Graph Sampling with Determiantal Point Processes

Quentin CHAN-WAI-NAM
MVA

Guillaume GAUTIER (tutor)
INRIA

Abstract

A great variety of data is originally structured as networks, or graphs. Data defined on the nodes of a given graph is called a graph signal. Here, we are interested in some results of the sampling theory, that state it is possible under some assumptions to reconstruct a graph signal given its values on only a node sample of size much smaller than the entire graph. We focus here on graph sampling using determinantal point processes, which are very useful tools to choose the nodes on which to sample the signal. This project is mainly focused on implementation and experimentation using results from [5].

1 Theory

1.1 Frequencies on graphs

The goal of the sampling theory on graphs [4] is similar to the goal of the Fourier sampling theory on classical signals: we try to decompose a given signal on some frequencies and use this decomposition to build a much compressed representation of the signal. Given a graph with N nodes, we define the Laplacian matrix as:

$$L = D - W$$

where $W \in \mathbb{R}^{N\times N}$ is the matrix of the edges of the graph such that $W_{i,j} \geq 0$ is the weight of the edge between nodes i and j, and $D \in \mathbb{R}^{N\times N}$ is diagonal with $D_i = \sum_j W_{i,j}$. Let (Λ, U) be the eigendecomposition of L such that $\Lambda = (\lambda_1, \cdots, \lambda_N)^\top$, $0 = \lambda_1 \leq \cdots \leq \lambda_N$ and $U = (u_1, \cdots, u_N)$ are the associated eigenvectors. We say that $U_k = (u_1, \cdots, u_k)$ are the k first low-frequency eigenmodes and any linear combination of $u_1, \cdots, u_k$ is called a k-bandlimited signal. Alternatively, x is a k-bandlimited signal iff

$$\exists\, \alpha \in \mathbb{R}^k, \qquad x = U_k \alpha. \tag{1}$$

1.2 Sampling on graphs and signal reconstruction

Sampling consists in choosing m nodes among N and measuring the signal on these nodes. If $\mathcal{A} = (\omega_1, \cdots, \omega_m)$ is the subset of the chosen nodes, $M \in R^{m\times N}$ is the measurement matrix such that $M_{i,j} = \delta_{j=\omega_i}$. Then a measurement of the signal x on the sample $\mathcal{A}$ writes

$$y = Mx + n \tag{2}$$

where $n \in \mathbb{R}^m$ is a measurement noise.

One can show that if x is a k-bandlimited signal, then one can hope to reconstruct x if $m \geq k$. The difficulty of the sampling theory is, given a measurement budget m, to find $\mathcal{A}$ such that $|\mathcal{A}| = m$ and we can invert (2) so as to reconstruct exactly x from y. A sufficient condition for this to be true if $n = 0$ is that

$$MU_k \text{ has its smallest singular value } \sigma_1 > 0. \tag{3}$$

1.3 Reconstruction with determinantal point processes on graphs

The *determinantal point processes* (DPP) [2] are tools that are very useful in this situation, since they enable to sample points in $\{1, \cdots, N\}$ with some negative correlation between the occurences of points that are "close" to each other, that is "close" points will be less likely to be sampled together. In other terms, sampling from a DPP ensures the diversity of the sample.

In our case, if $\{1, \cdots, N\}$ are the nodes of the graph, the random variable $\mathcal{A}$ from a DPP of *marginal kernel* $K \in \mathbb{R}^{N \times N}$ verifies

$$\forall \mathcal{S} \subset \mathcal{A}, \qquad \mathbb{P}(\mathcal{S} \subset \mathcal{A}) = \det(K_{\mathcal{S}}) \tag{4}$$

where $K_{\mathcal{S}}$ is the submatrix of K with lines and columns indices in $\mathcal{S}$.

When sampling the measurement nodes, if x_1 and x_2 are two graph signals, we want to ensure that $\|x_1 - x_2\| > 0 \implies \|y_1 - y_2\| > 0$. One way to ensure this is to reweight the measurement with a given matrix such that the norm of the reweighted measurement is close to the norm of the original vector. More specifically, if $\mathcal{A} = (\omega_1, \cdots, \omega_m)$, we define $P = \operatorname{diag}(\pi_{\omega_1}, \cdots, \pi_{\omega_m})$, and we have:

$$\mathbb{E}_{\mathcal{A}}\left(\left\|P^{-\frac{1}{2}} M x\right\|^2\right) = \|x\|^2 . \tag{5}$$

Thus, if we know the spanning eigenspace U_k, we can write the reconstruction problem as:

$$\begin{aligned} x_{\text{rec}} &= \operatorname{argmin}_{z \in \operatorname{span}(U_k)} \left\|P^{-\frac{1}{2}}\left(Mz - y\right)\right\|^2 \\ &= U_k \operatorname{argmin}_{\alpha \in \mathbb{R}^m} \left\|P^{-\frac{1}{2}}\left(MU_k\alpha - y\right)\right\|^2 \end{aligned}$$

$$\boxed{x_{\text{rec}} = U_k(U_k^\top M^\top P^{-1} M U_k)^{-1} U_k^\top M^\top P^{-1} y}. \tag{6}$$

In order to use the last formula, we must compute the inverse of an $N \times N$ matrix, which might be unfeasible if N is large. One could then use other methods, such as gradient descent.

The next step is then to find some marginal kernel K such that $x_{\text{rec}} = x$ with (6). We can show that if we use the kernel

$$\boxed{K_k = U_k U_k^\top}, \tag{7}$$

then any sample $\mathcal{A}$ from this DPP verifies $|\mathcal{A}| = k$ a.s., (3) and thus perfect reconstruction can be achieved if there is no noise.

1.4 Reconstruction with unknown U_k using DPPs

In the preceding sections, we assumed U_k was known. It is possible, if N is very large, that the computation of U_k is unfeasible ; in this situation, we can not use the DPP given by K_k. A possible substitute is to consider the kernel:

$$\boxed{K_q = U g_q(\Lambda) U^\top} \tag{8}$$

where $q > 0$ and $g_q(\lambda) = \frac{q}{q+\lambda}$. We can note that this is in fact an approximation of the kernel $K_k = U_k U_k^\top = U h_k(\Lambda) U^\top$ where $h_k(\lambda) = \mathbb{1}_{\lambda \leq \lambda_k}(\lambda)$. Moreover, there exists an Algorithm that enables sampling from the DPP of marginal kernel K_q without having to compute explicitely K_q, which makes this DPP particularly interesting.

In this case, we do not know explicitely U_k so we can not use the direct reconstruction formula (6). Instead, we can use a regularized version of this formula that penalizes high frequencies:

$$\boxed{x_{\text{rec}} = \operatorname{argmin}_{z \in \mathbb{R}^N} \left\|P^{-\frac{1}{2}}\left(Mz - y\right)\right\|^2 + \gamma z^\top L^r z} \tag{9}$$

where $\gamma > 0$, $r > 0$ are regularization parameters.

2 Implementation details

This project was mainly focused on implementation and experimentation. We used Python to implement the main algorithms. The code can be found here:

`https://github.com/14chanwa/graphsmlProject`

As we explained in the preceding section, our work decomposes in the following parts:

1. Generation of some graphs for benchmarking. We chose to work with community-structured graphs, as in [5].
2. Generation of k-bandlimited signals as in (1).
3. Sampling DPPs with marginal kernel K_k as in (7).
4. Sampling DPPs with marginal kernel K_k as in (8).
5. Reconstruct x from y as in (2) using the formula with known U_k (6).
6. Reconstruct x from y as in (2) using the formula without U_k (9).

Since L and W are sparse, we used functions from `scipy.sparse` as much as possible, for our implementation to scale well with N.

2.1 Random graph generation

Function `generate_graph_from_stochastic_block_model`

For benchmarks, we chose to use community-structured graphs using the Stochastic Block Model (SBM) as in [5]. Consider a graph with k communities of cardinal N/k. An edge has a probability q_1 to be drawn between nodes i and j if they belong to the same community, and q_2 else. We can then parameterize the SBM with N, k, q_1, q_2, or alternatively with N, k, ϵ, c where $\epsilon = \frac{q_2}{q_1}$ and $c = q_1\left(\frac{N}{k} - 1\right) + q_2\left(N - \frac{N}{k}\right)$ is the average degree of a node in the graph. One can also show that there is a critical value of ϵ, $\epsilon_c = (c - \sqrt{c})/(c + \sqrt{c}(k-1))$ above which the community structure becomes undetectable for large N's, so we can also use the ratio ϵ/ϵ_c as a parameter.

We used the Python library NetworkX [1] to generate L, W and provide the necessary plot functions.

2.2 Generation of k-bandlimited signals

Function `generate_k_bandlimited_signal`

In order to generate a k-bandlimited signal, we must compute U_k and take a linear combination of these vectors as in (1). This is equivalent to compute the k lowest eigenmodes of the Laplacian matrix L. Even though we could use the function `numpy.linalg.eigh` directly, we chose to use functions that scale better with N. The function `scipy.sparse.eigh` uses a routine that enables to compute some eigenmodes from a sparse matrix. It is more efficient to compute the largest eigenmodes than the lowest ; we make use of the *shift-inverse* mode[1] of this function in order to compute the lowest eigenmodes of L by computing the largest eigenmodes of a dual problem.

2.3 Sampling DPPs with marginal kernel K_k

Function `sample_from_DPP`

We suppose U_k known ; then we can compute $K_k = U_k U_k^\top$. We then have to build an algorithm that samples a DPP from a given marginal kernel K ; we used the algorithm given in [5, 2] in order to do so.

Our version of this algorithm is presented in Algorithm 1. Its principle is, given the eigendecomposition of K, to first sample some of its eigenvectors with probabilities given by their corresponding eigenvalues. Then, we select recursively the nodes of the sample with probabilities given by the selected eigenvectors.

[1] See: `https://docs.scipy.org/doc/scipy/reference/tutorial/arpack.html`

Algorithm 1 Sampling a DPP with marginal kernel K

```
Input: Eigendecomposition of $K$: $(\Lambda, V)$
$J \leftarrow \emptyset$
for $n = 1..N$ do
    $J \leftarrow J \cup \{n\}$ with probability $\lambda_n$
end for
$V \leftarrow \{v_n\}_{n \in J}$
$Y \leftarrow \emptyset$
while $|V| > 0$ do
    $P \leftarrow (\|v_i\|^2)^\top_{i=1..|V|} / |V|$ (vector of size $N$)
    $i \leftarrow$ choice of $i$ in $1..N$ with probability $P$
    $Y \leftarrow Y \cup \{i\}$
    if $|V| > 1$ then
        $j \leftarrow$ index of a $v_j$ such that $\langle v_j, e_i \rangle \neq 0$.
        Remove a linear combination of $v_j$ from $v_{j'}, j' \neq j$ such that $\forall j',\ \langle v_{j'}, e_i \rangle = 0$.
        Delete $v_j$ from $V$.
        Use Gram-Schmidt algorithm to orthonormalize $V$.
    else
        break
    end if
end while
Return: $Y$
```

In order to use this algorithm on K_k, we must be able to compute at least the k first eigenmodes of K_k (since the other eigenvalues of K_k are null, we do not make use of the corresponding eigenmodes) ; we could use the same technique as in Subsection 2.2.

2.4 Sampling DPPs with marginal kernel K_q (Wilson's algorithm)

Functions `wilson_algorithm`, `generate_maze`

Test file `test_generate_maze.py`

This is the main contribution of [5] ; in order to avoid computing K_k (and thus U_k), the author uses an approximation K_q and gives an algorithm capable of sampling from the DPP of marginal kernel K_q without explicitely computing the kernel. This algorithm is based on a modified version of Propp-Wilson's algorithm [3].

Originally, this algorithm samples a random spanning tree of a given directed graph. A recreational application of this algorithm is random maze generation: consider a graph composed of nodes on a grid, plus a node that will represent the border of the maze. The walls will be lines connecting these nodes: consider then the appropriate edges of this graph (two nodes are connected if they are adjacent, the border node is connected to all nodes in the border). Then when sampling a random spanning tree on this graph, one samples the walls of a maze such that there is one and only one way from one corridor to the other that do not imply turning back. We implemented this random maze generation in order to make sure our algorithm works, as presented in Figure 1.

The original algorithm works using loop-erased random walks in the graph until all nodes have been visited. In [5], we modify the algorithm so that it samples nodes from the graph by adding a sink node. At each step of the random walks, the walker has some probability depending on a parameter q to go to the sink: the last visited node is then part of the sample. This algorithm is presented in Algorithm 2.

3 Results

3.1 Sampling from K_k

Test file `test_sample_from_Kk.py`

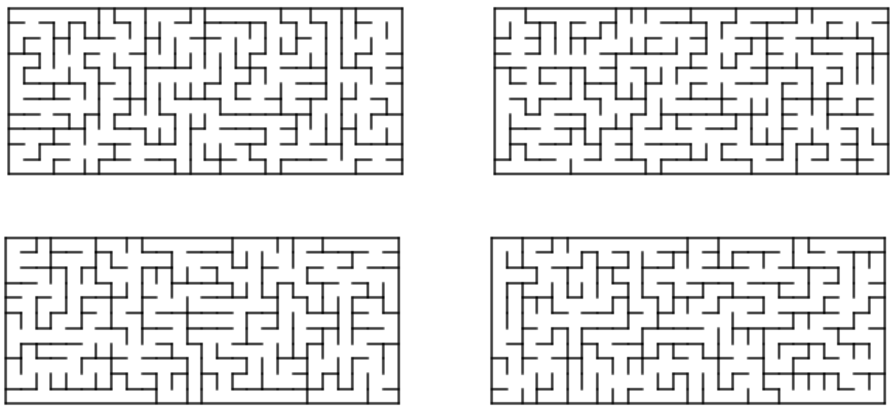

Figure 1: Examples of random mazes generated with random spanning trees on a grid of 25×10 wall nodes plus the border node.

Algorithm 2 Wilson's algorithm

```
**Input:** Adjacency matrix $W$, sink weight $q \geq 0$
$Y, \nu \leftarrow \emptyset$
**while** $\nu \neq \{1, \cdots, N\}$ **do**
   Begin a random walk $S$ from a node $i \notin \nu$.
   **if** it reaches itself **then**
      Erase the loop in the walk.
   **else if** it reaches some node in $\nu$ or the sink $\Delta$ **then**
      $\nu \leftarrow \nu \cup S$
      **if** the last node is $\Delta$ **then**
         If $\ell$ is the last visited node before $\Delta$, then $Y \leftarrow Y \cup \{\ell\}$.
      **end if**
   **end if**
**end while**
**Return:** $Y$
```

Note: our implementation also saves the sampled spanning tree. In the case $q = 0$, there is no sink and we have the original Propp-Wilson algorithm.

In order to make sure we sample the right DPP with marginal kernel K_k with our Algorithm 1, we generate a graph from the SBM (as in Subsection 2.1) and sample from our algorithm a number of times n. If the algorithm is correct:

- Every sample is of size k (this is verified in practice).
- For any singleton i, $p(i \in \mathcal{A}) = K_{i,i}$ (4).
- For any pair i, j, $i < j$, $p(i \in \mathcal{A}$ and $j \in \mathcal{A}) = \det \begin{pmatrix} K_{i,i} & K_{i,j} \\ K_{j,i} & K_{j,j} \end{pmatrix}$ (4).
- For a given k-bandlimited signal x, $\mathbb{E}_{\mathcal{A}} \left(\left\| P^{-\frac{1}{2}} M x \right\|^2 \right) = \|x\|^2$ (5).

We choose two singletons and two pairs, and we compute the empirical probabilities and expectations over the $n = 50000$ samples, with $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.5 \times \epsilon_c$. The results are as follow:

```
TODO: make a table instead
------- Singleton:
Theoretical proba= 0.0189521762207
Empirical proba= 0.01824
------- Singleton2:
 Theoretical proba= 0.010781904604
 Empirical proba= 0.01006
------- Pair:
 Theoretical proba= 0.000152315628169
 Empirical proba= 0.00016
------- Pair2:
 Theoretical proba= 0.000256030200653
 Empirical proba= 0.0002
------- Norms:
 mean np.linalg.norm(x)**2= 1.0
 mean np.linalg.norm(Pm12.dot(M).dot(x))**2= 1.00074165703
```

We thus confirm that our algorithm samples the right DPP.

3.2 Sampling from K_q

Test file `test_sample_from_Kq.py`

We proceed the same way with our Algorithm 2 that samples from a DPP with marginal kernel K_q, with the difference that the size of the sample is not fixed. Rather, we should have:

$$\mathbb{E}\left(|\mathcal{A}|\right) = \sum_{i=1}^{N} \frac{q}{q+\lambda_i}. \tag{10}$$

Since this sampling algorithm is much slower than the preceding one, we chose to test the following parameters: $n = 10000$ samples, with $N = 100$, $k = 2$, $c = 16$, $\epsilon = 0.5 \times \epsilon_c$ and $q = 1.0$. The results are as follow:

```
 TODO: make a table instead
------- Cardinal
 Theoretical= 6.95800812053
 Empirical= 6.9687
------- Singleton:
 Theoretical proba= 0.067520041977
 Empirical proba= 0.0657
------- Singleton2:
 Theoretical proba= 0.059207968264
 Empirical proba= 0.0559
------- Pair:
 Theoretical proba= 0.00417738808421
 Empirical proba= 0.0045
------- Pair2:
 Theoretical proba= 0.0047052228183
 Empirical proba= 0.0049
------- Norms:
 mean np.linalg.norm(x)**2= 1.0
 mean np.linalg.norm(Pm12.dot(M).dot(x))**2= 1.00289718132
```

We thus confirm that our algorithm samples the right DPP.

3.3 Signal reconstruction with known U_k

TODO: when no noise, perfect reconstruction ; when noise, sometimes it fails spectacularly, most of the time the error norm is small (10, 50, 90 quantiles)

3.4 Signal reconstruction with unknown U_k

TODO: when using sampling from K_q and reconstructing the signal with the exact formula (6), results similar to these of K_k (modulo cardinal of the sample, which we can not control very finely).

TODO: regularization not working yet

References

[1] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[2] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2-3):123–286, July 2012.

[3] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic markov chain and generate a random spanning tree of a directed graph. *J. Algorithms*, 27(2):170–217, May 1998.

[4] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. Signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular data domains. *CoRR*, abs/1211.0053, 2012.

[5] Nicolas Tremblay, Pierre-Olivier Amblard, and Simon Barthelmé. Graph sampling with determinantal processes. *EUSIPCO*, August 2017.