Graph signal processing for clustering

Nicolas Tremblay

PANAMA Team, INRIA Rennes with Rémi Gribonval,
Signal Processing Laboratory 2, EPFL, Lausanne with Pierre Vandergheynst.
What’s clustering?
Given a series of $N$ objects:

001112223334444
Given a series of $N$ objects:

1/ Find adapted descriptors

\[
0 0 1 1 1 2 2 2 3 3 3 4 4 4 4
\]
Given a series of $N$ objects:

1/ Find adapted descriptors

2/ Cluster
After step 1, one has:

- $N$ vectors in $d$ dimensions (descriptor dimension):

\[ x_1, x_2, \cdots, x_N \in \mathbb{R}^d \]
After step 1, one has:

- $N$ vectors in $d$ dimensions (descriptor dimension):
  \[ x_1, x_2, \ldots, x_N \in \mathbb{R}^d \]

- and their distance matrix $D \in \mathbb{R}^{N \times N}$. 

Conclusion

N. Tremblay

Rennes, 13th of January 2016
After step 1, one has:

- \( N \) vectors in \( d \) dimensions (descriptor dimension):
  
  \[
  x_1, x_2, \cdots, x_N \in \mathbb{R}^d
  \]

- and their distance matrix \( D \in \mathbb{R}^{N \times N} \).

The **goal of clustering** is to assign a label \( c(i) = 1, \cdots, k \) to each object \( i \) in order to **organize / simplify / analyze the data**.
After step 1, one has:

- $N$ vectors in $d$ dimensions (descriptor dimension):
  \[ x_1, x_2, \ldots, x_N \in \mathbb{R}^d \]

- and their *distance matrix* $D \in \mathbb{R}^{N \times N}$.

The *goal of clustering* is to assign a label $c(i) = 1, \ldots, k$ to each object $i$ in order to organize / simplify / analyze the data.

There exists two different general types of methods:

- methods directly based on the $x_i$ and/or $D$ like $k$-means or hierarchical clustering.
After step 1, one has:

- $N$ vectors in $d$ dimensions (descriptor dimension):
  \[ x_1, x_2, \ldots, x_N \in \mathbb{R}^d \]

- and their *distance matrix* $D \in \mathbb{R}^{N \times N}$.

The *goal of clustering* is to assign a label $c(i) = 1, \ldots, k$ to each object $i$ in order to organize / simplify / analyze the data.

There exists two different general types of methods:

- methods directly based on the $x_i$ and/or $D$ like $k$-means or hierarchical clustering.
- graph-based methods.
Graph construction from the distance matrix $D$

Create a graph $\mathcal{G} = (V, E)$:
Graph construction from the distance matrix $D$

Create a graph $\mathcal{G} = (V, E)$:

- each node in $V$ is one of the $N$ objects
Graph construction from the distance matrix $D$

Create a graph $\mathcal{G} = (V, E)$:

- each node in $V$ is one of the $N$ objects
- each pair of nodes $(i, j)$ is connected if the associated distance $D(i, j)$ is small enough.
Graph construction from the distance matrix $D$

Create a graph $G = (V, E)$:

- each node in $V$ is one of the $N$ objects
- each pair of nodes $(i, j)$ is connected if the associated distance $D(i, j)$ is small enough.

For example, two connectivity possibilities:

- **Gaussian kernel**:
  1. all pairs of nodes are connected with links of weights $\exp(-D(i, j)/\sigma)$
  2. remove all links of weight inferior to $\epsilon$
Graph construction from the distance matrix $D$

Create a graph $\mathcal{G} = (V, E)$:

- each node in $V$ is one of the $N$ objects
- each pair of nodes $(i, j)$ is connected if the associated distance $D(i, j)$ is small enough.

For example, two connectivity possibilities:

- Gaussian kernel:
  1. all pairs of nodes are connected with links of weights $\exp(-D(i, j)/\sigma)$
  2. remove all links of weight inferior to $\epsilon$

- $k$ nearest neighbors: connect each node to its $k$ nearest neighbors.
The problem now states:

Given the graph $G$ representing the similarity between the $N$ objects, find a partition of all nodes in $k$ clusters.
The problem now states:
Given the graph $G$ representing the similarity between the $N$ objects, find a partition of all nodes in $k$ clusters.

Many methods exist [Fortunato '10]:

- Modularity (or other cost-function) optimisation methods [Newman '04]
- Random walk methods [Delvenne '10]
- Methods inspired from statistical physics [Krzakala '12], information theory [Rosvall '07]...
- spectral methods
- ...
Three useful matrices

The adjacency matrix:

\[
W = \begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

The degree matrix:

\[
S = \begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

The Laplacian matrix:

\[
L = S - W = \begin{bmatrix}
2 & -1 & -1 & 0 \\
-1 & 3 & -1 & -1 \\
-1 & -1 & 2 & 0 \\
0 & -1 & 0 & 1 \\
\end{bmatrix}
\]
Three useful matrices

The adjacency matrix:

\[ W = \begin{bmatrix} 0 & .5 & .5 & 0 \\ .5 & 0 & .5 & 4 \\ .5 & .5 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{bmatrix} \]

The degree matrix:

\[ S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \]

The Laplacian matrix:

\[ L = S - W = \begin{bmatrix} 1 & -.5 & -.5 & 0 \\ -.5 & 5 & -.5 & -4 \\ -.5 & -.5 & 1 & 0 \\ 0 & -4 & 0 & 4 \end{bmatrix} \]
The classical spectral clustering algorithm [Von Luxburg ’06]:

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Compute:

$$U_k = (u_1 | u_2 | \cdots | u_k)$$

the first $k$ eigenvectors of $L = S - W$. 

The classical spectral clustering algorithm [Von Luxburg '06] : 

Given the $N$-node graph $G$ of adjacency matrix $W$ :

1. Compute :

$$U_k = (u_1 | u_2 | \cdots | u_k)$$

the first $k$ eigenvectors of $L = S - W$.

2. Consider each node $i$ as a point in $\mathbb{R}^k$ :

$$f_i = U_k^T \delta_i.$$
The classical spectral clustering algorithm [Von Luxburg '06]:

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Compute:
   \[ U_k = (u_1 | u_2 | \cdots | u_k) \]
   the first $k$ eigenvectors of $L = S - W$.

2. Consider each node $i$ as a point in $\mathbb{R}^k$:
   \[ f_i = U_k^T \delta_i. \]

3. Run $k$-means with the Euclidean distance:
   \[ D_{ij} = ||f_i - f_j|| \]
   and obtain $k$ clusters.
What’s the point of using a graph?

N points in $d = 2$ dimensions. Result with $k$-means ($k=2$):

After creating a graph from the N points’ interdistances, and running the spectral clustering algorithm (with $k=2$):
Computation bottlenecks of the spectral clustering algorithm

When $N$ and/or $k$ become too large, there are two main bottlenecks in the algorithm:

1. The partial eigendecomposition of the Laplacian.
When $N$ and/or $k$ become too large, there are two main bottlenecks in the algorithm:

1. The partial eigendecomposition of the Laplacian.
2. $k$-means.

Our goal: Circumvent both!
Computation bottlenecks of the spectral clustering algorithm

When \( N \) and/or \( k \) become too large, there are two main bottlenecks in the algorithm:

1. The partial eigendecomposition of the Laplacian.
2. \( k \)-means.

Our goal:

Circumvent both!
What’s graph signal processing?
What’s a graph signal?

Graph signal processing... applied to clustering

N. Tremblay  
Graph signal processing for clustering

Rennes, 13th of January 2016 11 / 26
What’s a graph signal?

- Graph signal processing...
- ... applied to clustering
- Conclusion
What’s a graph signal?

![Graph Signal Example]

- **C** vs. **Month**
- **C** vs. **Heure**

---

N. Tremblay
Graph signal processing for clustering
Rennes, 13th of January 2016
What’s a graph signal?
What’s a graph signal?
What’s a graph signal?
What’s the graph Fourier matrix?

[Hammond ’11]

The “classical” graph:

\[
L_{cl} = \begin{bmatrix}
2 & -1 & 0 & \cdots & 0 & -1 \\
-1 & 2 & -1 & \cdots & 0 & 0 \\
0 & 0 & 0 & \cdots & 2 & -1 \\
-1 & 0 & 0 & \cdots & -1 & 2 \\
\end{bmatrix}
\]

All classical Fourier modes are the eigenvectors of \( L_{cl} \)
What’s the graph Fourier matrix?

[Hammond ’11]

The “classical” graph:

Any graph:

\[ L_{cl} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ -1 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix} \]

All classical Fourier modes are the eigenvectors of \( L_{cl} \)

By analogy, any graph’s Fourier modes are the eigenvectors of its Laplacian matrix \( L \).
The graph Fourier matrix

\[ L = S - W \]

Its eigenvectors:

\[ U = (u_1 | u_2 | \cdots | u_N) \]

form the graph Fourier orthonormal basis.

Its eigenvalues:

\[ 0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N \]

represent the graph frequencies. \( \lambda_i \) is the squared frequency associated to the Fourier mode \( u_i \).
Illustration

Low frequency:

High frequency:
The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size $N$. 
The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size $N$.
- $\hat{f}$ is obtained by decomposing $f$ on the eigenvectors $u_i$:

$$
\hat{f} = \begin{pmatrix}
\langle u_1, f \rangle \\
\langle u_2, f \rangle \\
\langle u_3, f \rangle \\
\vdots \\
\langle u_N, f \rangle 
\end{pmatrix}, \text{ i.e. } \hat{f} = U^T f
$$
The Fourier transform

- given \( f \in \mathbb{R}^N \) a signal on a graph of size \( N \).
- \( \hat{f} \) is obtained by decomposing \( f \) on the eigenvectors \( u_i \):

\[
\hat{f} = \left( \begin{array}{c}
\langle u_1, f \rangle \\
\langle u_2, f \rangle \\
\langle u_3, f \rangle \\
\vdots \\
\langle u_N, f \rangle
\end{array} \right), \text{ i.e. } \hat{f} = U^T f
\]

- Inversely, the inverse Fourier transform reads:

\[
f = U \hat{f}
\]
The Fourier transform

- given $f \in \mathbb{R}^N$ a signal on a graph of size $N$.
- $\hat{f}$ is obtained by decomposing $f$ on the eigenvectors $u_i$:

$$
\hat{f} = \begin{pmatrix}
< u_1, f > \\
< u_2, f > \\
< u_3, f > \\
... \\
< u_N, f > 
\end{pmatrix}, \text{ i.e. } \hat{f} = U^T f
$$

- Inversely, the inverse Fourier transform reads:

$$
f = U \hat{f}
$$

- The Parseval theorem stays valid: $\forall (g, h) \quad < g, h > = < \hat{g}, \hat{h} >$
Filtering

Given a filter function $g$ defined in the Fourier space.

![Graph of a filter function $g(\lambda)$]
Filtering

Given a filter function $g$ defined in the Fourier space.

In the Fourier space, the signal filtered by $g$ reads:

$$\hat{f}g = \begin{pmatrix} \hat{f}(1)g(\lambda_1) \\ \hat{f}(2)g(\lambda_2) \\ \hat{f}(3)g(\lambda_3) \\ \vdots \\ \hat{f}(N)g(\lambda_N) \end{pmatrix} = \hat{G} \hat{f} \text{ with } \hat{G} = \begin{pmatrix} g(\lambda_1) & 0 & 0 & \cdots & 0 \\ 0 & g(\lambda_2) & 0 & \cdots & 0 \\ 0 & 0 & g(\lambda_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & g(\lambda_N) \end{pmatrix}$$
Filtering

Given a filter function \( g \) defined in the Fourier space.

In the Fourier space, the signal filtered by \( g \) reads:

\[
\hat{f}^g = \begin{pmatrix}
\hat{f}(1) g(\lambda_1) \\
\hat{f}(2) g(\lambda_2) \\
\hat{f}(3) g(\lambda_3) \\
\vdots \\
\hat{f}(N) g(\lambda_N)
\end{pmatrix} = \hat{G} \hat{f}
\]

with

\[
\hat{G} = \begin{pmatrix}
g(\lambda_1) & 0 & 0 & \cdots & 0 \\
0 & g(\lambda_2) & 0 & \cdots & 0 \\
0 & 0 & g(\lambda_3) & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & g(\lambda_N)
\end{pmatrix}
\]

In the node space, the filtered signal \( f^g \) reads therefore:

\[
f^g = U \hat{G} U^\top f = Gf
\]
So where’s the link?
Remember: the classical spectral clustering algorithm

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Compute: $$U_k = (u_1 | u_2 | \cdots | u_k)$$ the first $k$ eigenvectors of $L = S - W$.

2. Consider each node $i$ as a point in $\mathbb{R}^k$: $$f_i = U_k^\top \delta_i.$$ 

3. Run $k$-means with the Euclidean distance: $$D_{ij} = ||f_i - f_j||$$ and obtain $k$ clusters.
Remember : the classical spectral clustering algorithm

Given the $N$-node graph $G$ of adjacency matrix $W$ :

1. Compute :

$$U_k = (u_1|u_2|\cdots|u_k)$$

the first $k$ eigenvectors of $L = S - W$.

2. Consider each node $i$ as a point in $\mathbb{R}^k$ :

$$f_i = U_k^\top \delta_i.$$  

3. Run $k$-means with the Euclidean distance : $D_{ij} = ||f_i - f_j||$ and obtain $k$ clusters.

Let’s work on the first bottleneck : estimate $D_{ij}$ without partially diagonalizing the Laplacian matrix.
Ideal low-pass filtering

1st step : assume we know $U_k$ and $\lambda_k$

Given $h_{\lambda_k}$ an ideal LP, $H_{\lambda_k} = U H_{\lambda_k} U^\top = U_k U_k^\top$ is its filter matrix.
Ideal low-pass filtering

1st step : assume we know $U_k$ and $\lambda_k$

Given $h_{\lambda_k}$ an ideal LP, $H_{\lambda_k} = U H_{\lambda_k} U^\top = U_k U_k^\top$ is its filter matrix.

Let $R = (r_1 | r_2 | \cdots | r_{\eta}) \in \mathbb{R}^{N \times \eta}$ be a random Gaussian matrix. We define $\tilde{f}_i = (H_{\lambda_k} R)^\top \delta_i \in \mathbb{R}^\eta$ and $\tilde{D}_{ij} = \| \tilde{f}_i - \tilde{f}_j \|$. 
Ideal low-pass filtering

1st step : assume we know $U_k$ and $\lambda_k$

Given $h_{\lambda_k}$ an ideal LP,

$$H_{\lambda_k} = U H_{\lambda_k} U^\top = U_k U_k^\top$$

is its filter matrix.

Let $R = \begin{pmatrix} r_1 & r_2 & \cdots & r_\eta \end{pmatrix} \in \mathbb{R}^{N \times \eta}$ be a random Gaussian matrix.

We define $\tilde{f}_i = (H_{\lambda_k} R)^\top \delta_i \in \mathbb{R}^\eta$ and $\tilde{D}_{ij} = \| \tilde{f}_i - \tilde{f}_j \|$. 

Norm conservation theorem for ideal filter

Let $\epsilon > 0$, if $\eta > \eta_0 \sim \frac{\log N}{\epsilon^2}$, then, with proba $> 1 - 1/N$, we have :

$$\forall (i, j) \in [1, N]^2 \quad (1 - \epsilon) D_{ij} \leq \tilde{D}_{ij} \leq (1 + \epsilon) D_{ij}.$$
Non-ideal low-pass filtering
2nd step: assume all we know is $\lambda_k$

In practice, we use a poly approx of order $m$ of $h_{\lambda_k}$:

$$\tilde{h}_{\lambda_k} = \sum_{l=1}^{m} \alpha_l \lambda^l \simeq h_{\lambda_k}.$$

- Does not require the knowledge of $U_k$.
- Only involves matrix-vector multiplications [costs $O(m|E|)$].
Non-ideal low-pass filtering

2nd step: assume all we know is $\lambda_k$

In practice, we use a poly approx of order $m$ of $h_{\lambda_k}$:

$$\tilde{h}_{\lambda_k} = \sum_{l=1}^{m} \alpha_l \lambda^l \simeq h_{\lambda_k}.$$  

Indeed, in this case, filtering a vector $x$ reads:

$$\tilde{H}_{\lambda_k} x = U\tilde{h}_{\lambda_k}(\Lambda)U^\top x = U \sum_{l=1}^{m} \alpha_l \Lambda^l U^\top x = \sum_{l=1}^{m} \alpha_l L^l x$$

- Does not require the knowledge of $U_k$.
- Only involves matrix-vector multiplications [costs $O(m|E|)$].
Non-ideal low-pass filtering
2nd step : assume all we know is $\lambda_k$

In practice, we use a poly approx of order $m$ of $h_{\lambda_k}$:

$$\tilde{h}_{\lambda_k} = \sum_{l=1}^{m} \alpha_l \lambda^l \simeq h_{\lambda_k}.$$  

Indeed, in this case, filtering a vector $x$ reads:

$$\tilde{H}_{\lambda_k} x = U \tilde{h}_{\lambda_k} (\Lambda) U^\top x = U \sum_{l=1}^{m} \alpha_l \Lambda^l U^\top x = \sum_{l=1}^{m} \alpha_l L^l x$$

- Does not require the knowledge of $U_k$.
- Only involves matrix-vector multiplications [costs $O(m|E|)$].

The theorem stays (more or less) valid with this non-ideal filtering!
Goal : given L, estimate its $k$-th eigenvalue as fast as possible.
Last step: estimate $\lambda_k$

Goal: given $L$, estimate its $k$-th eigenvalue as fast as possible.

We use eigencount techniques (also based on polynomial filtering of random vectors!):

- given the interval $[0, b]$, get an approximation of the number of enclosed eigenvalues.
- And find $\lambda_k$ by dichotomy on $b$. 
Accelerated spectral algorithm

Given the $N$-node graph $\mathcal{G}$ of adjacency matrix $W$:

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$.
2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{H}_{\lambda_k}$ and treat each node $i$ as a point in $\mathbb{R}^\eta$:
   \[ \tilde{f}^\top_i = \delta^\top_i \tilde{H}_{\lambda_k} R. \]
4. Run $k$-means with the Euclidean distance:
   \[ \tilde{D}_{ij} = ||\tilde{f}_i - \tilde{f}_j|| \]
   and obtain $k$ clusters.

Let's work on the second bottleneck: avoid $k$-means in possibly very large dimension ($N$-step 4).
Accelerated spectral algorithm

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$. 

2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.

3. Filter them with $\tilde{H}_{\lambda_k}$ and treat each node $i$ as a point in $\mathbb{R}^\eta$:
$$\tilde{f}^\top_i = \delta^\top_i \tilde{H}_{\lambda_k} R.$$

4. Run $k$-means with the Euclidean distance $\tilde{D}_{ij} = ||\tilde{f}_i - \tilde{f}_j||$ and obtain $k$ clusters.

Let's work on the second bottleneck: avoid $k$-means in possibly very large dimension $N$ (step 4).
Accelerated spectral algorithm

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$.
2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.

Let's work on the second bottleneck: avoid $k$-means in possibly very large dimension $N$ (step 4).
Accelerated spectral algorithm

Given the $N$-node graph $G$ of adjacency matrix $W$ :

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$.
2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{H}_{\lambda_k}$ and treat each node $i$ as a point in $\mathbb{R}^\eta$ :

$$\tilde{f}_i^T = \delta_i^T \tilde{H}_{\lambda_k} R.$$
Accelerated spectral algorithm

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$.
2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{H}_{\lambda_k}$ and treat each node $i$ as a point in $\mathbb{R}^\eta$:
   $$\tilde{f}_i^\top = \delta_i^\top \tilde{H}_{\lambda_k} R.$$  
4. Run $k$-means with the Euclidean distance: $\tilde{D}_{ij} = ||\tilde{f}_i - \tilde{f}_j||$ and obtain $k$ clusters.
Accelerated spectral algorithm

Given the $N$-node graph $\mathcal{G}$ of adjacency matrix $W$:

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$.
2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{H}_{\lambda_k}$ and treat each node $i$ as a point in $\mathbb{R}^\eta$:
   \[
   \tilde{f}_i^\top = \delta_i^\top \tilde{H}_{\lambda_k} R.
   \]
4. Run $k$-means with the Euclidean distance: $\tilde{D}_{ij} = ||\tilde{f}_i - \tilde{f}_j||$
   and obtain $k$ clusters.

Let’s work on the second bottleneck: avoid $k$-means in possibly very large dimension $N$ (step 4).
Fast spectral algorithm?

Given the $N$-node graph $G$ of adjacency matrix $W$:

1. Estimate $\lambda_k$, the $k$-th eigenvalue of $L$.
2. Generate $\eta$ random graph signals in matrix $R \in \mathbb{R}^{N \times \eta}$.
3. Filter them with $\tilde{H}_{\lambda_k}$ and treat each node $i$ as a point in $\mathbb{R}^\eta$:

$$\tilde{f}_i^\top = \delta_i^\top \tilde{H}_{\lambda_k} R.$$

4. Sample randomly $\rho \approx k \log k << N$ nodes out of $N$:

$$\tilde{f}_i^r = M\tilde{f}_i = (M\tilde{H}_{\lambda_k} R)^\top \delta_i^r.$$

5. Run $k$-means in this reduced space with the Euclidean distance:

$$\bar{D}_{ij}^r = \|\tilde{f}_i^r - \tilde{f}_j^r\|$$

and obtain $k$ clusters.

6. Interpolate cluster indicator functions $c_i^r$ on the whole graph:

$$\tilde{c}_l = \arg\min_{x \in \mathbb{R}^N} \|Mx - c_i^r\|^2 + \mu x L^\top x.$$
Compressive spectral clustering: a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $G$;
Compressive spectral clustering : a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $G$ ;
2. subsample the set of nodes ;
Compressive spectral clustering: a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $\mathcal{G}$;
2. subsample the set of nodes;
3. cluster the reduced set of nodes;
Compressive spectral clustering : a summary

1. generate a feature vector for each node by filtering few random gaussian random signal on $G$ ;
2. subsample the set of nodes ;
3. cluster the reduced set of nodes ;
4. interpolate the cluster indicator vectors back to the complete graph.
This work was done in collaboration with:

- Gilles Puy and Rémi Gribonval from the PANAMA team (INRIA).
- Pierre Vandergheynst from EPFL.
This work was done in collaboration with:

- **Gilles Puy** and **Rémi Gribonval** from the PANAMA team (INRIA).
- **Pierre Vandergheynst** from EPFL.

Part of this work has been published (or submitted):

- Circumventing the first bottleneck has been accepted to ICASSP 2016
- Interpolation of $k$-bandlimited graph signals has been submitted to ACHA in November (an application of which helps us circumvent the second bottleneck).
Perspectives and difficult questions

Two difficult questions (among others):

1. Given a semi-definite positive matrix, how to estimate as fast as possible its $k$-th eigenvalue, and only that one?
2. How to subsample $\rho$ nodes out of $N$ while ensuring that clustering them in $k$ classes is the result one would have obtained by clustering all $N$ nodes?
Perspectives and difficult questions

Two difficult questions (among others):

1. Given a semi-definite positive matrix, how to estimate as fast as possible its $k$-th eigenvalue, and only that one?
2. How to subsample $\rho$ nodes out of $N$ while ensuring that clustering them in $k$ classes is the result one would have obtained by clustering all $N$ nodes?

Perspectives

1. How about if nodes are added one by one?
2. Rational filters instead of polynomial filters?
3. Approximating other spectral clustering algorithms?