A clustering package for nucleotide sequences using Laplacian Eigenmaps and Gaussian Mixture Model

Marine Bruneau[1,4], Thierry Mottet[2,4], Serge Moulin[2,4,*], Maël Kerbiriou[1,4], Franz Chouly[1,4], Stéphane Chretien[3], Christophe Guyeux[2,4]

[1] *Laboratoire de Mathématiques de Besançon, UMR 6623 CNRS*
[2] *Computer Science Department, FEMTO-ST Institute, UMR 6174 CNRS*
[3] *National Physical Laboratory, Hampton Road, Teddington, United Kingdom*
[4] *Université de Bourgogne Franche-Comté, 16 route de Gray, 25030 Besançon, France*

Abstract

In this article, a new Python package for nucleotide sequences clustering is proposed. This package, freely available on-line, implements a Laplacian eigenmap embedding and a Gaussian Mixture Model for DNA clustering. It takes nucleotide sequences as input, and produces the optimal number of clusters along with a relevant visualization. Despite the fact that we did not optimise the computational speed, our method still performs reasonably well in practice. Our focus was mainly on data analytics and accuracy and as a result, our approach outperforms the state of the art, even in the case of divergent sequences. Furthermore, an a priori knowledge on the number of clusters is not required here. For the sake of illustration, this method is applied on a set of 100 DNA sequences taken from the mitochondrially encoded NADH dehydrogenase 3 (ND3) gene, extracted from a collection of *Platyhelminthes* and *Nematoda* species. The resulting clusters are tightly consistent with the phylogenetic tree computed using a maximum likelihood approach on gene alignment. They are coherent too with the NCBI taxonomy. Further test results based on synthesized data are then provided, showing that the proposed approach is better able to recover the clusters than the most widely used software, namely Cd-hit-est and BLASTClust.

Keywords: DNA clustering, Genomics, Laplacian Eigenmap, Gaussian mixture model.

* *Email address:* `serge.moulin@univ-fcomte.fr` (Serge Moulin)

1. Introduction

As the amount of available genetic sequences increases drastically every year, accurate methods to deeply study them are strongly demanded [1]. Among these methods, clustering is a very powerful tool that helps to understand relations between sequences. It can be used, for instance, to classify 16S RNA sequences into OTUs [2], which are standard proxies for microbial species (clustering is here a way to identify an a priori unknown number of "species"). Clustering is used too to define taxa within groups of species represented by their DNA sequences. Other utilizations of sequence clustering in genomics encompass the study of sub-populations within the same species [3], the discovery of possible hidden variables that can explain differences between such sub-populations, and so on [4].

However, most of the times, only generic methods for clustering a matrix of similarity scores is applied, and methods that are more specific to DNA sequences are still waited. Furthermore, the few existing methods that focus on sequence clustering mainly target on speed and scalability, and they need a strong similarity between sequences to produce accurate clusters. Furthermore, an a priori knowledge on the number of clusters is required, for instance by providing a similarity threshold cutoff. The objective of this article is to provide a new method that relaxes such constrains, allowing the sequences to be really divergent, and returning an a posteriori of the optimal number of clusters in an efficient manner.

One important issue in any clustering procedure is to find an appropriate embedding of the data under study, that will make the respective salient features in each of the groups more clearly delineateable. However, as often stated in the Machine Learning (ML) literature, high dimensional data are often much too scattered in order for an off-the-shelf method to be able to work properly. This phenomenon, often referred as the "curse of dimensionality" [5, 6], explains why several embeddings have been proposed recently. Some of these embeddings are very over-parametrized and can thus only be implemented in the supervised setting. This is the case for methods based on neural networks (*e.g.*, auto-encoders). Other methods need less parameters and are more suitable to unsupervised learning, which is the case of our proposal. Among many nonlinear embedding methods, the Laplacian Eigenmap [7] approach has been extensively studied from both the theoret-

ical and the application viewpoint [8]. More involved methods relying on Semi-Definite programming have also appeared recently with higher separation power in practice than spectral methods, see *e.g.*, [9].

In this article, the aim is to establish the practical efficiency, for DNA sequence clustering, of the combination of a plain Laplacian Eigenmap approach coupled with a Gaussian Mixture based clustering. This particular choice is motivated by its remarkable computational efficiency, even in the case where the objects to classify are really divergent. The overall procedure can be divided in three steps.

1. Compute a similarity matrix between each pair of DNA sequences, *i.e.*, provide a matrix W of size $n \times n$, where n is the number of sequences, which is such that $W_{i,j}$ increases with the "similarity" between sequences number i and j.
2. Diagonalise the Laplacian matrix of W. By such an operation, DNA sequences are mapped to elements of a given vector space, whose dimension is much smaller than the sequence lengths. This reduction of the problem dimension is a key element that usually has a great impact on both visualization and clustering. The combination of the two stages above is often referred as the "Laplacian eigenmap" approach.
3. Cluster the transformed data using a Gaussian Mixture Model (GMM [10]).

By using a ready to use Python package designed at this occasion, and freely available on-line, we have demonstrated the accuracy and efficiency of this approach for DNA sequence clustering. Indeed, the methodology has firstly been tested on a sample of 100 ND3 genes (DNA sequences) from *Platyhelminthes* and *Nematoda* species that have been downloaded from the NCBI website [11]. The classification obtained via this approach has been compared with the phylogenetic tree of these species obtained by a likelihood maximization method using PhyML [12]. Obtained clusters are consistent with both clades appearing in the phylogenetic tree and the NCBI taxonomy. In particular, this clustering perfectly separates the *Nematoda* and *Platyhelminthes* phyla.

To evaluate the method further, extensive simulation experiments have secondly been run on synthesized data: n DNA sequences have been randomly generated, and random mutations and block deletions have been applied on them, leading finally to $N > n$ sequences. The objective was then to group these N sequences within n clusters (n is not a priori known), in such a

way that all the elements in each cluster are originated from the same initial DNA sequence. On this set of synthesized data, our proposal has outperformed the two other state-of-the-art software for DNA clustering, namely Cd-hit-est and BLASTClust.

The remainder of this article is structured as follows. The three stages of the proposed method are detailed in the next section. Numerical results are then presented: the application example involving a real dataset is provided in Section 3.1, while the simulation based procedure is detailed in Section 3.2. A general discussion about the proposed approach is provided in Section 4. This research work ends by a conclusion section, in which the contribution is summarized and intended future work is provided.

2. The Clustering Method

2.1. Laplacian Eigenmap

The so-called Laplacian Eigenmap [7] is an original method for embedding data living in a structured set into a k-dimension vector space. The main information needed to compute the eigenmap is a matrix containing the value of the measured similarity between pairs of data. The main motivation for such embeddings is dimension reduction and visualization. Moreover, spectral methods often exhibit the nice property of separating clusters.

2.1.1. The matrix of similarity

The first step in the construction of a good embedding is the creation of a similarity matrix W. This matrix is intended to measure the similarity between each pair of sequences by providing a number ranging between 0 and 1. The main assumption on W is that the greater the similarity is, the closer are the sequences to each other.

In order to create this similarity matrix, a multiple global alignment of the DNA sequences is first run using the MUSCLE (Multiple Sequence Comparison by Log-Expectation [13]) software. Then, an ad hoc "Needleman Wunsch distance" [14] is computed for each pair of aligned sequence, and with the "EDNAFULL" scoring matrix. This distance takes into account that DNA sequences usually face (1) mutations and (2) insertion/deletion. Note that, by using MUSCLE as first stage of this matrix computation, we operate only one (multiple) sequence alignment, instead of $\frac{n(n-1)}{2}$ (pairwise) alignments in the classical Needleman Wunsch algorithm (that usually contains two stages: finding the best pairwise alignment, and then compute

the edit distance). Introducing Muscle leads to a real acceleration in the construction of the similarity matrix.

Let us call M the distance matrix obtained by this way. M is then divided by the largest distance value, so that all its coefficients are between 0 and 1. W can finally be obtained as follows:

$$\forall\ i, j \in [\![1, n]\!],\ W_{i,j} = 1 - M_{i,j},$$

in such a way that $W_{i,j}$ represents the similarity score between sequences i and j.

2.1.2. Operations on W

Once the similarity matrix has been constructed, the next step is to create the normalized Laplacian matrix, as follows [15]:

$$L = D^{-1/2}(D - W)D^{-1/2},$$

where W is the similarity matrix defined previously and D is the degree matrix of W. That is to say, D is the diagonal matrix defined by:

$$\forall i \in [\![1, n]\!],\ D_{i,i} = \sum_{j=1}^{n} W_{i,j}\ .$$

L being symmetric and real, it is diagonalisable in a basis of pairwise orthogonal eigenvectors $\{\phi_1, ..., \phi_n\}$ associated with eigenvalues $0 = \lambda_1 \leqslant \lambda_2 \leqslant ... \leqslant \lambda_n$. The Laplacian Eigenmap consists in considering the following embedding function:

$$c_{k_1}(i) = \begin{pmatrix} \phi_2(i) \\ \phi_3(i) \\ \vdots \\ \phi_{k_1+1}(i) \end{pmatrix} \in \mathbb{R}^{k_1},$$

where $c_{k_1}(i)$ is the coordinate vector of the point corresponding to the i^{th} sequence. In other words, the coordinate vector of the point corresponding to the i^{th} sequence is composed of the i^{th} coordinate of each of the k_1 first eigenvectors, ordered according to the size of their eigenvalues.

The choice of the k_1 cutoff is a crucial step and one usually proceeds as follows. The ordered eigenvalues are plotted, and we stop at the index where the increase in the eigenvalue is negligible: the number of eigenvalues that are not discarded is k_1. For instance, in our program, we have chosen to set

k_1 as the first time the difference between the k^{th} and $(k+1)^{th}$ value is lower than 0.01.

Note that W can be seen as a weighted adjacency matrix of a graph, where nodes are the DNA sequences while edges are labeled by the degree of affinity between their adjacent nodes. In the literature, the Laplacian matrix is often described as constructed from the weighted adjacency matrix of such a graph rather than constructed from a similarity matrix. These definitions are equivalent.

2.2. Gaussian Mixture based clustering

The final step is performed by applying Gaussian Mixture based clustering (GMM, [10]) to the point cloud. Gaussian Mixture Models belong to the class of unsupervised learning schemes [16], and allow to distribute the data points into different clusters without *a priori* assumption about the clusters interpretation. One of the very useful features of model based clustering is that the model allows to use information criteria in order to estimate the number of clusters using Akaike Information Criterion (AIC [17]), Bayesian Information Criterion (BIC [18]), or Integrated Completed Likelihood (ICL [19]) well-known criteria. The mathematical assumption of a GMM is that the point cloud follows the distribution:

$$\sum_{i=1}^{k_2} \tau_i \, \mathcal{N}(\mu_i, \Sigma_i),$$

where k_2 is the number of clusters, τ_i is the probability for a point to be in cluster i, and $\mathcal{N}(\mu_i, \Sigma_i)$ is the normal distribution of mean μ_i and covariance matrix Σ_i. GMM parameters are computed with the Expectation-Maximization (EM) algorithm [20]. Notice that the EM algorithm may converge to singular distributions exponentially fast [21]. However, degenerate situations can be easily discarded and consistent estimators can be easily obtained in practice. Gaussian Mixture models are still a topic of current extensive research, both from the statistical perspective [22, 23] and the computational one [24].

The Bayesian Information Criterion has been chosen to determine the most relevant number of clusters k_2 to be considered. The BIC, which is a criterion for model selection, is defined as follows:

$$BIC = -2\ln(L) + \ln(n)p,$$

where L is the likelihood of the estimated model, n is the number of observations in the sample, and p is the number of model parameters. This criterion allows us to select a model whose validity is based on a balance between the value of the model's likelihood (fidelity term) and the number of parameters to estimate (complexity term). The likelihood of the model increases with k_2 as well as the number of parameters. The selected model will be by default the one that minimizes this criterion. In the proposed package, the user can also set the number of clusters manually.

2.3. The clustering software

The Python program corresponding to the algorithm described in this section is freely available online[1]. The main function of this package provides a clustering from nucleotide sequences. Its prototype meets the following canvas:

```
clustering = Gclust(liste, nbClusters='BIC', drawgraphs=True,
nbEVMethod = 'delta', nbEVCutOff = 'default',
AddToNamesOfOutputs = ''):
```

where:

- `list` is an in-memory image of a fasta file containing the sequences associated with their names. The fasta file must be configured as the "ND3.fasta" file in our github repository.

- `nbClusters` is the number of clusters desired by the user. By default, the program applies the BIC criterion to determine it. The user may also choose to use the AIC criterion by writing "nbClusters = AIC".

- `drawgraphs` is an optional Boolean value to produce some graphics. If `drawgraphs = TRUE`, a two dimensional clustering of data is plotted (*cf.* Figures 4, 5, and 6), as well as the graphical representation of similarities (as in Figure 1).

- `nbEVMethod` is the method chosen to determine the number of considered eigenvectors k_1. The user can choose between 3 methods, usually reported in the literature.

[1] https://github.com/SergeMOULIN/clustering-tool-for-nucleotide-sequences-using-Laplacian-Eigenmaps-and-Gaussian-Mixture-Models

- If `nbEVMethod = 'delta'` then k_1 is the lowest value such that $\lambda_{k_1+1} - \lambda_{k_1} < \delta$ where δ is a constant to be fixed by the user. `'delta'` is the default method, and with $\delta = 0.01$.
- If `nbEVMethod = 'energy'` then k_1 is the lowest value such that $\sum_{i=1}^{k_1} (\lambda_{max} - \lambda_i) \geq C \times \sum_{i=1}^{n} (\lambda_{max} - \lambda_i)$ where C is a constant to be chosen by the user (default C = 0.9)
- If `nbEVMethod = 'log'` then k_1 is the rounded value of $log(n)$.

- `nbEVCutOff` is the constant δ if `nbEVMethod == 'delta'` or the constant C if `nbEVMethod == 'energy'`. By default, `nbEVCutOff` $= 0.01$ if `nbEVMethod == 'delta'` and `nbEVCutOff` $= 0.9$ if `nbEVMethod == 'energy'`.

- `AddToNamesOfOutputs` is a character string that is inserted as prefix in the output filenames. By default, the file that contains the clustering is named "Clustering.txt" while the one that contains the graphical representation of the similarity matrix is named "similarity_matrix.png". If the user specifies that `AddToNamesOfOutputs = 'Job1'`, then these filenames become "Job1Clustering.txt" and "Job1similarity_matrix.png" respectively.

The `GClust` function can be, for instance, launched as follows within a Python script:

```
list = open('MyFileName.fasta').read()
groups = gclust(list, drawgGraphs=True)
```

The output is a list of k_2 lists, each list containing the references of the sequences grouped in a particular cluster.

2.4. Module and package dependencies

As explained in Section 2.1.1, the DNA sequence alignment software used during the similarity matrix stage is MUSCLE. More precisely, the "muscle_v38" function of `cogent` package has been used [25]. The Gaussian Mixture Model, for its part, is performed using the `GMM` function of `sklearn.mixture` package [26].

3. Numerical evaluations

3.1. Evaluation on real genomic data

The proposed method has been applied to a sample of 100 DNA sequences from the mitochondrial gene ND3 taken from various species of both *Platyhelminthes* and *Nematoda*. Figure 1 graphically displays the similarity matrix that was obtained from the data. As expected, the largest similarities are obtained on the diagonal, that is, when the sequences are compared to themselves. However, blocks seem to appear as well. This justifying the prior intuition that the embedding method separates the clusters if there happens to be more than one in the dataset.

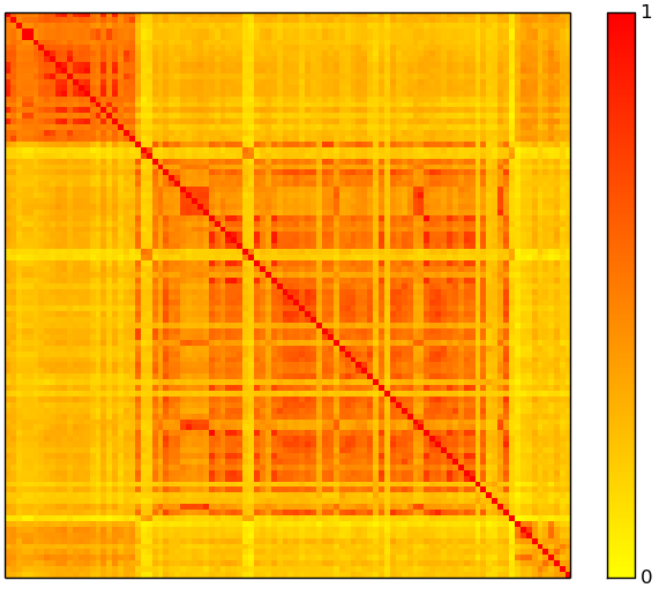

Figure 1: Similarity matrix

Figure 2 displays the 14 first ascending eigenvalues, labeled λ_i, and satisfying $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{14}$. According to the proposed methodology, we have chosen k_1 as the minimal index k such that $\lambda_{k+1} - \lambda_k$ is lower than or equal to 0.01. In this case study, we found $k_1 = 4$.

Figure 3 shows the Bayesian Information Criterion of the Gaussian Mixture Models for various number of clusters. One can see that this BIC reaches

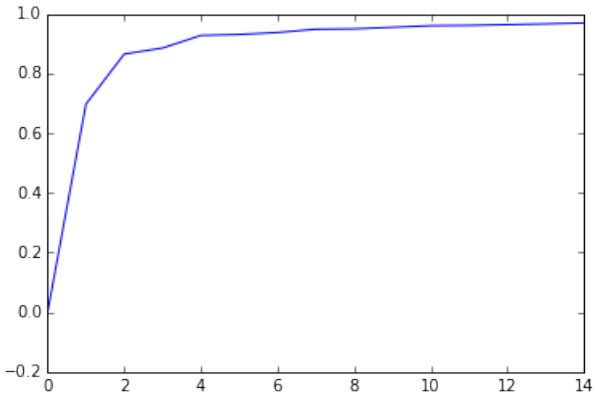

Figure 2: Curve representing the first 14 eigenvalues

its minimum for $k_2 = 4$. Thus, the program automatically partitions the sequence dataset into four clusters.

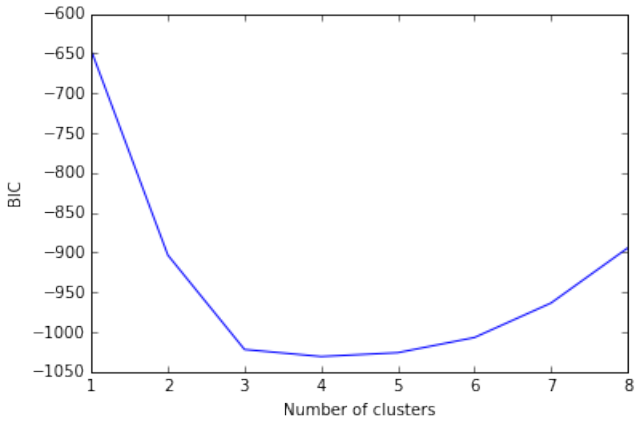

Figure 3: Bayesian Information Criterion of the Gaussian Mixture Models

Figures 4, 5, and 6 represent the point cloud divided in 4 clusters. This point cloud is projected into the plane formed by the first and second eigenvectors in Figure 4, to the one formed by the first and third eigenvectors in Figure 5, and to the plane formed by the second and the third eigenvectors in Figure 6. In these graphs, clusters 0, 1, 2, and 3 are represented in blue, red, yellow and cyan respectively.

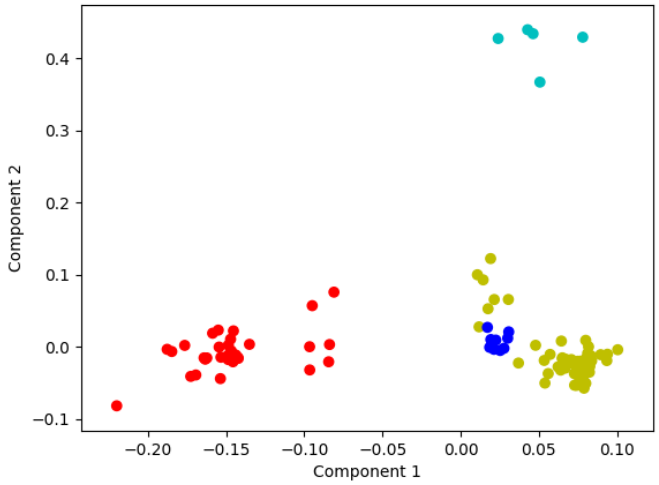

Figure 4: GMM clustering in the plane formed by the eigenvectors 1 and 2

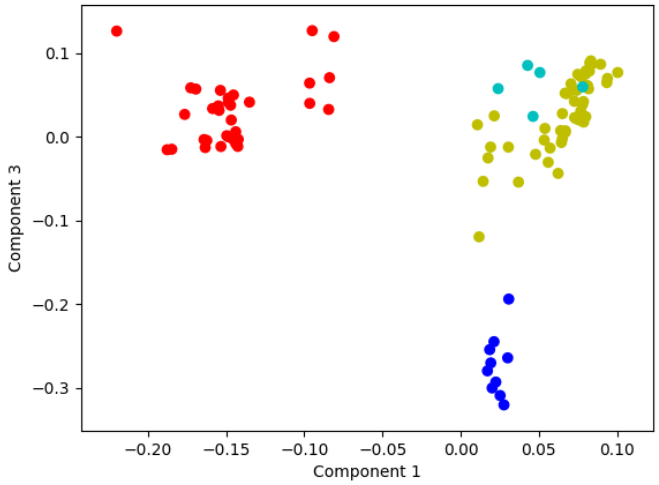

Figure 5: GMM clustering in the plane formed by the eigenvectors 1 and 3

A FASTA file was then written, in which each nucleotide sequence has been labeled according to its taxonomy and its cluster number. The tax-

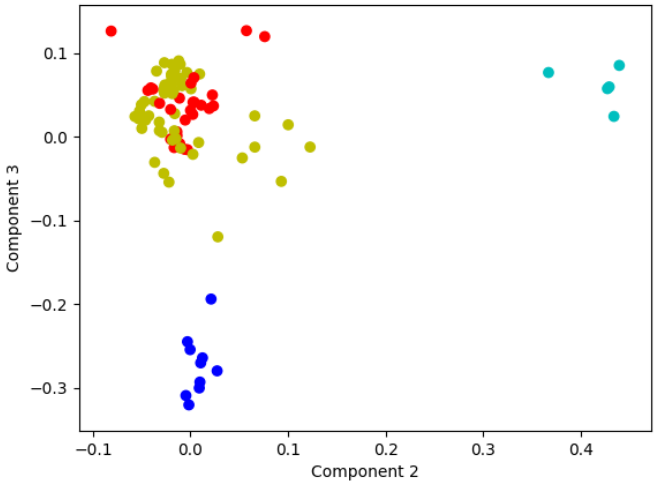

Figure 6: GMM clustering in the plane formed by the eigenvectors 2 and 3

onomies have been found thanks to the `efetch` function (sub-package `Entrez`[2], Python package `Bio`). We then have used the online software *PhyML* (maximum likelihood method for phylogenetic tree reconstruction) with default options [12], to build a tree based on the same mitochondrial ND3 gene that we have used during clustering. The tree, whose leaves contain taxa names and cluster id, has been displayed using *FigTree* software [27]. This tree is depicted in Figures 7 and 8.

The `efetch` function provides 8 taxonomic levels for each species in our sample. In Figures 7 and 8, for readability reasons, we only show the taxonomy between levels 4 and 6. Note first that this clustering perfectly separates *Platyhelminthes* and *Nematoda* phyla. Indeed *Trematoda* and *Cestoda* are two classes of *Platyhelminthes*. Cluster 1 corresponds exactly to the *Platyhelminthes*, while Clusters 0, 2, and 3 represent the *Nematoda*. Cluster 0 is only composed of *Spirurida*, it contains 10 of the 12 members of this taxon. More precisely, when considering the seventh taxonomic level, we found that Cluster 0 contains nine *Filarioidea* and one *Thelazioidea*, while the *Spirurida* that are not in this cluster are a *Dracunculoidea* and a *Physalopteroidea*. Cluster 3, for its part, corresponds exactly with the *Trichocephalida* taxon.

[2]A package that provides code to access NCBI over the world wide web.

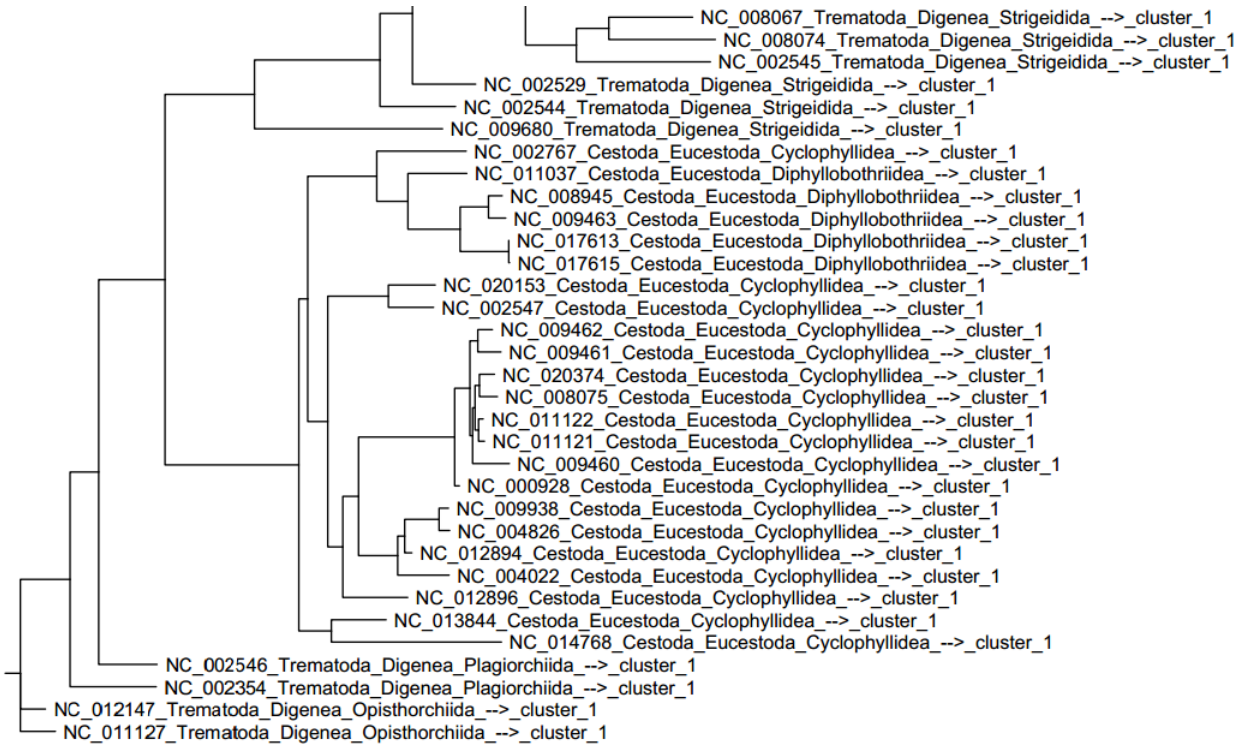

Figure 7: First part of the phylogenetic tree (*Platyhelminthes*)

Finally, in addition to recover taxonomy, the clustering agrees very well with the tree obtained via PhyML, as shown in Figures 7 and 8. Note that the taxonomy, based on morphology and nuclear genome, fully agrees with the PhyML tree and our clustering, which are both based on the mitochondrial genome. This fact suggests that this mitochondrial gene evolves like the nuclear genome.

3.2. Tests on simulated data

In addition to the analysis of real data processed in the previous section, we also performed tests on simulated data. This new series of tests allow us to assess the accuracy of our tool and to compare this accuracy with other existing tools. The data are simulated according to the following steps:

- We generate a "global root". That is to say, a random word of 500 letters {A,T,C,G} representing a sequence of 500 nucleotides.

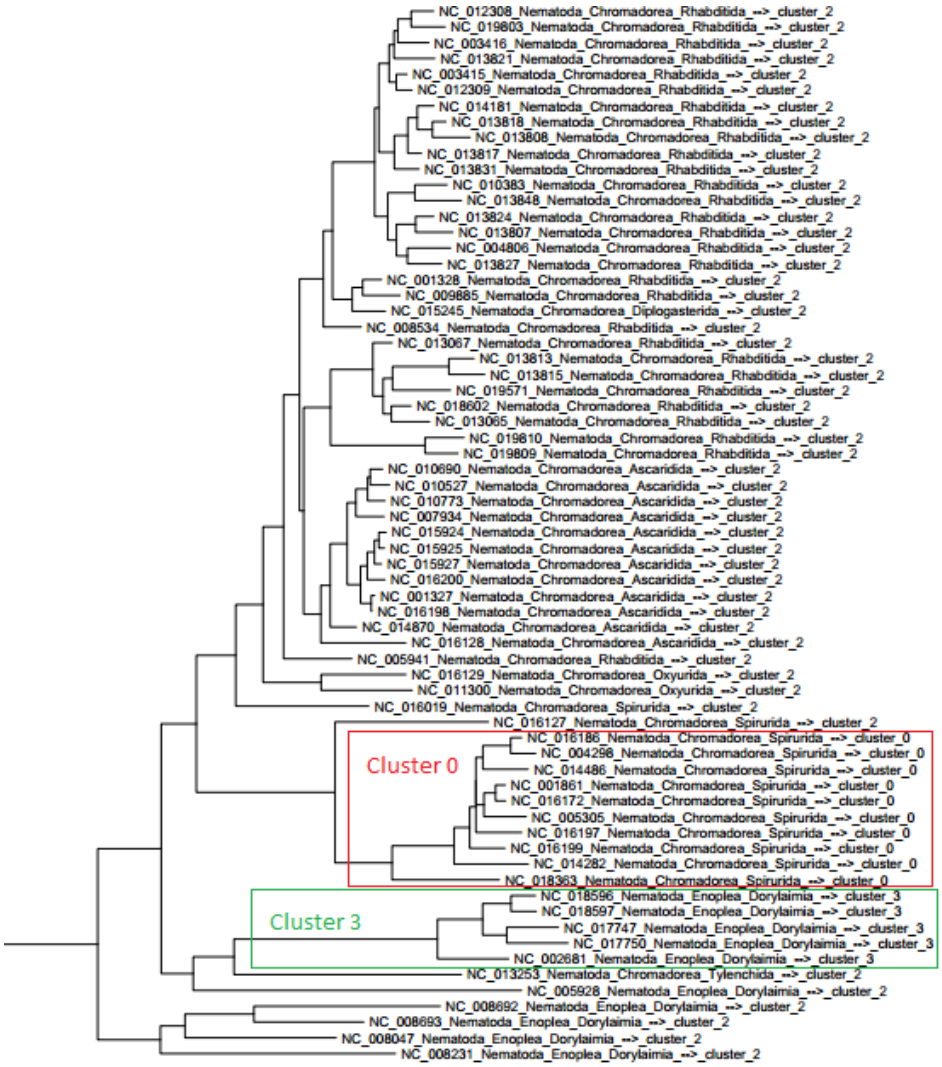

Figure 8: Second part of the phylogenetic tree (*Nematoda*)

- From this global root, we generate 3 "cluster roots" which are each an evolution of the global root for which each nucleotide has mutated with probability 0.05.
- From each cluster root, we generate a cluster of 10 sequences. Each sequence is an evolution of the cluster root for which each nucleotide has mutated with probability 0.05.

- Each of the obtained sequences undergoes a block deletion of 2.5% to 7.5% of its nucleotides with a probability of 0.5. That is to say, for each sequence:
  - A random draw is performed in order to decide on whether a block deletion is carried out or not. The deletion is carried out with probability 0.5.
  - A uniform random draw is performed to determine how many nucleotides are deleted between 2.5% and 7.5% of the sequence size.
  - A uniform random draw is performed in order to decide on where the block deletion starts.

The procedure defined above makes it possible to generate a simulated base of 30 sequences. We repeatedly applied this procedure 25 times with different random seeds (seed = 0 ... 24) in order to simulate 25 different databases. Figure 9 shows the similarity matrix obtained via these simulations (with a random seed of 0).

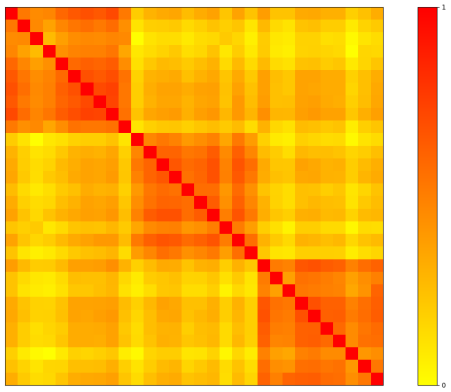

Figure 9: Similarity matrix of the simulated clusters (seed = 0)

The clusters are sorted in each of the simulated databases (as can be seen on the similarity matrix). Thus, the ideal outcome we would like to obtain in applying our tool to these databases is, for example:

[['S11', 'S12', 'S13', 'S14', 'S15', 'S16', 'S17', 'S18', 'S19', 'S20'],
['S21', 'S22', 'S23', 'S24', 'S25', 'S26', 'S27', 'S28', 'S29', 'S30'],
['S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7', 'S8', 'S9', 'S10']]

In order to evaluate the performance of our tool to find the right clustering, we have created a distance that corresponds to the number of sequences that need to move to get the perfect clustering. For instance, the following clustering:

[['S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7', 'S8', 'S9', 'S10'],
['S21', 'S23', 'S24', 'S25', 'S26', 'S27', 'S28', 'S29', 'S30'],
['S11', 'S12', 'S13', 'S14', 'S15', 'S16', 'S17', 'S18', 'S19', 'S20'],
['S22']]

has a distance from the perfect clustering equal to 1.

Table 1 summarizes the distances obtained with our tool for each of the simulated database. This table also summarizes the distance obtained with the "CD-Hit-est" and "BlastClust" tools.

3.2.1. Tests on simulated data with other tools

"CD-Hit-est" and "BLASTClust" are two of the most used clustering tools for nucleotide sequences. According to [28]: "Cd-hit-est is a greedy incremental clustering algorithm. Briefly, the sequences are first sorted in decreasing length. The longest sequence becomes the representative of the first cluster. Then, each remaining sequence is compared with the representatives of the existing clusters. If the similarity with any representative is above a given threshold, it is grouped into that cluster. Otherwise, a new cluster is defined with that sequence as the representative." And, according to [29]: "The program [BLASTClust] begins with pairwise matches and places a sequence in a cluster if the sequence matches at least one sequence already in the cluster. In the case (...) of nucleotide sequences, the Megablast algorithm is used."

In both cases, the user has to provide a similarity threshold. In the case of CD-hit-est, this threshold indicates the minimum similarity that a sequence must have with the reference of a cluster to integrate this cluster. In the case of BlastClust, this threshold indicates the minimum similarity that a sequence must have with at least one other sequence in the cluster. This necessity to specify the threshold for similarity makes a important difference with our tool. Indeed, our tool determines the number of clusters automatically using BIC. Of course, CD-hit-est and BLASTClust provide a default

random seed	Distance our tool	Distance CD-hit-est	Distance BlastClust
0	0	7	3
1	0	11	19
2	2	7	1
3	10	13	20
4	0	8	3
5	0	7	5
6	0	8	2
7	0	7	9
8	0	7	12
9	1	10	19
10	0	6	5
11	0	7	10
12	1	6	5
13	1	10	2
14	0	7	9
15	0	7	5
16	0	6	4
17	0	11	17
18	0	12	16
19	0	6	17
20	0	8	17
21	10	8	17
22	0	7	1
23	0	7	2
24	0	8	6
total	25	201	226

Table 1: Distance from the perfect clustering

value for this similarity (*e.g.*, 0.9 for CD-hit-est) but it is simple to find situations in which this value is not adapted. Indeed, this value does not fit the data as BIC does.

We have not found a way to choose similarity thresholds based only on the simulated databases. Thus, to apply CD-hit-est and BLASTClust to

the simulated data we have deliberately biased our inputs in favor of these programs: we have sought similarity thresholds that minimize the distance to the ideal clustering (as if this ideal clustering was known in advance) in the case of the first simulated database. Then we have applied these thresholds to the 25 simulated bases.

In the case of CD-hit-est, we have tested different similarities between 0.82 and 0.90. Indeed, we have calculated that, given the parameters of the simulations, the ideal threshold should be in this area. The results are shown in Table 2. On can see that the distance to the ideal clustering is minimized for a similarity threshold of 0.84. When studying these CD-hit-est clustering in detail, one can see that, even with this ideal similarity of 0.84, some sequences are isolated and do not fit into their ideal cluster. If the chosen similarity threshold increases, this number of isolated sequences increases. On the other hand, if the chosen similarity threshold decreases (below 0.84), some clusters merge together.

The case of BLASTClust is a little more complicated. In this case, we have varied four parameters detailed below:

- -S <threshold> similarity threshold
  - if <3 then the threshold is set as a BLAST score density (0.0 to 3.0; default = 1.75)
  - if ≥ 3 then the threshold is set as a percent of identical residues (3 to 100)
- -L <threshold> minimum length coverage (0.0 to 1.0; default = 0.9)
- -b <T—F> require coverage as specified by -L and -S on both (T) or only one (F) sequence of a pair (default = TRUE)

There are thus two binary parameters (-b on the one hand and the fact that $S < 3$ or $S \geq 3$ on the other hand) and two continuous parameters (S et L). The two binary parameters constitute 4 configurations ($b =$ 'T 'and $S < 3$ ... $b =$' F' and $S \geq 3$). For each of these 4 cases, we have tested BlastClust on a 10,000 points grid (100 possibilities for $S \times 100$ possibilities for L). We have observed the minimum distance to the ideal clustering obtained on these 40,000 points (it is 3). Then we have counted which of the four configurations contain the largest number of points that minimize the distance to the ideal clustering (it is b = 'F' and $S < 3$). After this, we have selected the point

of this configuration closest to the average of the points that minimize the distance to the ideal clustering among the points that minimize the distance to the ideal clustering. The setting obtained is: $S = 1.91$ (BLAST score density), b = 'F', L = 0.49 and p = 'F' (p = 'F' means that we work on nucleotide sequences, not on proteins).

After we had obtained these parameters for CD-hit-est and BLASTClust, we finally applied these two tools to our 25 simulated datasets. The results are shown in Table 1 with those of our tool.

Similarity Intra-Cluster	0.82	0.83	0.84	0.85	0.86	0.87	0.88	0.89	0.90
Distance to Ideal Clustering	10	8	7	9	9	9	10	10	13

Table 2: Search for the best similarity threshold for CD-hit-est

4. Discussion

4.1. Comparison with other tools

Compared to CD-hit-est and BLASTClust, our tool has several advantages.

1. It uses a statistical criterion (in this case, the reputed BIC) to determine the number of clusters to consider. Thus, it can propose a number of clusters adapted to the database without the user having to provide any *a priori*.
2. As shown in Section 3.2, it allows a better reconstruction of the ideal clustering, even when the similarity threshold is chosen advantageously for CD-hit-est and BLASTClust.
3. It allows the user to plot useful graphical representations of the clustered data.

Finally, we note that CD-hit-est works only for intra-cluster similarities larger than 75%.

One drawback of our tool as compared to CD-hit-est and BLASTClust is the computation speed. According to [30], CD-hit-est is the fastest of these two programs. We have not been able to rigorously confirm this remark in our study, simply because we used BLASTClust on our computers while we used CD-hit-est online.

To sum up, the main features of our tool are different from that of CD-hit-is and BLASTClust. For users wanting to cluster their dataset into meaningful taxa, which makes it possible to apprehend the evolution, our tool might be of greater interest. On the other hand, if the objective is to reduce the size of the dataset by removing duplicates (or retain only one sequence per group of close sequences), it is more appropriate to use one of the other tools with a similarity of 100% (or close to 100%). In particular, CD-hit-est is written so as to provide a representative sequence for each cluster.

4.2. Possible alternatives with the same caneva

Various options are possible to perform the analysis we have presented previously, some of them being listed below.

4.2.1. Similarity matrix

As stated previously, the multiple global alignment step is performed first before computing similarities, using MUSCLE software [13]. Among the most extensively used methods, we can choose MAFFT [31] too, as well as ClustalW or ClustalX [32]. In addition, instead of defining the similarity matrix W as $W_{i,j} = 1 - M_{i,j}$, it could be possible to consider $W_{i,j} = \frac{1}{M_{i,j}}$ or $W_{i,j} = e^{-M_{i,j}}$.

4.2.2. Number of considered eigenvectors

The number of eigenvectors to keep is another point to investigate. As explained in Section 2.1.2, it is usually advised to check graphically when the increase of eigenvalues is reducing. In this article, we have chosen to consider as default proposal: k_1 such that $\delta = \lambda_{k_1+1} - \lambda_{k_1} < 0.01$. This criterion has led to $k = 4$ in the case study, which seems acceptable according to the considered taxonomy.

Some authors in the literature proposed to compute k_1 as the logarithm of n [33]. This method has been implemented as an option, as specified in Section 2.3. In addition, the "energy" method defined in Section 2.3 has been implemented too, in order to propose something similar to what is done in usual Principal Component Analysis (PCA). Other approaches can be considered to solve this problem, which is still an open one.

4.2.3. Number of clusters

We have chosen to consider the BIC [18] to determine the optimal number of clusters, which is a common choice for this type of problem. An alternative

may be to use the Akaike Information Criterion (AIC, [17]). The principle of calculating the AIC is the same as the BIC, since the goal is to maximize log-likelihood penalized by the number of parameters (or more precisely, to minimize the number of parameters to which the log-likelihood is subtracted). AIC formula is the following:

$$AIC = -2ln(L) + 2 \times p,$$

where L is the likelihood of the estimated model and p the number of model parameters.

When $ln(n) \geq 2$, that is to say $n \geq 8$ (which is always the case in practice), BIC penalization is larger than that of AIC. Thus the number of clusters obtained by BIC is lower or equal to the one obtained by AIC. BIC is said to be "more conservative" than AIC. The choice between these two criteria can be dependent on how stringent the clustering is desired. The user of the `GClust` function may choose to use the AIC rather than the BIC as specified in Section 2.3. The user of the `GClust` function is also able to choose the number of clusters of his or her choice.

4.3. Conclusion

In this work, we have proposed a new method of nucleotide sequence clustering. This clustering is produced by a methodology combining Laplacian Eigenmap with Gaussian Mixture models, while the number of clusters is automatically determined by using the Bayesian Information Criterion. The proposed methodology was applied to 100 sequences of mitochondrially encoded NADH dehydrogenase 3. The resulting clusters appeared to be coherent with the phylogeny (gene tree obtained with PhyML) as well as with the NCBI taxonomy. In addition, further tests have also been carried out on fully simulated data. These tests showed that our methodology allows to recover the expected clusters with greater accuracy.

One possible extension for future work could be to investigate more deeply the impact of parameters in the obtained clusters. The effects of using a different similarity matrix, or choosing a different dimension of the image space in Laplacian Eigenmap, or the number of desired clusters, could be investigated. Moreover, our tests on real data allowed us to watch our tool in action while performing a clustering of species into taxons. It might be interesting to also test the ability of our tool to classify 16S RNA sequences into OTUs on real data. Another avenue could consist in adapting the code to the very similar problem of protein clustering.

On a more computer-oriented aspect, our tools could be easier to access by being packaged with pypi. An online tool is also possible. Finally, the graphical interface could also be enhanced, for instance so as to make easier the identification of sequences associated to each point cloud.

[1] B. Valot, C. Guyeux, J. Y. Rolland, K. Mazouzi, X. Bertrand, D. Hocquet, What it takes to be a pseudomonas aeruginosa? the core genome of the opportunistic pathogen updated 10 (5) (2015) e0126468.

[2] X. Hao, R. Jiang, T. Chen, Clustering 16s rRNA for OTU prediction: a method of unsupervised bayesian clustering, Bioinformatics 27 (5) (2011) 611–618.

[3] A. Torroni, T. G. Schurr, C.-C. Yang, E. J. Szathmary, R. C. Williams, M. S. Schanfield, G. A. Troup, W. C. Knowler, D. N. Lawrence, K. M. Weiss, Native american mitochondrial dna analysis indicates that the amerind and the nadene populations were founded by two independent migrations., Genetics 130 (1) (1992) 153–162.

[4] B. E. Suzek, H. Huang, P. McGarvey, R. Mazumder, C. H. Wu, Uniref: comprehensive and non-redundant uniprot reference clusters, Bioinformatics 23 (10) (2007) 1282–1288.

[5] R. Bellman, Dynamic programming, Courier Corporation, 2013.

[6] R. E. Bellman, Adaptive control processes: a guided tour, Princeton university press, 2015.

[7] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering., in: NIPS, Vol. 14, 2001, pp. 585–591.

[8] D. Spielman, Spectral graph theory, Lecture Notes, Yale University (2009) 740–0776.

[9] S. Chrétien, C. Dombry, A. Faivre, A semi-definite programming approach to low dimensional embedding for unsupervised clustering, arXiv preprint arXiv:1606.09190 *.

[10] N. E. Day, Estimating the components of a mixture of normal distributions, Biometrika 56 (3) (1969) 463–474.

[11] National center for biotechnology information, https://www.ncbi.nlm.nih.gov/.

[12] S. Guindon, F. Lethiec, P. Duroux, O. Gascuel, PHYML online-a web server for fast maximum likelihood-based phylogenetic inference, Nucleic acids research 33 (suppl 2) (2005) W557–W559.

[13] R. C. Edgar, MUSCLE: multiple sequence alignment with high accuracy and high throughput, Nucleic acids research 32 (5) (2004) 1792–1797.

[14] S. B. Needleman, C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, Journal of molecular biology 48 (3) (1970) 443–453.

[15] H. Chen, F. Zhang, Resistance distance and the normalized laplacian spectrum, Discrete Applied Mathematics 155 (5) (2007) 654–661.

[16] J. Friedman, T. Hastie, R. Tibshirani, The elements of statistical learning, Vol. 1, Springer series in statistics Springer, Berlin, 2001.

[17] H. Akaike, A new look at the statistical model identification, IEEE transactions on automatic control 19 (6) (1974) 716–723.

[18] G. Schwarz, et al., Estimating the dimension of a model, The annals of statistics 6 (2) (1978) 461–464.

[19] C. Biernacki, G. Celeux, G. Govaert, Assessing a mixture model for clustering with the integrated completed likelihood, IEEE transactions on pattern analysis and machine intelligence 22 (7) (2000) 719–725.

[20] G. McLachlan, D. Peel, Finite mixture models, John Wiley & Sons, 2004.

[21] C. Biernacki, S. Chrétien, Degeneracy in the maximum likelihood estimation of univariate gaussian mixtures with EM, Statistics & probability letters 61 (4) (2003) 373–382.

[22] X. He, D. Cai, Y. Shao, H. Bao, J. Han, Laplacian regularized gaussian mixture model for data clustering, IEEE Transactions on Knowledge and Data Engineering 23 (9) (2011) 1406–1418.

[23] Z. Wang, Q. Gu, Y. Ning, H. Liu, High dimensional expectation-maximization algorithm: Statistical optimization and asymptotic normality, arXiv preprint arXiv:1412.8729.

[24] X. Yi, C. Caramanis, Regularized EM algorithms: A unified framework and statistical guarantees, in: Advances in Neural Information Processing Systems, 2015, pp. 1567–1575.

[25] R. Knight, P. Maxwell, A. Birmingham, J. Carnes, J. G. Caporaso, B. C. Easton, M. Eaton, M. Hamady, H. Lindsay, Z. Liu, et al., Pycogent: a toolkit for making sense from sequence, Genome biology 8 (8) (2007) R171.

[26] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al., Api design for machine learning software: experiences from the scikit-learn project, arXiv preprint arXiv:1309.0238.

[27] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, L. S. Davis, Automatic online tuning for fast gaussian summation, in: Advances in Neural Information Processing Systems (NIPS), 2008.

[28] W. Li, A. Godzik, Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences, Bioinformatics 22 (13) (2006) 1658–1659.

[29] Using blastclust to make non-redundant sequence sets, `https://www.ncbi.nlm.nih.gov/Web/Newsltr/Spring04/blastlab.html`.

[30] Cd-hit user's guide, http://weizhong-lab.ucsd.edu/cd-hit/wiki/doku.php?id=cd-hit_user_guide.

[31] K. Katoh, D. M. Standley, MAFFT multiple sequence alignment software version 7: improvements in performance and usability, Molecular biology and evolution 30 (4) (2013) 772–780.

[32] M. A. Larkin, G. Blackshields, N. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, et al., Clustal w and clustal x version 2.0, bioinformatics 23 (21) (2007) 2947–2948.

[33] C. Matias, Notes de cours: Analyse statistique de graphes (2015).