2006, pages 126–135, 01 2006. doi: 10.1145/1150402.1150420.

Gérard Govaert and Mohamed Nadif. Block clustering with Bernoulli mixture models: Comparison of different approaches. *Computational Statistics & Data Analysis*, 52(6):3233–3245, February 2008. doi: 10.1016/j.csda.2007.09.007.

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL `https://doi.org/10.1145/2827872`.

Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109 – 137, 1983.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, Dec 1985. ISSN 1432-1343. doi: 10.1007/BF01908075.

Tommi S. Jaakkola. Tutorial on variational approximation methods. In Manfred Opper and David Saad, editors, *Advanced Mean Field Methods: Theory and Practice*, pages 129–159. MIT Press, 2000.

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183—-233, November 1999. ISSN 0885-6125. doi: 10.1023/A:1007665907178. URL `https://doi.org/10.1023/A:1007665907178`.

Christine Keribin, Vincent Brault, Gilles Celeux, and Gérard Govaert. Model selection for the binary latent block model. In *Proceedings of COMPSTAT*, 08 2012.

Christine Keribin, Vincent Brault, Gilles Celeux, and Gérard Govaert. Estimation and selection for the latent block model on categorical data. *Statistics and Computing*, 25(6):1201–1216, 2015.

Yuval Kluger, Ronen Basri, Joseph Chang, and Mark Gerstein. Spectral biclustering of microarray data: Coclustering genes and conditions. *Genome Research*, 13:703–716, 05 2003. doi: 10.1101/gr.648603.

Lazhar Labiod and Mohamed Nadif. Co-clustering for binary and categorical data with maximum modularity. In *11th IEEE International Conference on Data Mining (ICDM)*, pages 1140–1145, 2011. doi: 10.1109/ICDM.2011.37.

Pierre Latouche, Etienne Birmelé, and Christophe Ambroise. Overlapping stochastic block models with application to the French political blogosphere. *The Annals of Applied Statistics*, 5(1):309–336, Mar 2011. ISSN 1932-6157. doi: 10.1214/10-aoas382.

Jean-Benoist Leger. Blockmodels: A r-package for estimating in latent block model and stochastic block model, with various probability functions, with or without covariates, 2016.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press. URL `https://projecteuclid.org/euclid.bsmsp/1200512992`.

Catherine Matias and Vincent Miele. Statistical clustering of temporal networks through a dynamic stochastic block model. *Journal of the Royal Statistical Society Series B*, 79(4):1119–1141, 2017. doi: 10.1111/rssb.12200.

Mohamed Nadif and Gérard Govaert. Latent block model for contingency table. *Communications in Statistics—Theory and Methods*, 39(3):416–425, 01 2010. doi: 10.1080/03610920903140197.

Giorgio Parisi. *Statistical Field Theory*. Frontiers in Physics. Addison-Wesley, 1988. URL `https://cds.cern.ch/record/111935`.

Valerie Robert, Yann Vasseur, and Vincent Brault. Comparing high-dimensional partitions with the co-clustering adjusted rand index. *Journal of Classification*, Nov 2020. ISSN 1432-1343. doi: 10.1007/s00357-020-09379-w.

Margot Selosse, Julien Jacques, and Christophe Biernacki. Textual data summarization using the self-organized co-clustering model. *Pattern Recognition*, 103:107315, 2020. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2020.107315. URL `http://www.sciencedirect.com/science/article/pii/S0031320320301199`.

Emilie Shireman, Douglas Steinley, and Michael Brusco. Examining the effect of initialization strategies on the performance of Gaussian mixture modeling. *Behavior Research Methods*, 49, 12 2015. doi: 10.3758/s13428-015-0697-6.

Timothée Tabouy. *Impact of Sampling on Structure Inference in Networks : Application to Seed Exchange Networks and to Ecology*. PhD thesis, Université Paris-Saclay, September 2019. URL `https://tel.archives-ouvertes.fr/tel-02414300`.

Jason Wyse and Nial Friel. Block clustering with collapsed latent block models. *Statistics and Computing*, 22(2):415–428, 2012.