



HAL
open science

Algorithmic and software contributions to graph mining

Nathan de Lara

► **To cite this version:**

Nathan de Lara. Algorithmic and software contributions to graph mining. Data Structures and Algorithms [cs.DS]. Institut Polytechnique de Paris, 2020. English. NNT: 2020IPPAT029 . tel-03104061

HAL Id: tel-03104061

<https://theses.hal.science/tel-03104061>

Submitted on 8 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSTITUT
POLYTECHNIQUE
DE PARIS

NNT : 2020IPPAT029

Thèse de doctorat



Algorithmic and software contributions to graph mining

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique, Données et Intelligence Artificielle

Thèse présentée et soutenue à Paris, le 22 septembre 2020, par

NATHAN DE LARA

Composition du Jury :

Michalis Vazirgiannis Professeur, École polytechnique (LIX)	Président
Alexandre Gramfort Statut, INRIA (PARIETAL)	Rapporteur
Fabrice Rossi Professeur, Université Paris-Dauphine (CEREMADE)	Rapporteur
Meeyoung Cha Professeur associé, KAIST (School of Computing)	Examineur
Claire Lucas Ingénieur de recherche, Bouygues Télécom	Examineur
François Sausset Ingénieur de recherche, Thales	Examineur
Thomas Bonald Professeur, Télécom Paris (LTCI)	Directeur de thèse

Algorithmic and software contributions to graph mining

Nathan de Lara

Abstract

Since the introduction of Google's PageRank method for Web searches in the late 1990s, graph algorithms have been part of our daily lives. In the mid 2000s, the arrival of social networks has amplified this phenomenon, creating new use-cases for these algorithms. Relationships between entities can be of multiple types: *user-user* symmetric relationships for Facebook or LinkedIn, *follower-followee* asymmetric ones for Twitter or even *user-content* bipartite ones for Netflix, Deezer or Amazon. They all come with their own challenges and the applications are numerous: centrality calculus for influence measurement, node clustering for knowledge discovery, node classification for recommendation or embedding for link prediction, to name a few.

In the meantime, the context in which graph algorithms are applied has rapidly become more constrained. On the one hand, the increasing size of the datasets with millions of entities, and sometimes billions of relationships, bounds the asymptotic complexity of the algorithms for industrial applications. On the other hand, as these algorithms affect our daily lives, there is a growing demand for *explainability* and *fairness* in the domain of artificial intelligence in general. Graph mining is no exception. For example, the European Union has published a set of ethics guidelines for trustworthy AI. This calls for further analysis of the current models and even new ones.

This thesis provides specific answers via a novel analysis of not only standard, but also extensions, variants, and original graph algorithms. Scalability is taken into account every step of the way. Following what the Scikit-learn project does for standard machine learning, we deem it important to make these algorithms available to as many people as possible and participate in graph mining popularization. Therefore, we have developed an open-source software, Scikit-network, which implements and documents the algorithms in a simple and efficient way. With this tool, we cover several areas of graph mining such as graph embedding, clustering, and semi-supervised node classification.

Resumé

Depuis l'invention du PageRank par Google pour les requêtes Web à la fin des années 1990, les algorithmes de graphe font partie de notre quotidien. Au milieu des années 2000, l'arrivée des réseaux sociaux a amplifié ce phénomène, élargissant toujours plus les cas d'usage de ces algorithmes. Les relations entre entités peuvent être de multiples sortes : relations symétriques *utilisateur-utilisateur* pour Facebook ou LinkedIn, relations asymétriques *follower-followee* pour Twitter, ou encore, relations bipartites *utilisateur-contenu* pour Netflix ou Amazon. Toutes soulèvent des problèmes spécifiques et les applications sont nombreuses : calcul de centralité pour la mesure d'influence, le partitionnement de nœuds pour la fouille de données, la classification de nœuds pour les recommandations ou l'embedding pour la prédiction de liens en sont quelques exemples.

En parallèle, les conditions d'utilisation des algorithmes de graphe sont devenues plus contraignantes. D'une part, les jeux de données toujours plus gros avec des millions d'entités et parfois des milliards de relations limite la complexité asymptotique des algorithmes pour les applications industrielles. D'autre part, dans la mesure où ces algorithmes influencent nos vies, les exigences d'*explicabilité* et d'*équité* dans le domaine de l'intelligence artificielle augmentent. Les algorithmes de graphe ne font pas exception à la règle. L'Union européenne a par exemple publié un guide de conduite pour une IA fiable. Ceci implique de pousser encore plus loin l'analyse des modèles actuels, voire d'en proposer de nouveaux.

Cette thèse propose des réponses ciblées via l'analyse d'algorithmes classiques, mais aussi de leurs extensions et variantes, voire d'algorithmes originaux. La capacité à passer à l'échelle restant un critère clé. Dans le sillage de ce que le projet Scikit-learn propose pour l'apprentissage automatique sur données vectorielles, nous estimons qu'il est important de rendre ces algorithmes accessibles au plus grand nombre et de démocratiser la manipulation de graphes. Nous avons donc développé un logiciel libre, Scikit-network, qui implémente et documente ces algorithmes de façon simple et efficace. Grâce à cet outil, nous pouvons explorer plusieurs tâches classiques telles que l'embedding de graphe, le partitionnement, ou encore la classification semi-supervisée.

La thèse démarre par une introduction sur les structures de graphe ainsi qu'un certain nombre de rappels d'algèbre linéaire. Une attention particulière est accordée aux décompositions matricielles ainsi qu'aux méthodes stochastiques implémentées en pratique pour permettre de traiter les jeux de données massives. Dans un deuxième temps, cette thèse présente Scikit-network en détaillant les choix de design, les algorithmes implémentés ainsi que les performances obtenues vis-à-vis de logiciels concurrents.

Les chapitres qui suivent présentent les travaux originaux inclus dans Scikit-network. La première contribution théorique concerne la régularisation de l'embedding spectral obtenu par décomposition du Laplacien de graphe. On démontre sur un modèle simple qu'ajouter un graphe complet à la donnée d'origine permet à l'embedding spectral d'être moins sensible aux sous-graphes disjoints de la composante connexe principale. Dans un deuxième temps, on propose une variante de cet embedding pour les graphes dirigés et bipartis. On exhibe les propriétés mathématiques qui permettent d'interpréter facilement les représentations obtenues. L'ensemble de ces résultats est illustré par un grand nombre d'expériences sur données réelles.

On s'intéresse ensuite au célèbre algorithme de partition de nœuds Louvain. On démontre que cet algorithme peut être généralisé à toute une classe de fonctions de modularité et non pas

seulement celle proposée par Newman. On vérifie la pertinence des diverses modularités sur un ensemble de graphes réels de tous types : non-dirigés, dirigés et bipartis.

Dans une partie suivante, cette thèse propose une modification simple d'un célèbre algorithme de classification de nœuds semi-supervisé introduit par Zhou. On démontre sur un modèle simple en quoi l'algorithme d'origine est biaisé envers la classe majoritaire dans les *seeds* et que la modification proposée assure bien une classification correcte. Cette amélioration est confirmée par la suite sur données simulées et réelles.

Enfin, la thèse se termine par quelques contributions sur les algorithmes de classification de graphes. On compare une méthode basée sur la décomposition spectrale du Laplacien à une autre qui combine parcours de graphe et réseau de neurones récurrents. Les performances sont mises en perspective de l'état de l'art dans le domaine afin de démontrer la pertinence des approches proposées.

Contents

1	Introduction	4
1.1	Graphs	4
1.2	Graph Mining	6
1.3	Data structures	8
1.3.1	Sparse matrix formats	8
1.3.2	Performance analysis	9
1.4	Linear algebra for graph mining	10
1.4.1	Transposition	10
1.4.2	Matrix-vector product	11
1.4.3	Eigenvalue decomposition	11
1.4.4	Singular value decomposition	12
1.4.5	Halko's randomized method	13
1.4.6	Linear operators	14
1.4.7	Experiments	15
1.5	Contributions	16
1.6	Publications	17
2	Scikit-network	21
2.1	Motivation	22
2.2	Software features	22
2.3	Project Assets	24
2.4	Resources	24
2.5	Performance	25
2.6	The NetSet repository	25
3	Spectral embedding	27
3.1	Problem statement	28
3.2	Regularization	29
3.2.1	Aggregation of Block Models	30
3.2.2	Regularization of Block Models	31
3.2.3	Illustration	34
3.2.4	Extensions	34
3.2.5	Regularization of Bipartite Graphs	35
3.3	Interpretable Graph Embedding	38
3.3.1	Matrix norm	38
3.3.2	Co-neighbor graph	38
3.3.3	Graph embedding	39

3.3.4	Case of directed graphs	45
3.3.5	Case of undirected graphs	46
3.3.6	Link with spectral embedding	46
3.4	About embedding metrics	47
3.5	Experiments	49
3.5.1	Metrics	49
3.5.2	Datasets	50
3.5.3	Influence of Laplacian normalization	50
3.5.4	Influence of regularization	51
3.5.5	Influence of embedding shift	52
3.5.6	Cosine similarity versus L2 norm	52
4	Node clustering	54
4.1	Louvain algorithms	54
4.1.1	Modularity functions	55
4.1.2	Generalized Louvain algorithm	59
4.2	Experiments	61
4.2.1	Illustration	61
4.2.2	General benchmark	61
4.2.3	Digits classification	63
5	Semi-supervised node classification	65
5.1	Introduction	65
5.2	Dirichlet problem on graphs	67
5.2.1	Heat equation	67
5.2.2	Random walk	67
5.2.3	Solution to the Dirichlet problem	68
5.2.4	Heat kernel	69
5.2.5	Extensions	70
5.3	Node classification algorithm	70
5.3.1	Binary classification	70
5.3.2	Multi-class classification	71
5.3.3	Time complexity	72
5.4	Analysis	72
5.4.1	Block model	72
5.4.2	Dirichlet problem	73
5.4.3	Classification	73
5.5	Experiments	74
5.5.1	Synthetic data	74
5.5.2	Real data	75
5.5.3	General benchmark	77
6	Graph classification	80
6.1	Challenges	80
6.2	Related work	81
6.3	Spectral method	82
6.4	Variational RNN method	83
6.5	Experiments	87
6.5.1	Datasets	87

6.5.2	General benchmark	88
6.5.3	Influence of the classifier for the spectral model	89
6.5.4	Influence of k for the spectral model	90
6.5.5	Node indexing invariance for VRGC	90
7	Conclusions and perspectives	92

Chapter 1

Introduction

1.1 Graphs

Graphs are mathematical objects describing the relations between entities called *vertices* or *nodes*. If two nodes are in relation, they are said to be connected by an *edge*. Mathematically, we denote by $G = (V, E)$ a graph with V the set of vertices, which we assume finite, and $E \subset V \times V$, the set of edges.

In this work, we consider an extension of the previous definition where weights are associated with edges. These objects are called weighted graphs, but we simply refer to them as graphs. Let n be the number of nodes of a graph, $n = |V|$.

The adjacency matrix associated with the graph is the matrix $A \in \mathbb{R}^{n \times n}$ such that

$$A_{ij} = \begin{cases} \omega & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

ω being the weight associated to the edge (i, j) .

Besides, we differentiate three main families of graphs which we illustrate in Figure 1.1.

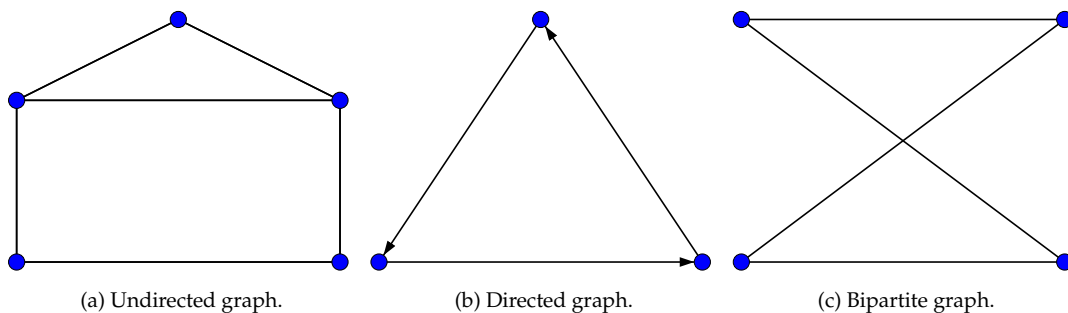


Figure 1.1: Some toy graphs.

Undirected graphs In these graphs, relationships are symmetric. This means that if $(i, j) \in E$, then $(j, i) \in E$ as well. This is for example the case for most social networks such as Facebook friendship graph or LinkedIn professional relationships. In particular, this implies that the associated adjacency matrix is symmetric.

Directed graphs In these graphs, the relationships are not necessarily symmetric, i.e. $(i, j) \in E$ does not imply $(j, i) \in E$. This is the case for the Twitter graph in which a user can follow another one without being followed in return. Another example is the graph of hyperlinks between Web pages. In this case, A is not necessarily symmetric. We use the terms *source node* and *target node* to differentiate both ends of directed edges.

Bipartite graphs These are a special case of undirected graphs with two disjoint sets of nodes V_1 and V_2 such that relationships only exist between V_1 and V_2 . On the other hand, two nodes of V_1 cannot be connected nor can two nodes of V_2 . This is the case for relationships such as *user-content* in Netflix or *user-item* in retail. We denote by B the biadjacency matrix, possibly rectangular, such that the adjacency matrix of the graph seen as undirected is

$$A = \begin{bmatrix} 0 & B \\ B^\top & 0 \end{bmatrix}.$$

Note that this list is not exhaustive. There are many other types of graphs such as knowledge graphs, signed graphs, hypergraphs, annotated or infinite graphs... However, they lie beyond the scope of the present work.

To conclude this section, we introduce a few notations that will help further reading. Unless otherwise specified:

- n denotes the number of nodes in a graph.
- m denotes the number of non-zero entries in an adjacency matrix. For an undirected graph without self-loops, this is twice the number of edges.
- $\mathbf{1}$ denotes a vector of ones whose dimension depends on context.
- d is the out-degree vector defined by $A\mathbf{1}$ or $B\mathbf{1}$. If the graph is not weighted, the out-degree d_i of a given node i is simply the number of edges for which i is the source. If the edges are weighted, d_i is the total weight of edges for which i is the source.
- D is the out-degree diagonal matrix defined by $D = \text{diag}(d)$.
- f is the in-degree vector defined by $A^\top\mathbf{1}$ or $B^\top\mathbf{1}$. If the graph is not weighted, the in-degree f_i of a given node i is simply the number of edges for which i is the target. If the edges are weighted, f_i is the total weight of edges for which i is the target.
- F is the in-degree diagonal matrix defined by $F = \text{diag}(f)$.
- w is the total weight of the graph defined by $\mathbf{1}^\top A\mathbf{1}$ or $\mathbf{1}^\top B\mathbf{1}$.

1.2 Graph Mining

The notion of graph mining covers a wide variety of tasks on graphs, most of which have direct industrial applications. Here, we provide a non-exhaustive list of examples.

Node embedding This task consists in mapping each node of the graph to a vector, see Figure 1.2. Usually, the embedding space has a small dimension with respect to the number of nodes in the graph. Embeddings are designed such that the distances between nodes in the vector space reflect some topological properties of the original graph. The embedding can be used to perform standard machine learning tasks such as clustering or classification or link prediction as in the famous Netflix Prize [Bell et al., 2007]. Chapter 3 is dedicated to the study of a widely used embedding: the spectral embedding.

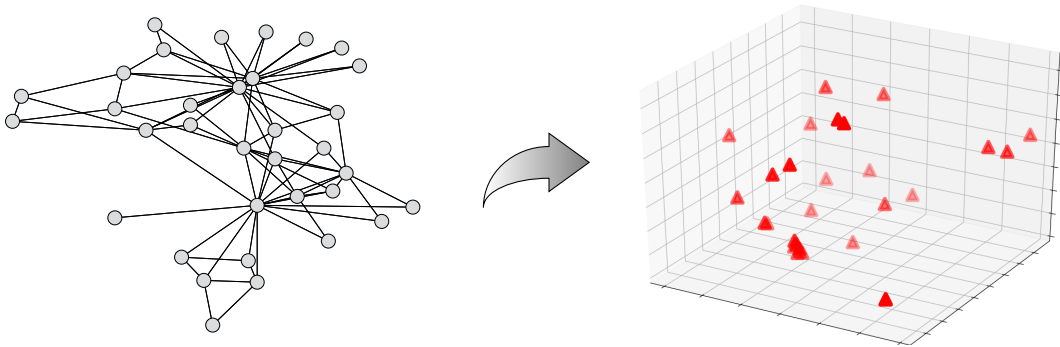


Figure 1.2: Node embedding.

Node clustering This task consists in assigning each node to a group or cluster such that nodes in the same cluster are more connected with one another than with the rest of the graph. See Figure 1.3 for an illustration and Schaeffer [2007] for a review of some algorithms. Applications include load balancing for wireless sensors networks [Younis et al., 2006], anomaly detection for cyber-security or financial fraud detection [Akoglu et al., 2015]. Node clustering is the subject of Chapter 4.

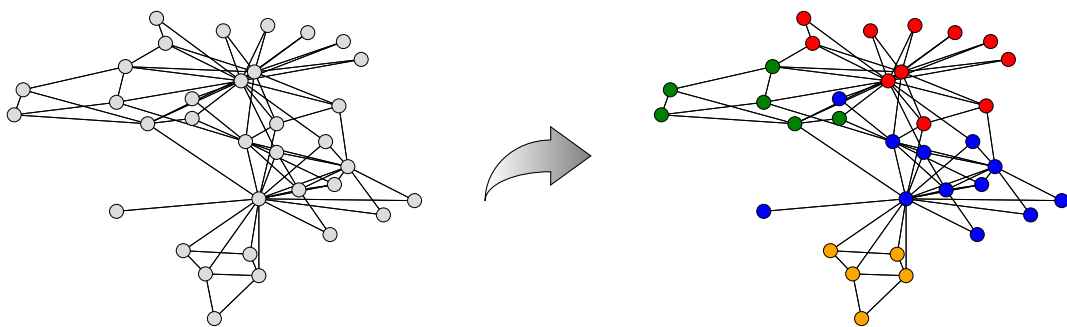


Figure 1.3: Node clustering.

Node classification This task consists in assigning a label to each node of the graph based on some partial ground-truth information, see Figure 1.4 for an illustration. It can be used to assign a polarity to tweets [Speriosu et al., 2011], predict potentially dangerous drug-drug interactions [Zhang et al., 2015] or automatically assign a genre to a movie or a song in a database. We discuss node classification algorithms in Chapter 5.

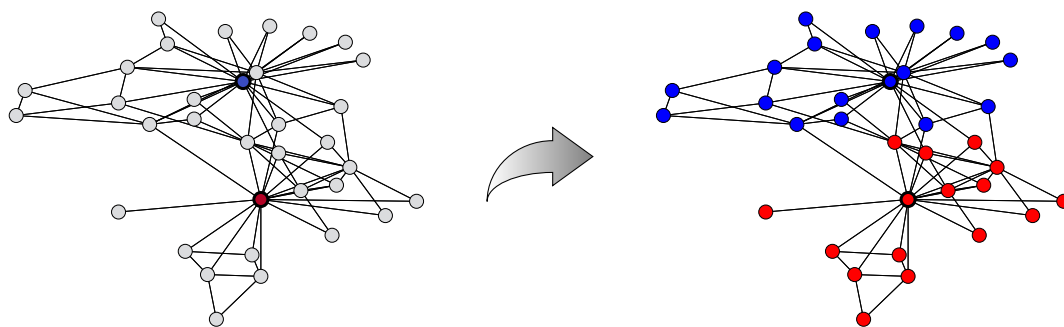


Figure 1.4: Node classification.

Node ranking This task consists in assigning an importance score to each node of the graph with respect to a certain criterion, see Figure 1.5 for an illustration. This is particularly useful for recommender systems such as Web search engines [Page, 2001] or contact recommendation in social networks. Such methods can also be used in a pipeline for node classification as we will see in Chapter 5.

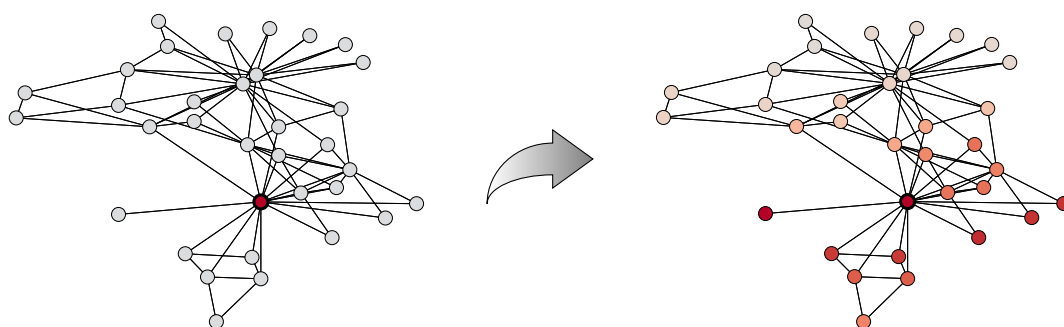


Figure 1.5: Node ranking.

Graph classification This task is the same as the standard supervised classification in machine learning with the main difference that the samples are graphs instead of vectors. Standard applications are molecular compounds classification in biology [Childs et al., 2009, Kudo et al., 2005] or malware detection in cyber-security [Hu et al., 2009, Gascon et al., 2013, Kinable and Kostakis, 2011]. We discuss some graph classification algorithms in Chapter 6.

Link prediction Among other graph mining tasks, we deem it important to mention *edge prediction* which consists in assigning a probability of existence to unobserved relations among nodes. However, we do not tackle this task in the present work.

1.3 Data structures

In order for graph algorithms to scale, they must rely on efficient data structures. This section introduces some standard graph formats. We present some of their respective strengths and weaknesses with a special emphasis on the Compressed Sparse Row format (CSR) which is used in most of our implementations. We refer the reader to SciPy’s documentation¹ for a more extensive list.

1.3.1 Sparse matrix formats

Let $G = (V, E)$ be a graph with n nodes and associated adjacency matrix $A \in \mathbb{R}^{n \times n}$ such that m is the number of nonzero entries in A . Storing G boils down to storing A . The main assumption we make is that A is sparse, which means that it has very few nonzero coefficients: $m \ll n^2$. In other words, the average number of neighbors of a node is small compared to n . For real large graphs, it is common to have densities, i.e. m/n^2 ratios, below 10^{-5} (see Table 1.4). Hence, these adjacency matrices are made of more than 99.99% of zeros which it is useless to store explicitly in memory. Table 1.1 presents an overview of some standard formats which we detail below.

format	memory space	linear algebra friendly	flexible
COO	$2mM_{\text{int}} + mM_{\text{edge}}$	✗	✓
CSR/CSC	$(n + m)M_{\text{int}} + mM_{\text{edge}}$	✓	✗
DOK	mM_{key}	✗	✓

Table 1.1: Some standard formats for adjacency matrices. M_{int} , M_{edge} and M_{key} denote the memory space of an integer, an edge and a key of the hash table, respectively.

Coordinates (COO) A first natural way to store the adjacency matrix of a graph is to store the corresponding edges of the graph (i, j, ω) where i is the source node, j is the target node and ω is the associated weight (possibly Boolean, float or integer). In this format, the edges are stored in no particular order which has some implications for basic graph primitives.

Compressed Sparse Row or Column (CSR, CSC) The CSR format is composed of three arrays, an array of size m named **indices**, which is the concatenation of the nodes successors, an array of size m named **data** to store the associated edges weights and an index pointer, **indptr**, of size $n + 1$ indicating which section of indices corresponds to which node in the graph. Example 1.3.1 illustrate this format on the *house graph* introduced in Figure 1.1a. See that node 0, top of the roof, has two neighbors, nodes 1 and 4. The CSC format works the same way but the indices array indicates the nodes predecessors.

¹<https://docs.scipy.org/doc/scipy/reference/sparse.html>

Example. 1.3.1: CSR format

```
[1]: from sknetwork.data import house
      A = house()
      print(A.indptr)
      print(A.indices)

[1]: [ 0  2  5  7  9 12]
      [1  4  0  2  4  1  3  2  4  0  1  3]
```

Dictionary of keys (DOK) As suggested by the name, this format maps tuples of indices (i, j) to keys of a dictionary with associated edges weights (possibly Boolean). It is effective for values access or updates but the hash table is usually more expensive in memory usage than other formats.

1.3.2 Performance analysis

We illustrate the performance of the different graph formats for a few standard graph primitives on a real dataset, Wikivitals (WV). The graph represents hyperlinks between some pages of Wikipedia and the adjacency matrix has 10 012 rows and 792 091 Boolean entries. The pages are labeled with their Wikipedia category (Art, Science, History...) which allows to later use supervised metrics on this dataset in Chapters 4 and 5. Note that we made this dataset available online² and via Scikit-network as explained in Chapter 2.

The test tasks are the following:

- **Memory space:** Make a pickle dump of the adjacency matrix and record the memory space.
- **Node insertion:** Add one isolated node i.e. add one extra row and one extra column with a diagonal entry to the adjacency matrix.
- **Node removal:** Remove the first row and first column of the matrix.
- **Edge insertion:** Add a link between nodes 0 and 1 (which does not exist in the original graph).
- **Edge removal:** Remove the previously created link between nodes 0 and 1 and eliminate the explicit 0 value in the matrix.
- **Out-neighbors access:** Get all successors of node 0.
- **In-neighbors access:** Get all predecessors of node 0.

We repeat each task involving a runtime measurement a hundred times for each format. Average values are reported in Table 1.2. If a format requires to be converted to perform a given operation, we do not record the runtime and replace it by "-".

²https://graphs.telecom-paristech.fr/Home_page.html

Setups Unless otherwise specified, we always use one of the following setups for our experiments which we refer to as *laptop setup* and *server setup*:

- **Laptop:** Laptop equipped with an Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz processor and 16 Go of RAM.
- **Server:** Computer equipped with an AMD Ryzen Threadripper 1950X 16-Core Processor and 32 Go of RAM. For deep learning experiments, the computer is equipped with an NVIDIA TitanXp GPU.

In this case, all experiments are run on the laptop setup.

The conclusion of these experiments is that, as long as the structure of the matrix does not need to be modified, the CSR and CSC formats are both lighter in memory and more efficient. On the other hand, the DOK format is relevant when edges need to be modified often (in a dynamic graph, for example). In the rest of this thesis, the default format of adjacency matrices is the CSR one, unless otherwise specified.

task	COO	CSR	CSC	DOK
Memory space (Mo)	6.8	3.8	3.8	46.8
Node insertion (ms)	5.7	11	11	108
Node removal (ms)	-	2.6	2.5	743
Edge insertion (ms)	-	0.16	0.15	0.01
Edge removal (ms)	-	0.7	0.7	0.01
Out-neighbors access (μ s)	301	0.6	-	1992
In-neighbors access (μ s)	331	-	0.6	3220

Table 1.2: Performance on Wikivitals.

1.4 Linear algebra for graph mining

This section covers some standard linear algebra primitives in the context of graph mining. Part of this work has been published in [De Lara \[2019\]](#).

1.4.1 Transposition

Depending on whether an algorithm requires access to the successors or the predecessors of the nodes in the graph, it might need to transpose the adjacency matrix. This operation is immediate in the COO format as it only requires to switch the *row* and *col* vectors. However, this is not the case for the CSR format.

The canonical way to compute A^T in CSR format from A in CSR format requires an intermediate transformation into COO format. Going from CSR to COO only requires to compute the *row* vector from the *indptr* one which is an operation linear in m . However, going from COO to CSR requires a sort of the indices which is $O(m \log(m))$ operations. In the end, the complexity of CSR matrix transposition is $O(m \log(m))$.

1.4.2 Matrix-vector product

Matrix-vector product is a key primitive for many graph algorithms. It can be used to emulate discrete time random walks, perform low dimension projections, message passing... It is therefore critical to perform it efficiently.

Respective pseudocodes for COO and CSR formats are presented in Example 1.4.1. In both cases, the complexity is linear with the number of nonzero coefficients in the matrix. The main difference is that, as indices are sorted in the CSR, the processor can optimize caching operations.

Example. 1.4.1: Sparse matrix-vector products

```
[1]: import numpy as np

def coo_dot(n, row, col, data, x):
    y = np.zeros(n)
    for i, w in enumerate(data):
        y[row[i]] += w * x[col[i]]
    return y

def csr_dot(indptr, indices, data, x):
    n = len(indptr) - 1
    y = np.zeros(n)
    for i in range(n):
        start, end = indptr[i], indptr[i+1]
        y[i] = data[start:end].dot(x[indices[start:end]])
    return y
```

There are several more complex sparse matrix formats designed to optimize matrix-vector product, especially in the context of parallel computing and GPU computations. See Liu et al. [2013], Bell and Garland [2008] for more details.

1.4.3 Eigenvalue decomposition

Let A be a symmetric matrix with real coefficients. The spectral theorem states that there exists an orthonormal matrix X and a diagonal matrix Λ with real coefficients such that

$$A = X\Lambda X^\top.$$

The columns of X are the eigenvectors of A while the diagonal entries of Λ are the associated eigenvalues.

Several graph algorithms rely on the computation of some of these eigenvectors and eigenvalues. In this case, we refer to this as a partial eigenvalue decomposition.

In practice, iterative power methods are used to compute the k leading eigenvectors, i.e. the eigenvectors associated with the k largest eigenvalues. The idea is to start from a random vector x_0 and compute the sequence (x_t) defined by

$$x_{t+1} = \frac{Ax_t}{\|Ax_t\|}.$$

The sequence (x_t) provably converges with probability 1 towards the leading eigenvector of A , provided that it is unique. Based on this result, the Arnoldi method [Arnoldi, 1951] uses the stabilized Gram-Schmidt process to compute the k leading eigenvectors by orthonormalization of $x_t, x_{t-1}, \dots, x_{t-k+1}$. Note that the Arnoldi method applies to non-Hermitian matrices. In the Hermitian case, it is called the Lanczos method [Lanczos, 1950].

Observe once again that the main primitive for this method is the matrix-vector product.

An extension of the eigenvalue decomposition is the generalized eigenvalue decomposition. Given another matrix W , the generalized eigenvalue of A is such that

$$AX = WX\Lambda.$$

Proposition 1. *Assume that W is a diagonal matrix with positive entries and let X denote some eigenvectors for the eigenvalue decomposition of $W^{-1/2}AW^{-1/2}$. Then, $X' = W^{-1/2}X$ are eigenvectors for the generalized eigenvalue decomposition of A .*

Proof. Let X be eigenvectors of $W^{-1/2}AW^{-1/2}$ such that

$$W^{-1/2}AW^{-1/2} = X\Lambda X^\top.$$

Then,

$$\begin{aligned} AW^{-1/2}X &= W^{1/2}X\Lambda(X^\top X), \\ AX' &= WX'\Lambda. \end{aligned}$$

Such that X' are generalized eigenvectors of A .

□

1.4.4 Singular value decomposition

The singular value decomposition (SVD) of a real matrix $M \in \mathbb{R}^{n_1 \times n_2}$ is a factorization of the form

$$M = U\Sigma V^\top,$$

where $U \in \mathbb{R}^{n_1 \times n_1}$ and $V \in \mathbb{R}^{n_2 \times n_2}$ are orthonormal matrices whose columns are respective left and right singular vectors and $\Sigma \in \mathbb{R}^{n_1 \times n_2}$ is a diagonal matrix containing the nonnegative singular values.

Note that, U and V are respective eigenvectors of MM^\top and $M^\top M$, the associated eigenvalues being the square of the singular values. Besides, a partial SVD, extracting only the k leading singular vectors couples is an optimal rank k approximation of M with respect to the Frobenius norm:

$$U^{(k)}\Sigma^{(k)}V^{(k)\top} \in \arg \min_{\text{rank}(X)=k} \|M - X\|_F.$$

As for the eigenvalue decomposition, this factorization admits a generalization. Given two matrices W_1 and W_2 , the generalized singular value decomposition of M satisfies

$$\begin{cases} MV &= W_1 U \Sigma, \\ M^\top U &= W_2 V \Sigma. \end{cases}$$

Proposition 2. Assume that W_1 and W_2 are diagonal matrices with positive entries and let U, V be left and right singular vectors of $W_1^{-1/2}MW_2^{-1/2}$. Then, $U' = W_1^{-1/2}U$ and $V' = W_2^{-1/2}V$ are singular vectors for the generalized eigenvalue decomposition of M .

Proof. Let U, V be singular vectors of $W_1^{-1/2}MW_2^{-1/2}$ such that

$$W_1^{-1/2}MW_2^{-1/2} = U\Sigma V^\top.$$

Then,

$$\begin{aligned} MW_2^{-1/2}V &= W_1^{1/2}U\Sigma(V^\top V), \\ MV' &= W_1U'\Sigma. \end{aligned}$$

Similar derivations lead to the second equation such that U', V' are generalized singular vectors of M . \square

1.4.5 Halko's randomized method

Even with an efficient matrix-vector product, computing partial decompositions of very large matrices can be computationally prohibitive. In this section we present Halko's randomized projection method [Halko et al., 2009] which is a good solution to this issue.

The idea is to compute the decomposition of another matrix, much smaller than the original one and approximate the desired eigenvectors or singular vectors through a linear transformation.

First, one must find an orthonormal projection matrix $Q \in \mathbb{R}^{n \times p}$ such that

$$M \approx QQ^\top M.$$

Algorithm 1.1 offers to do so using a combination of random projection, power iteration and QR decompositions. Note that, once again, the only two actions on the sparse matrix M are matrix-vector products and transposition. In practice, p has to be chosen between $k + 7$ and $2k$ where k is the number of desired eigenvectors or singular vectors. As k is usually very small compared to n , the dimension p of the rows of Q is also small. The power iteration parameter q has to be a small integer as well, typically between 3 and 10.

Algorithm 1.1 Randomized Subspace Iteration

Require: input matrix $M \in \mathbb{R}^{n \times n}$

Generate a Gaussian test matrix $\Omega \in \mathbb{R}^{n \times p}$.

Form $Y_0 = M\Omega \in \mathbb{R}^{n \times p}$ and compute its QR factorization $Y_0 = Q_0R_0$, with $Q_0 \in \mathbb{R}^{n \times p}$ and $R_0 \in \mathbb{R}^{p \times p}$.

for $i = 1, \dots, q$ **do**

Form $\tilde{Y}_i = M^\top Q_{i-1}$ and compute its QR factorization $\tilde{Y}_i = \tilde{Q}_i\tilde{R}_i$.

Form $Y_i = M\tilde{Q}_i$ and compute its QR factorization $Y_i = Q_iR_i$.

end for

$Q = Q_q$.

Once the matrix Q is computed, one can perform an eigenvalue decomposition of either

$$M' = Q^\top M Q \in \mathbb{R}^{p \times p}$$

for eigenvalue decomposition of M , or

$$M'' = Q^\top M M^\top Q \in \mathbb{R}^{p \times p}$$

for its singular value decomposition. As $p \ll n$, such decompositions are much less computationally expensive than a direct decomposition of M .

Now, let $\bar{x} \in \mathbb{R}^p$ be an eigenvector of M' associated with the eigenvalue λ . Then, $x = Q\bar{x} \in \mathbb{R}^n$ approximates an eigenvector of M as

$$Mx = MQ\bar{x} \approx QQ^\top MQ\bar{x} = QM'\bar{x} = Q\lambda\bar{x} = \lambda x.$$

The same thing goes for \bar{u}, \bar{v} left and right singular vectors of M'' associated with the eigenvalue σ^2 . Let $u = Q\bar{u}$ and $v = Q\bar{v}$, then

$$MM^\top v = MM^\top Q\bar{v} \approx QQ^\top MM^\top Q\bar{v} = QM''\bar{v} = Q\sigma^2\bar{v} = \sigma^2 v.$$

So that v is an approximate eigenvector of MM^\top associated with the eigenvalue σ^2 and consequently an approximate right singular vector of M associated with the singular value σ . Similar derivations show that u is an approximate left singular vector of M .

A classic use-case for Halko's algorithm is the implementation of large-scale Principal Component Analysis [Yu et al., 2017]. We present some other ones in Section 1.4.6 and in Chapter 3.

1.4.6 Linear operators

A linear operator of \mathbb{R}^n is a function f such that for $u, v \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$,

$$f(u + \lambda v) = f(u) + \lambda f(v).$$

In computer science, they are defined by two primitives, namely **matvec** (for matrix-vector product) and **transpose** which would respectively be f and its conjugate f^* in mathematical terms. Recall that the i^{th} column of a matrix is simply the image of the i^{th} vector of the canonical basis by the associated linear operator.

These structures are of great interest in graph mining for memory reasons. Indeed, they do not require to explicitly store the images of the canonical vectors. As previously stated, the matrix-vector product and the transposition are enough to actually compute eigenvalue and singular value decompositions.

As a matter of fact, several matrices of interest in graph mining have an explicit "sparse + low-rank" structure, which means they are of the form

$$M = S + xy^\top,$$

where S is a sparse matrix and x, y are vectors. Such matrices are dense, however, storing them as linear operators requires only to store $2m + 2n$ coefficients (as S^\top might have to be stored as well depending on the use-case). The matrix-vector product being computed as follows:

$$Mz = Sz + (y^\top z)x.$$

The operator corresponding to M can be written $(S, (x, y))$, such that the operator corresponding to M^\top is simply $(S^\top, (y, x))$.

For each one of the following examples from the literature, Table 1.3 identifies the sparse term S and the low-rank term (x, y) .

matrix	S (sparse)	x	y
Random surfer	$\alpha D^{-1}A$	$\frac{1-\alpha}{n}\mathbf{1}$	$\mathbf{1}$
Modularity	$\frac{1}{w}A$	$-\frac{\gamma}{w}\mathbf{d}$	$\frac{1}{w}\mathbf{d}$
Laplacian	$I - D_\alpha^{-1/2}AD_\alpha^{-1/2}$	$-\frac{\alpha}{n}\sqrt{\mathbf{d} + \alpha\mathbf{1}}$	$\sqrt{\mathbf{d} + \alpha\mathbf{1}}$
PCA	A	$\frac{1}{w}\mathbf{f}$	$\mathbf{1}$

Table 1.3: Identification of sparse and low-rank factors. ($D_\alpha = D + \alpha I$.)

- Random surfer’s transition matrix [Page et al., 1999a],

$$R_{ij} = \frac{\alpha A_{ij}}{d_i} + \frac{1-\alpha}{n},$$

where α is a float between 0 and 1 and n is the number of nodes in the graph.

- The modularity matrix [Newman, 2006b],

$$\mathcal{M}_{ij} = \frac{1}{w}A_{ij} - \frac{\gamma}{w^2}d_i d_j,$$

where $\gamma \geq 0$ is a resolution parameter.

- The regularized graph Laplacian [Zhang and Rohe, 2018],

$$(L_\alpha)_{ij} = \delta_{ij} - \frac{A_{ij} + \alpha/n}{\sqrt{(d_i + \alpha)(d_j + \alpha)}},$$

where δ is the Kronecker symbol such that $\delta_{ij} = 1$ if $i = j$ and 0 otherwise and $\alpha \geq 0$.

- The principal components analysis (PCA) matrix [Saerens et al., 2004],

$$\bar{A}_{ij} = A_{ij} - f_j/w.$$

1.4.7 Experiments

In this section, we illustrate the use of linear operators combined to Halko’s algorithm on real datasets.

We compute the partial eigenvalue decomposition and the partial singular value decomposition of the modularity matrix of 7 undirected and 7 bipartite graphs, respectively. In the bipartite case, the modularity matrix is defined as

$$Q = \frac{1}{w}B - \frac{\gamma}{w^2}df^\top,$$

as later discussed in Chapter 4.

The graphs are collected from the Konect database [Kunegis, 2013] so that they have a varying number of nodes and edges. Table 1.4 presents some characteristics of these datasets.

code	name	n	m	density
MK	Kangaroo	$2 \cdot 10^1$	$2 \cdot 10^2$	10^{-1}
JZ	Jazz musicians	$2 \cdot 10^2$	$5 \cdot 10^3$	10^{-1}
Shf	Hamsterster	$2 \cdot 10^3$	$2 \cdot 10^4$	10^{-3}
RC	Reactome	$6 \cdot 10^3$	$3 \cdot 10^5$	10^{-2}
GW	Gowalla	$2 \cdot 10^5$	$2 \cdot 10^6$	10^{-4}
SK	Skitter	$2 \cdot 10^6$	$2 \cdot 10^7$	10^{-5}
OR	Orkut	$3 \cdot 10^6$	$2 \cdot 10^8$	10^{-5}

(a) Undirected graphs.

code	name	n_1	n_2	m	density
SW	Southern women 1	$2 \cdot 10^1$	$2 \cdot 10^1$	$9 \cdot 10^1$	10^{-1}
UL	Unicode languages	$2 \cdot 10^2$	$6 \cdot 10^1$	$1 \cdot 10^3$	10^{-1}
SX	Sexual escorts	$1 \cdot 10^4$	$6 \cdot 10^3$	$4 \cdot 10^4$	10^{-3}
Mut	MovieLens user-tag	$4 \cdot 10^3$	$2 \cdot 10^4$	$4 \cdot 10^4$	10^{-3}
R2	Reuters-21578	$2 \cdot 10^4$	$4 \cdot 10^4$	$1 \cdot 10^6$	10^{-3}
M3	MovieLens 10M	$7 \cdot 10^4$	$1 \cdot 10^4$	$1 \cdot 10^7$	10^{-2}
RE	Reuters	$8 \cdot 10^5$	$3 \cdot 10^5$	$6 \cdot 10^7$	10^{-4}

(b) Bipartite graphs.

Table 1.4: Some characteristics of the datasets.

In a first experiment, we extract $k = 16$ components in each case and record the computation times for our modified Python implementation of Halko’s method. We compare our results to the computation times of the direct Lanczos method using SciPy [Virtanen et al., 2020]. Results are displayed in Figure 1.6.

In a second experiment, we record the respective running times for the *Reactome* (undirected) and *MovieLens user-tag* (bipartite) graphs for $k \in \{8, 16, 32, 64, 128, 256\}$. See Figure 1.7.

Results In the laptop setup, memory errors arise around n^2 or $n_1 n_2 = 10^9$ for the dense implementation. At this point, the Lanczos method is already between one and two orders of magnitude slower than Halko’s. On the other hand, the "sparse + low-rank trick" enables to handle graphs with more than 3 million nodes and 234 million edges in about five minutes.

Furthermore, note that the running time for the implicit dense matrix is quite close to the running time for its sparse component. This holds for a fixed k and varying graphs as well as for fixed graphs and varying k .

1.5 Contributions

In this chapter, we have introduced graphs, their related tasks and data structures as well as some useful linear algebra that will help further reading [De Lara, 2019]. The rest of this thesis is organized as follows:

In Chapter 2, we introduce Scikit-network, a Python software for graph mining which is used in almost all our experiments.

Chapter 3 presents two distinct contributions to spectral embedding. First, a novel analysis of a standard regularization technique [Lara and Bonald, 2020]. Second, an interpretable extension to bipartite and directed graphs based on the Generalized Singular Value Decomposition [Bonald and De Lara, 2018].

Chapter 4 is dedicated to node clustering. We propose a general framework to interpret *modularity-like* quality functions. Specifically, we make a novel connection between modularity maximization and block model fitting. Then, we detail how the Louvain heuristic can be generalized to find local optimums for all these quality functions.

In Chapter 5, we propose a novel algorithm for semi-supervised node classification based on heat diffusion. We highlight its efficiency with respect to other diffusion-based methods by proving its consistency on a simple model, then, we assess its performance on real graphs.

In Chapter 6, we discuss graph classification. We propose two distinct algorithms to address this problem [de Lara and Pineau, 2018, Pineau and de Lara, 2019b] and benchmark them on standard datasets.

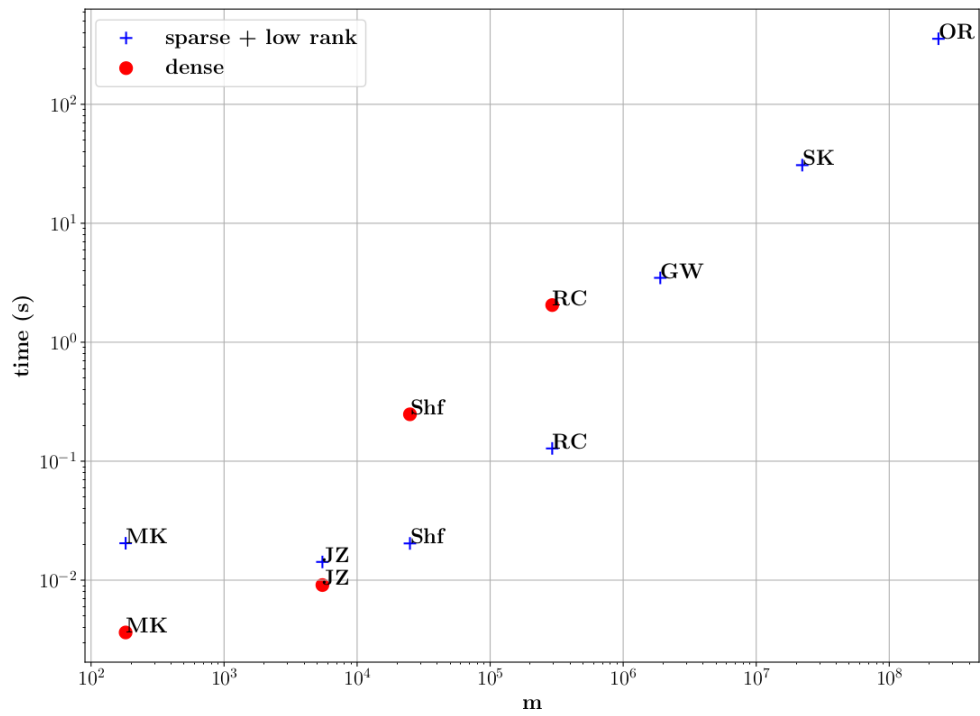
Finally, Chapter 7 concludes the thesis.

A schematic structure of the thesis is displayed in Figure 1.8.

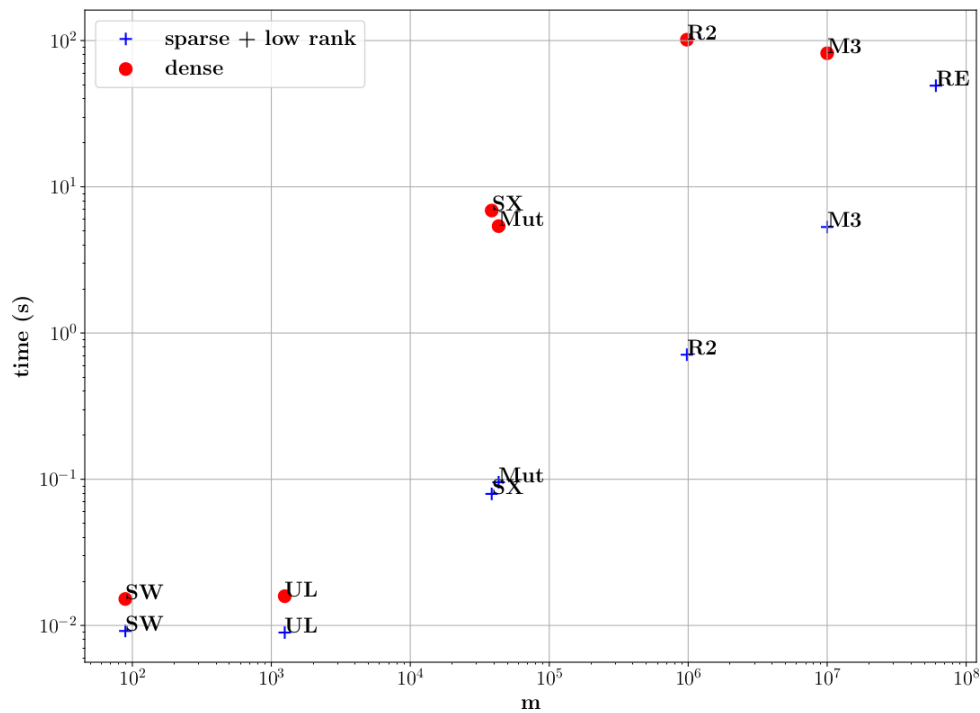
1.6 Publications

This section enumerates the publications related to the present thesis in chronological order.

- Bonald, T., & De Lara, N. (2018). *The Forward-Backward Embedding of Directed Graphs*. Openreview. Chapter 3.
- De Lara, N., & Pineau, E. (2018). *A simple baseline algorithm for graph classification*. Relational Representation Learning, NeurIPS 2018 Workshop. Chapter 6.
- Pineau, E., & De Lara, N. (2019). *Variational recurrent neural networks for graph classification*. In Representation Learning on Graphs and Manifolds Workshop. Chapter 6.
- De Lara, N. (2019). *The sparse + low rank trick for matrix factorization-based graph algorithms*. In Proceedings of the 15th International Workshop on Mining and Learning with Graphs. Chapter 1.
- De Lara, N., & Bonald, T. (2019). *Spectral embedding of regularized block models*. International Conference on Learning Representations. Chapter 3.
- Bonald, T., De Lara, N., & Lutz, Q. (2020). *Scikit-network: graph analysis in Python*. To be published in the Journal of Machine Learning Research. Chapter 2.
- De Lara, N., & Bonald, T. (2020). *A Consistent Diffusion-Based Algorithm for Semi-Supervised Classification on Graphs*. preprint. Chapter 5.

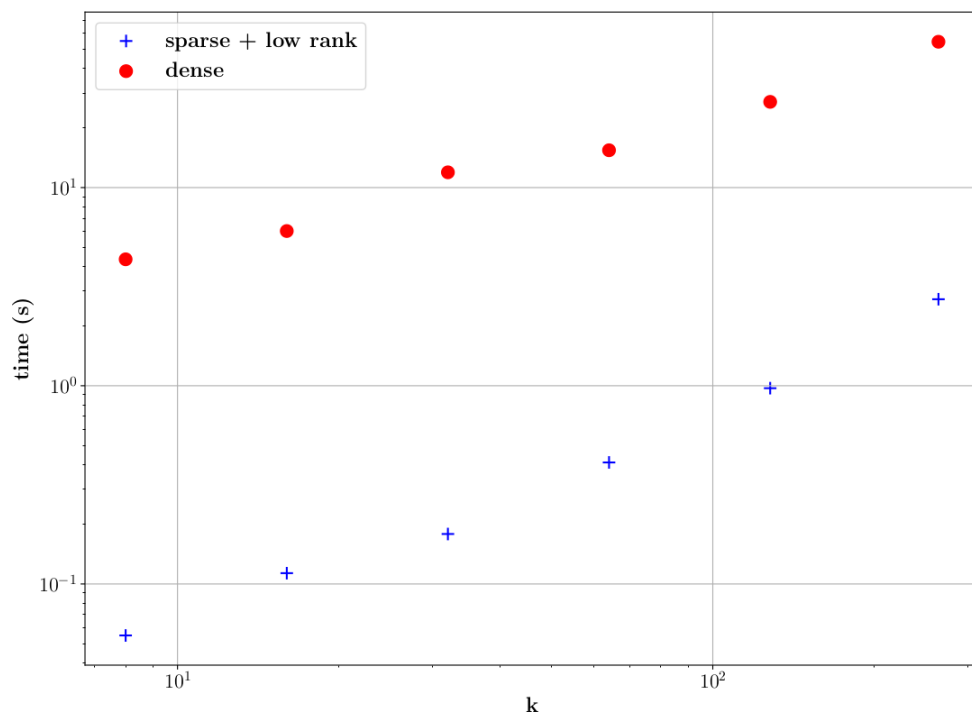


(a) Undirected graphs.

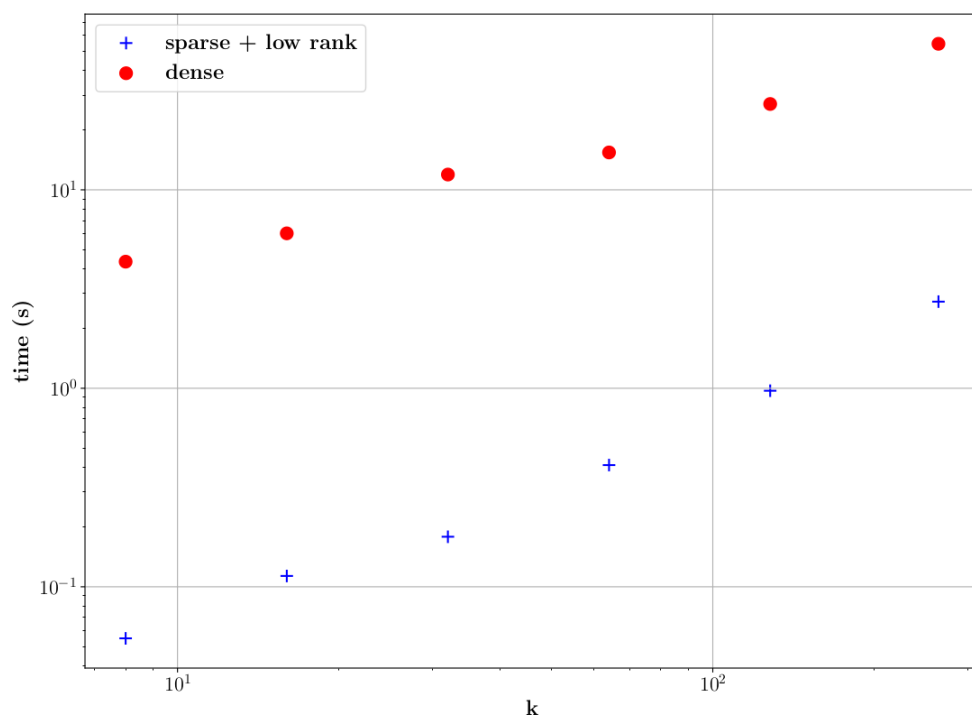


(b) Bipartite graphs.

Figure 1.6: Computation times in seconds for partial decomposition of matrices.



(a) Reactome.



(b) MovieLens.

Figure 1.7: Computation times in seconds for different values of the number of components, k .

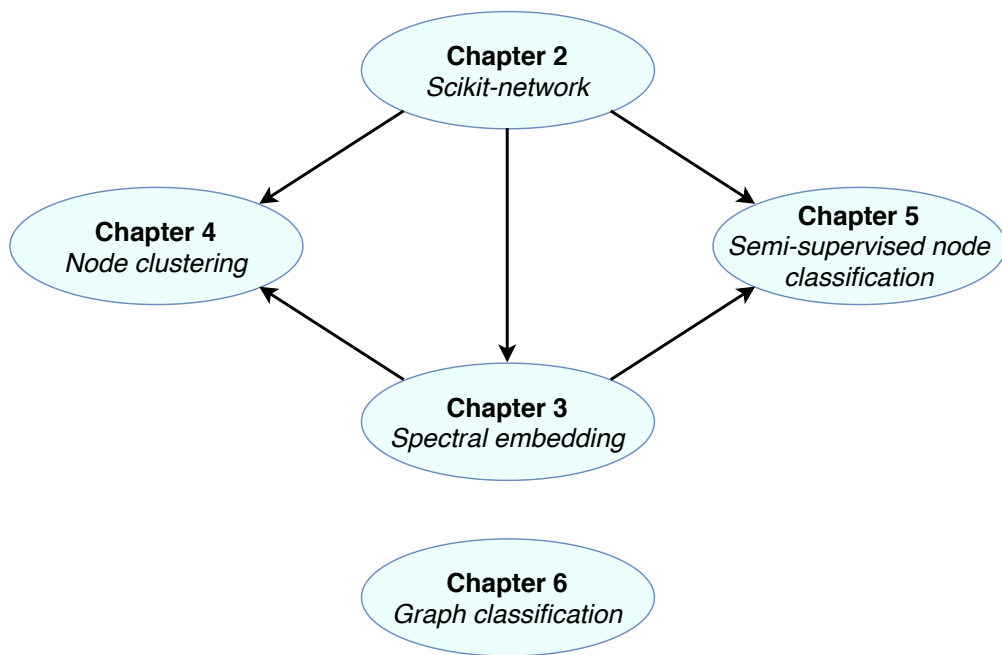


Figure 1.8: Structure of the thesis.

Chapter 2

Scikit-network

Scikit-network (Figure 2.1) is a Python package inspired by Scikit-learn for the analysis of large graphs [Bonald et al., 2020]. Graphs are represented by their adjacency matrix in the sparse CSR format of SciPy. The package provides state-of-the-art algorithms for ranking, clustering, classifying, embedding and visualizing the nodes of a graph. High performance is achieved through a mix of fast matrix-vector products (using SciPy), compiled code (using Cython) and parallel processing. The package is distributed under the BSD license, with dependencies limited to NumPy and SciPy. It is compatible with Python 3.6 and newer. Source code, documentation and installation instructions are available online¹. This chapter describes the version 0.18.0 of the package.

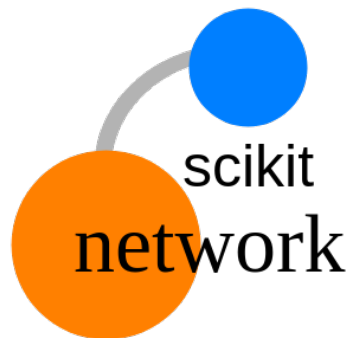


Figure 2.1: Scikit-network’s logo.

References

- Bonald, T., De Lara, N., & Lutz, Q. (2020). *Scikit-network: graph analysis in Python*. Journal of Machine Learning Research.

¹<https://scikit-network.readthedocs.io/en/latest/>

2.1 Motivation

Scikit-learn [Pedregosa et al., 2011b] is a machine learning package based on the popular Python language. It is well-established in today's machine learning community thanks to its versatility, performance and ease of use, making it suitable for both researchers, data scientists and data engineers. Its main assets are the variety of algorithms, the performance of their implementation and their common API.

Scikit-network is a Python package inspired by Scikit-learn for graph analysis. The sparse nature of real graphs, with up to millions of nodes, prevents their representation as dense matrices and rules out most algorithms of Scikit-learn. *Scikit-network* takes as input a sparse matrix in the CSR format of SciPy and provides state-of-the-art algorithms for ranking, clustering, classifying, embedding and visualizing the nodes of a graph.

The design objectives of *scikit-network* are the same as those having made Scikit-learn a success: versatility, performance and ease of use. The result is a Python-native package, like NetworkX [Hagberg et al., 2008], that achieves the state-of-the-art performance of iGraph [Csardi and Nepusz, 2006] and graph-tool [Peixoto, 2014] (see the benchmark in section 2.5). *Scikit-network* uses the same API as Scikit-learn, with algorithms available as classes with the same methods (e.g., `fit`). It is distributed with the BSD license, with dependencies limited to NumPy [Walt et al., 2011] and SciPy [Virtanen et al., 2020].

2.2 Software features

The package is organized in modules with consistent API, covering various tasks:

- **Data.** Module for loading graphs from distant repositories, including Konect [Kunegis, 2013], parsing `tsv` files into graphs, and generating graphs from standard models, like the stochastic block model [Airoldi et al., 2008].
- **Clustering.** Module for clustering graphs, including a soft version that returns a node-cluster membership matrix. It currently implements several variations of the Louvain algorithm [Blondel et al., 2008], as a pipeline to apply k-means on a graph embedding or even a label propagation [Raghavan et al., 2007].
- **Hierarchy.** Module for the hierarchical clustering of graphs, returning dendrograms in the standard format of SciPy. The module also provides various post-processing algorithms for cutting and compressing dendrograms. Implemented algorithms include Paris [Bonald et al., 2018a] and a pipeline to apply Ward's algorithm [Ward Jr, 1963] on a graph embedding.
- **Embedding.** Module for embedding graphs in a space of low dimension. This includes spectral embedding and standard dimension reduction techniques like SVD and GSVD, with key features like regularization. This module also implements the *spring layout* [Fruchterman and Reingold, 1991] for visualization purposes.
- **Ranking.** Module for ranking the nodes of the graph by order of importance. This includes PageRank [Page et al., 1999b] and various centrality scores such as HITS [Kleinberg, 1999], diffusion based-ranking [Chung, 2007], closeness or harmonic centrality [Marchiori and Latora, 2000].
- **Classification.** Module for classifying the nodes of the graph based on the labels of a few

nodes (semi-supervised learning). Some of the implemented algorithms are evaluated in Chapter 5.

- **Path.** Module relying on SciPy for graph traversals: shortest paths, breadth first search, depth first search, diameter estimation, etc.
- **Topology.** Module for topological structures extraction such that connected components, bipartite structure, k -core, clustering coefficient etc.
- **Visualization.** Module for visualizing graphs and dendrograms in SVG (Scalable Vector Graphics) format. Examples are displayed in Figure 2.2.

These modules are only partially covered by existing graph softwares (see Table 2.1). Another interesting feature of *Scikit-network* is its ability to work directly on bipartite graphs, represented by their biadjacency matrix.

Modules	scikit-network	NetworkX	iGraph	graph-tool
Data	✓	✓	✗	✓
Clustering	✓	✓	✓	✗
Hierarchy	✓	✗	✓	✓
Embedding	✓	✓	✗	✓
Ranking	✓	✓	✓	✓
Classification	✓	✓	✗	✗
Path	✓	✓	✓	✓
Topology	✓	✓	✓	✓
Visualization	✓	✓	✓	✓

Table 2.1: Overview of graph software features. ✓: Available. ✓: Partial overlap. ✗: Not available.

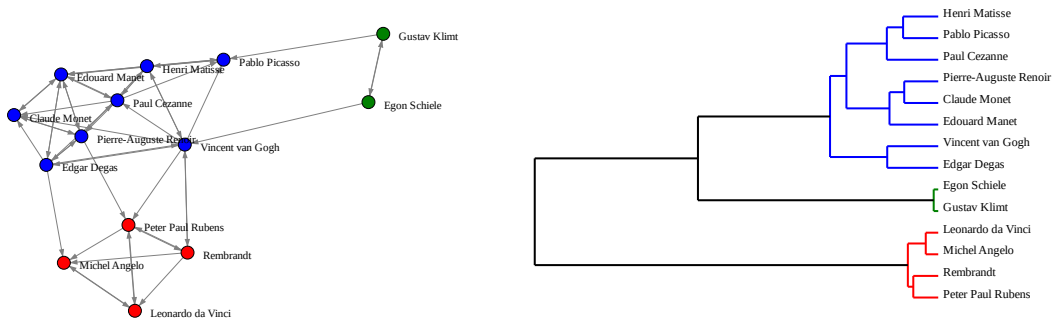


Figure 2.2: Visualization of a graph and a dendrogram as SVG images.

2.3 Project Assets

Code quality and availability. Code quality is assessed by standard code coverage metrics. This version's coverage is greater than 98% for the whole package. Requirements are also kept up to date thanks to the PyUp tool. *Scikit-network* relies on TravisCI for continuous integration and cibuildwheel and manylinux for deploying on common platforms. OSX, Windows 32 or 64-bit and most Linux distributions [McGibbon and Smith, 2016] are supported for Python versions 3.6 and newer.

Open-source software. The package is hosted on GitHub² and part of SciPy kits aimed at creating open-source scientific software. Its BSD license enables maximum interoperability with other software. Guidelines for contributing are described in the package's documentation³ and guidance is provided by the GitHub-hosted Wiki.

Documentation. *Scikit-network* is provided with a complete documentation³. The API reference presents the syntax while the tutorials present applications on real graphs. Algorithms are documented with relevant formulas, specifications, examples and references, when relevant.

Code readability. The source code follows the stringent PEP8 guidelines. Explicit variable naming and type hints make the code easy to read. The number of object types is kept to a minimum.

Data collection. The package offers multiple ways to fetch data. Some small graphs are embedded in the package itself for testing or teaching purposes. Part of the API makes it possible to fetch data from selected graph databases easily. Parsers are also present to enable users to import their own data and save it in a convenient format for later reuse.

2.4 Resources

Scikit-network relies on a very limited number of external dependencies for ease of installation and maintenance. Only SciPy and NumPy are required on the user side.

SciPy. Many elements from SciPy are used for both high performance and simple code. The sparse matrix representations allow for efficient manipulations of large graphs while the linear algebra solvers are used in many algorithms. *Scikit-network* also relies on the *LinearOperator* class for efficient implementation of certain algorithms.

NumPy. NumPy arrays are used through SciPy's sparse matrices for memory-efficient computations. NumPy is used throughout the package for the manipulation of arrays. Some inputs and most of the outputs are given in the NumPy array format.

Cython. In order to speed up execution times, Cython [Behnel et al., 2011] generates C++ files automatically using a Python-like syntax. Thanks to the Python wheel system, no compilation is required from the user on most platforms. Note that Cython has a built-in module for parallel computing on which *scikit-network* relies for some algorithms. Otherwise, it uses Python's native multiprocessing.

²<https://github.com/sknetwork-team/scikit-network>

³<https://scikit-network.readthedocs.io/en/latest/>

2.5 Performance

To show the performance of *scikit-network*, we compare the implementation of some representative algorithms with those of the graph softwares of Table 2.1: the Louvain clustering algorithm [Blondel et al., 2008], PageRank [Page et al., 1999b], HITS [Kleinberg, 1999] and the spectral embedding [Belkin and Niyogi, 2002c]. For PageRank, the number of iterations is set to 100 when possible (that is for all packages except iGraph). For Spectral, the dimension of the embedding space is set to 16.

Table 2.2 gives the running times of these algorithms on the *Orkut* graph of Konect [Kunegis, 2013] in the server setup. The graph has 3 072 441 nodes and 117 184 899 edges. As we can see, *Scikit-network* is highly competitive.

	scikit-network	NetworkX	iGraph	graph-tool
Louvain	139	✗	1 978	✗
PageRank	48	🔥	236	45
HITS	109	🔥	80	144
Spectral	534	🔥	✗	✗

Table 2.2: Execution times (in seconds). ✗: Not available. 🔥: Memory overflow.

We also give in Table 2.3 the memory usage of each package when loading the graph. Thanks to the CSR format, *scikit-network* has a minimal footprint.

	scikit-network	NetworkX	iGraph	graph-tool
RAM usage	1 222	🔥	17 765	10 366

Table 2.3: Memory usage (in MB). 🔥: Memory overflow.

2.6 The NetSet repository

As mentioned in Section 2.2, Scikit-network allows to load datasets from some repositories. Among those is the NetSet⁴ repository which has been implemented by the Scikit-network team.

As in Scikit-learn, datasets are stored as *Bunch* objects with consistent attribute names. Adjacency and biadjacency matrices are stored in the compressed *npz* format of SciPy which is highly efficient both in memory space and loading speed. Other attributes such as labels or names of the nodes are stored in the *numpy* format of NumPy.

The repository currently gathers some standard networks from the literature such as Cora or the 20newsgroup, but also some original ones such as Wikipedia Vitals, already mentioned in Chapter 1. See Example 2.6.1 for an illustration.

⁴<https://netset.telecom-paris.fr/>

Example. 2.6.1: Loading a graph from the NetSet repository

```
[1]: from sknetwork.data import load_netset
      dataset = load_netset('cora')
      dataset
```

```
[1]: {'name': 'cora',
      'labels': array([2, 5, 4, ..., 1, 0, 2]),
      'adjacency': <2708x2708 sparse matrix of type '<class 'numpy.
      ↪float64'>'
           with 10556 stored elements in Compressed Sparse Row format>,
      'meta': {'name': 'cora'}}}
```


Chapter 3

Spectral embedding

Learning from network-structured data often requires to embed the underlying graph in some Euclidian space [Yan et al. \[2006\]](#), [Grover and Leskovec \[2016\]](#), [Bronstein et al. \[2017\]](#). The quality of the subsequent learning tasks (e.g., clustering, classification) then critically depends on this embedding, that must reflect the graph structure.

Usual graph embedding techniques rely on the heuristic that nodes which are close in the graph according to some similarity metric should also be close in the embedding space [[Cai et al., 2018](#)]. Standard similarity metrics include first-order proximity (i.e., edges) [[Belkin and Niyogi, 2002c](#)], second-order proximity [[Roweis and Saul, 2000](#), [Dhillon, 2001](#), [Tang et al., 2015a](#)] or higher-order proximity (i.e., random walks) [[Grover and Leskovec, 2016](#), [Perozzi et al., 2014](#), [Tenenbaum et al., 2000](#)].

The embedding is then the solution to some optimization problem, which is solved either by matrix factorization [[Belkin and Niyogi, 2002c](#), [Dhillon, 2001](#), [Roweis and Saul, 2000](#), [Tenenbaum et al., 2000](#), [Wold et al., 1987](#)] or by iterative methods based on skip-gram negative sampling [[Grover and Leskovec, 2016](#), [Perozzi et al., 2014](#), [Tang et al., 2015a](#)]. However, even for SVD-based methods [[Dhillon, 2001](#), [Wold et al., 1987](#)], there is most of the time no clear interpretation of the distance between nodes in the embedding space.

A classical embedding of *undirected* graphs is based on the spectral decomposition of the Laplacian [[Belkin and Niyogi, 2002c](#), [Ng et al., 2002](#)]; after proper normalization, the distance between nodes in the embedding space corresponds to the mean commute time of a random walk in the graph, making this embedding meaningful and easy to interpret [[Qiu and Hancock, 2007](#), [Fouss et al., 2007](#)].

References

- Bonald, T., & De Lara, N. (2018). *The Forward-Backward Embedding of Directed Graphs*. Openreview.
- De Lara, N., & Bonald, T. (2019). *Spectral embedding of regularized block models*. International Conference on Learning Representations.

3.1 Problem statement

In [Belkin and Niyogi \[2002c\]](#), the embedding is presented as the minimizer of the following loss function:

$$X \in \begin{cases} \arg \min \mathcal{L} = \sum_{i,j} A_{ij} \|X_i - X_j\|^2, \\ X^\top W X = I, \end{cases}$$

where W can be either I or $D = \text{diag}(A\mathbf{1})$. See that the loss function \mathcal{L} penalizes nodes which are connected but distant in the embedding space while the orthonormality constraint rules out the trivial constant embedding.

Yet, the name *spectral* comes from the following equivalent formulation. Given the adjacency matrix $A \in \mathbb{R}_+^{n \times n}$ of the graph, the embedding is obtained by solving either the eigenvalue problem:

$$LX = X\Lambda, \text{ with } X^\top X = I, \quad (3.1)$$

or the generalized eigenvalue problem:

$$LX = DX\Lambda, \text{ with } X^\top DX = I, \quad (3.2)$$

where $L = D - A$ is the Laplacian matrix of the graph, $\Lambda \in \mathbb{R}^{k \times k}$ is the diagonal matrix of the k smallest (generalized) eigenvalues of L and $X \in \mathbb{R}^{n \times k}$ is the corresponding matrix of (generalized) eigenvectors.

Besides, the spectral embedding can be interpreted as equilibrium states of some physical systems [[Snell and Doyle, 2000](#), [Spielman, 2007](#), [Bonald et al., 2018b](#)] as illustrated in [Figure 3.1](#). In this virtual mechanical system, nodes are masses connected by springs corresponding to edges, sliding without friction on an axis rotating at angular speed ω . Let W denote the diagonal matrix of node masses and x the position of the nodes along the axis. The system is in equilibrium if and only if

$$Lx = \omega^2 Wx.$$

If all nodes have the same mass, $W \propto I$ and x is a solution of [\(3.1\)](#) associated with the eigenvalue $\lambda = \omega^2$. If nodes have masses proportional to their degree, $W \propto D$ and x is a solution of [\(3.2\)](#) for the same eigenvalue.

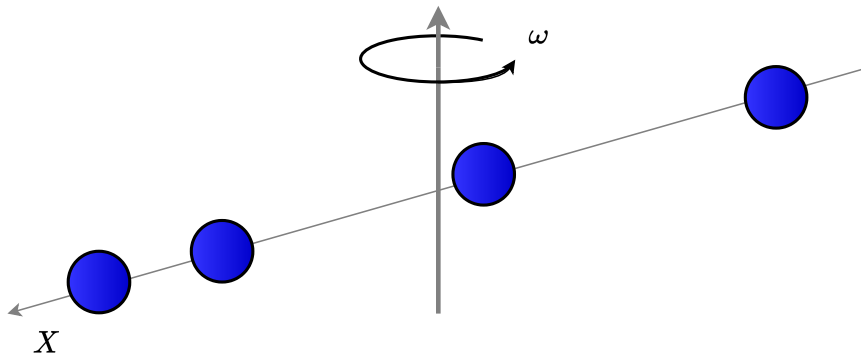


Figure 3.1: Virtual mechanical system.

In this chapter, unless otherwise specified, we consider the generalized eigenvalue problem. In view of proposition 1 the solution of (3.2) is given by the spectral decomposition of the normalized Laplacian matrix

$$L_{\text{norm}} = I - D^{-1/2}AD^{-1/2}.$$

This choice will be justified by our experiments. We refer to the eigenvectors of this normalized Laplacian as the *vanilla spectral embedding*.

In the following sections, we discuss how to improve this vanilla embedding. Specifically, Section 3.2 presents a new interpretation of a standard graph regularization technique, published in Lara and Bonald [2020] and Section 3.3 offers an interpretable extension to directed and bipartite graphs, partially introduced in Bonald and De Lara [2018].

Graphs embeddings are traditionally combined with k-means [MacQueen et al., 1967] for clustering, Ward’s algorithm [Ward Jr, 1963] for hierarchical clustering or some classifier such as a nearest neighbor’s one for classification. Thus, we defer some related experiments to Chapters 4 and 5, respectively.

3.2 Regularization

Vanilla spectral embedding tends to produce poor results on real datasets if applied directly on the graph [Amini et al., 2013]. One reason is that real graphs are often either disconnected or have sets of dangling nodes loosely connected to the rest of the graph due to noise or outliers in the dataset. In this section, we analyze how a simple graph regularization can help to overcome this issue.

Several regularization techniques have been proposed to improve the quality of the embedding with respect to downstream tasks like clustering. In this section, we explain on a simple block model the impact of the complete graph regularization, whereby a constant is added to all entries of the adjacency matrix. Specifically, we show that the regularization forces the spectral embedding to focus on the largest blocks, making the representation less sensitive to noise or outliers. We illustrate these results on both synthetic and real data, showing how regularization improves standard clustering scores.

In order to improve the quality of the embedding, two main types of regularization have been proposed. The first artificially increases the degree of each node by a constant factor [Chaudhuri et al., 2012, Qin and Rohe, 2013], while the second adds a constant to all entries of the original adjacency matrix [Amini et al., 2013, Joseph et al., 2016, Zhang and Rohe, 2018]. See Figure 3.2 for an illustration. In the practically interesting case where the original adjacency matrix A is sparse, the regularized adjacency matrix is dense but has a so-called sparse + low rank structure, enabling the computation of the spectral embedding on very large graphs as presented in Section 1.4.

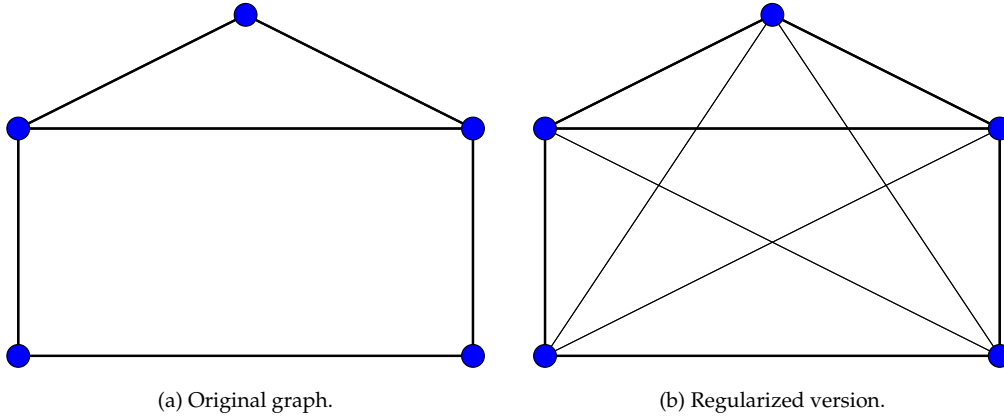


Figure 3.2: Regularization of a toy graph.

While [Zhang and Rohe \[2018\]](#) explains the effects of regularization through graph conductance and [Joseph et al. \[2016\]](#) through eigenvector perturbation on the Stochastic Block Model, there is no simple interpretation of the benefits of graph regularization. In this section, we show on a simple block model that the complete graph regularization forces the spectral embedding to separate the blocks in decreasing order of size, making the embedding less sensitive to noise or outliers in the data.

Indeed, [Zhang and Rohe \[2018\]](#) identified that, without regularization, the cuts corresponding to the first dimensions of the spectral embedding tend to separate small sets of nodes, so-called dangling sets, loosely connected to the rest of the graph. Our work shows more explicitly that regularization forces the spectral embedding to focus on the largest clusters. Moreover, our analysis involves some explicit characterization of the eigenvalues, allowing us to quantify the impact of the regularization parameter.

The rest of this section is organized as follows. Section 3.2.1 presents block models and an important preliminary result about their aggregation. Section 3.2.2 presents the main result, about the regularization of block models, while Section 3.2.5 extends this result to bipartite graphs.

3.2.1 Aggregation of Block Models

Let $A \in \mathbb{R}_+^{n \times n}$ be the adjacency matrix of an undirected, weighted graph, that is a symmetric matrix such that $A_{ij} > 0$ if and only if there is an edge between nodes i and j , with weight A_{ij} . Assume that the n nodes of the graph can be partitioned into K blocks of respective sizes n_1, \dots, n_K so that any two nodes of the same block have the same neighborhood, i.e., the corresponding rows (or columns) of A are the same. Without any loss of generality, we assume that the matrix A has rank K . We refer to such a graph as a block model.

Let $Z \in \mathbb{R}^{n \times K}$ be the associated membership matrix, with $Z_{ij} = 1$ if index i belongs to block j and 0 otherwise. We denote by $W = Z^\top Z \in \mathbb{R}^{K \times K}$ the diagonal matrix of block sizes.

Now define $\bar{A} = Z^\top A Z \in \mathbb{R}^{K \times K}$. This is the adjacency matrix of the aggregate graph, where each block of the initial graph is replaced by a single node; two nodes in this graph are con-

nected by an edge of weight equal to the total weight of edges between the corresponding blocks in the original graph. We denote by $\bar{D} = \text{diag}(\bar{A}1_K)$ the degree matrix and by $\bar{L} = \bar{D} - \bar{A}$ the Laplacian matrix of the aggregate graph.

The following result shows that the solution to the generalized eigenvalue problem (3.2) follows from that of the aggregate graph:

Proposition 3. *Let x be a solution to the generalized eigenvalue problem:*

$$Lx = \lambda Dx. \quad (3.3)$$

Then either $Z^\top x = 0$ and $\lambda = 1$ or $x = Zy$ where y is a solution to the generalized eigenvalue problem:

$$\bar{L}y = \lambda \bar{D}y. \quad (3.4)$$

Proof. Consider the following reformulation of the generalized eigenvalue problem (3.3):

$$Ax = Dx(1 - \lambda). \quad (3.5)$$

Since the rank of A is equal to K , there are $n - K$ eigenvectors x associated with the eigenvalue $\lambda = 1$, each satisfying $Z^\top x = 0$. By orthogonality, the other eigenvectors satisfy $x = Zy$ for some vector $y \in \mathbb{R}^K$. We get:

$$AZy = DZy(1 - \lambda),$$

so that

$$\bar{A}y = \bar{D}y(1 - \lambda).$$

Thus y is a solution to the generalized eigenvalue problem (3.4). \square

3.2.2 Regularization of Block Models

Let A be the adjacency matrix of some undirected graph. We consider a regularized version of the graph where an edge of weight α is added between all pairs of nodes, for some constant $\alpha > 0$. The corresponding adjacency matrix is given by:

$$A_\alpha = A + \alpha J,$$

where $J = 1_n 1_n^\top$ is the all-ones matrix of same dimension as A . We denote by $D_\alpha = \text{diag}(A_\alpha 1_n)$ the corresponding degree matrix and by $L_\alpha = D_\alpha - A_\alpha$ the Laplacian matrix.

We first consider a simple block model where the graph consists of K disjoint cliques of respective sizes $n_1 > n_2 > \dots > n_K$ nodes, with $n_K \geq 1$. In this case, we have $A = ZZ^\top$, where Z is the membership matrix.

The objective of this section is to demonstrate that, in this setting, the k -th dimension of the spectral embedding isolates the $k - 1$ largest cliques from the rest of the graph, for any $k \in \{2, \dots, K\}$

Lemma 1. *Let $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ be the eigenvalues associated with the generalized eigenvalue problem:*

$$L_\alpha x = \lambda D_\alpha x. \quad (3.6)$$

We have $\lambda_1 = 0 < \lambda_2 \leq \dots \leq \lambda_K < \lambda_{K+1} = \dots = \lambda_n = 1$.

Proof. Since the Laplacian matrix L_α is positive semi-definite, all eigenvalues are non-negative [Chung, 1997]. We know that the eigenvalue 0 has multiplicity 1 on observing that the regularized graph is connected. Now for any vector x ,

$$x^\top A_\alpha x = x^\top A x + \alpha x^\top J x = \|Z^\top x\|^2 + \alpha (1_n^\top x)^2 \geq 0,$$

so that the matrix A_α is positive semi-definite. In view of (3.5), this shows that $\lambda \leq 1$ for any eigenvalue λ . The proof then follows from Proposition 3, on observing that the eigenvalue 1 has multiplicity $n - K$. \square

Lemma 2. *Let x be a solution to the generalized eigenvalue problem (3.6) with $\lambda \in (0, 1)$. There exists some $s \in \{+1, -1\}$ such that for each node i in block j ,*

$$\text{sign}(x_i) = s \iff n_j \geq \alpha \frac{1 - \lambda}{\lambda} n.$$

Proof. In view of Proposition 3, we have $x = Zy$ where y is a solution to the generalized eigenvalue problem of the aggregate graph, with adjacency matrix:

$$\bar{A}_\alpha = Z^\top A_\alpha Z = Z^\top (A + \alpha J) Z.$$

Since $A = ZZ^\top$ and $W = Z^\top Z$, we have $\bar{A}_\alpha = W^2 + \alpha Z^\top J Z$. Using the fact that $Z1_K = 1_n$, we get $J = 1_n 1_n^\top = Z J_K Z^\top$ with $J_K = 1_K 1_K^\top$ the all-ones matrix of dimension $K \times K$, so that:

$$\bar{A}_\alpha = W(I_K + \alpha J_K)W,$$

where I_K is the identity matrix of dimension $K \times K$. We deduce the degree matrix:

$$\bar{D}_\alpha = W(W + \alpha n I_K),$$

and the Laplacian matrix:

$$\bar{L}_\alpha = \bar{D}_\alpha - \bar{A}_\alpha = \alpha W(n I_K - J_K W).$$

The generalized eigenvalue problem associated with the aggregate graph is:

$$\bar{L}_\alpha y = \lambda \bar{D}_\alpha y.$$

We get:

$$\alpha(n I_K - J_K W)y = \lambda(W + \alpha n I_K)y.$$

Observing that $J_K W y \propto 1_K$, we conclude that:

$$(\alpha n(1 - \lambda) - \lambda W)y \propto 1_K, \tag{3.7}$$

and since $W = \text{diag}(n_1, \dots, n_K)$,

$$\forall j = 1, \dots, K, \quad y_j \propto \frac{1}{\lambda n_j - \alpha(1 - \lambda)n}. \tag{3.8}$$

The result then follows from the fact that $x = Zy$. \square

Lemma 3. *The K smallest eigenvalues satisfy:*

$$0 = \lambda_1 < \mu_1 < \lambda_2 < \mu_2 < \cdots < \lambda_K < \mu_K,$$

where for all $j = 1, \dots, K$,

$$\mu_j = \frac{\alpha n}{\alpha n + n_j}.$$

Proof. We know from Lemma 1 that the K smallest eigenvalues are in $[0, 1)$. Let x be a solution to the generalized eigenvalue problem (3.6) with $\lambda \in (0, 1)$. We know that $x = Zy$ where y is an eigenvector associated with the same eigenvalue λ for the aggregate graph. Since 1_K is an eigenvector for the eigenvalue 0, we have $y^\top \bar{D}_\alpha 1_K = 0$. Using the fact that $\bar{D}_\alpha = W(W + \alpha n I_K)$, we get:

$$\sum_{j=1}^K n_j(n_j + \alpha n)y_j = 0.$$

We then deduce from (3.7) and (3.8) that $\lambda \notin \{\mu_1, \dots, \mu_K\}$ and

$$\sum_{j=1}^K n_j(n_j + \alpha n) \frac{1}{\lambda/\mu_j - 1} = 0.$$

This condition cannot be satisfied if $\lambda < \mu_1$ or $\lambda > \mu_K$ as the terms of the sum would be either all positive or all negative.

Now let y' be another eigenvector for the aggregate graph, with $y'^\top \bar{D}_\alpha y' = 0$, for the eigenvalue $\lambda' \in (0, 1)$. By the same argument, we get:

$$\sum_{j=1}^K n_j(n_j + \alpha n)y_j y'_j = 0,$$

and

$$\sum_{j=1}^K n_j(n_j + \alpha n) \frac{1}{\lambda/\mu_j - 1} \frac{1}{\lambda'/\mu_j - 1} = 0.$$

with $\lambda' \notin \{\mu_1, \dots, \mu_K\}$. This condition cannot be satisfied if λ and λ' are in the same interval (μ_j, μ_{j+1}) for some j as the terms in the sum would be all positive. There are $K - 1$ eigenvalues in $(0, 1)$ for $K - 1$ such intervals, that is one eigenvalue per interval. \square

The main result of the section is the following, showing that the $k - 1$ largest cliques of the original graph can be recovered from the spectral embedding of the regularized graph in dimension k .

Theorem 1. *Let X be the spectral embedding of dimension k , as defined by (3.2), for some k in the set $\{2, \dots, K\}$. Then $\text{sign}(X)$ gives the $k - 1$ largest blocks of the graph.*

Proof. Let x be the j -th column of the matrix X , for some $j \in \{2, \dots, k\}$. In view of Lemma 3, this is the eigenvector associated with eigenvalue $\lambda_j \in (\mu_{j-1}, \mu_j)$, so that

$$\alpha \frac{1 - \lambda_j}{\lambda_j} n \in (n_{j-1}, n_j).$$

In view of Lemma 2, all entries of x corresponding to blocks of size n_1, n_2, \dots, n_{j-1} have the same sign, the other having the opposite sign. \square

3.2.3 Illustration

We illustrate Theorem 1 with a toy graph consisting of 3 cliques of respective sizes 5, 3 and 2. We compute the embedding in dimension 2 and report the plots in Figure 3.3.

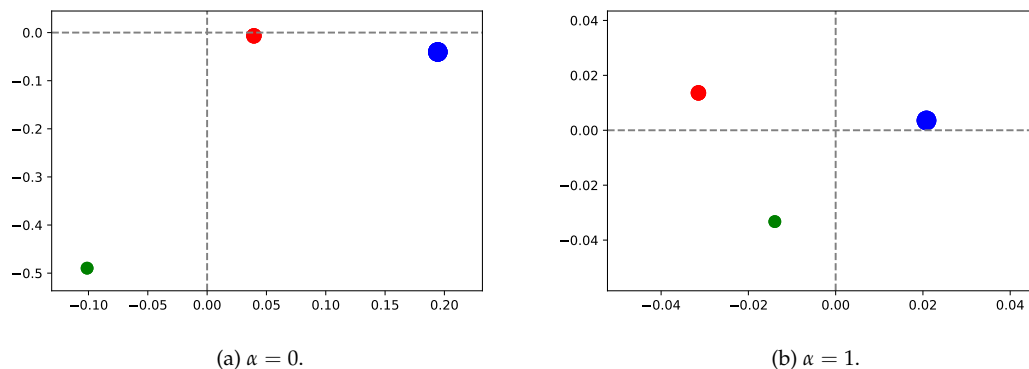


Figure 3.3: Embeddings of a toy graph. The size of a marker is proportional to the size of the corresponding clique.

As we can see, the embedding without regularization isolates the smallest clique from the two others. On the other hand, in the regularized embedding, the first axis isolates the biggest clique from the others and the second axis isolates the two biggest cliques from the small one.

3.2.4 Extensions

Theorem 1 can be extended in several ways. First, the assumption of distinct block sizes can easily be relaxed. If there are L distinct values of block sizes, say m_1, \dots, m_L blocks of sizes $n_1 > \dots > n_L$, there are L distinct values for the thresholds μ_j and thus L distinct values for the eigenvalues λ_j in $[0, 1)$, the multiplicity of the j -th smallest eigenvalue being equal to m_j . The spectral embedding in dimension k still gives $k - 1$ cliques of the largest sizes.

Secondly, the graph may have edges between blocks. Taking $A = ZZ^\top + \varepsilon J$ for instance, for some parameter $\varepsilon \geq 0$, the results are exactly the same, with α replaced by $\varepsilon + \alpha$. A key observation is that regularization really matters when $\varepsilon \rightarrow 0$, in which case the initial graph becomes disconnected and, in the absence of regularization, the spectral embedding may isolate small connected components of the graph. In particular, the regularization makes the spectral embedding much less sensitive to noise, as will be demonstrated in the experiments.

Finally, degree correction can be added by varying the node degrees within blocks. Taking $A = \theta ZZ^\top \theta$, for some arbitrary diagonal matrix θ with positive entries, similar results can be obtained under the regularization $A_\alpha = A + \alpha \theta J \theta$. Interestingly, the spectral embedding in dimension k then recovers the $k - 1$ largest blocks in terms of *normalized weight*, the ratio of the total weight of the block to the number of nodes in the block. For example, a block with 5 nodes connected by edges of weight 10 has a normalized weight of $(5^2 \times 10)/5 = 50$ while a block of 10 nodes connected by edges of weight 4 has a normalized weight of 40. Such case is likely to happen for communication networks. There can be a large group of connected individuals with small communication rate and a much smaller group but with a very high communication rate.

3.2.5 Regularization of Bipartite Graphs

Let $B = \mathbb{R}_+^{n_1 \times n_2}$ be the biadjacency matrix of some bipartite graph with respectively n_1, n_2 nodes in each part, i.e., $B_{ij} > 0$ if and only if there is an edge between node i in the first part of the graph and node j in the second part of the graph, with weight B_{ij} . This is an undirected graph of $n_1 + n_2$ nodes with adjacency matrix:

$$A = \begin{bmatrix} 0 & B \\ B^\top & 0 \end{bmatrix}$$

The spectral embedding of the graph (3.2) can be written in terms of the biadjacency matrix as follows:

$$\begin{cases} BX_2 = D_1 X_1 (I - \Lambda) \\ B^\top X_1 = D_2 X_2 (I - \Lambda) \end{cases} \quad (3.9)$$

where X_1, X_2 are the embeddings of each part of the graph, with respective dimensions $n_1 \times k$ and $n_2 \times k$, $D_1 = \text{diag}(B1_{n_2})$ and $D_2 = \text{diag}(B^\top 1_{n_1})$. In particular, the spectral embedding of the graph follows from the generalized SVD of the biadjacency matrix B .

The complete regularization adds edges between all pairs of nodes, breaking the bipartite structure of the graph. Another approach consists in applying the regularization to the biadjacency matrix, i.e., in considering the regularized bipartite graph with biadjacency matrix:

$$B_\alpha = B + \alpha J,$$

where $J = 1_{n_1} 1_{n_2}^\top$ is here the all-ones matrix of same dimension as B . The spectral embedding of the regularized graph is that associated with the adjacency matrix:

$$A_\alpha = \begin{bmatrix} 0 & B_\alpha \\ B_\alpha^\top & 0 \end{bmatrix} \quad (3.10)$$

As in Section 3.2.2, we consider a block model so that the biadjacency matrix B is block-diagonal with all-ones block matrices on the diagonal. Each part of the graph consists of K groups of nodes of respective sizes $n_{1,1} > \dots > n_{1,K}$ and $n_{2,1} > \dots > n_{2,K}$, with nodes of block j in the first part connected only to nodes of block j in the second part, for all $j = 1, \dots, K$.

We consider the generalized eigenvalue problem (3.6) associated with the above matrix A_α . In view of (3.9), this is equivalent to the generalized SVD of the regularized biadjacency matrix B_α . We have the following results:

Lemma 4. *Let $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ be the eigenvalues associated with the generalized eigenvalue problem (3.6). We have $\lambda_1 = 0 < \lambda_2 \leq \dots \leq \lambda_K < \lambda_{K+1} = \dots = \lambda_{n-2K} < \dots < \lambda_n = 2$.*

Lemma 5. *Let x be a solution to the generalized eigenvalue problem (3.6) with $\lambda \in (0, 1)$. There exists $s_1, s_2 \in \{+1, -1\}$ such that for each node i in block j of part $p \in \{1, 2\}$,*

$$\text{sign}(x_i) = s_p \iff \frac{n_{1,j} n_{2,j}}{(n_{1,j} + \alpha n_1)(n_{2,j} + \alpha n_2)} \geq 1 - \lambda.$$

Lemma 6. *The K smallest eigenvalues satisfy:*

$$0 = \lambda_1 < \mu_1 < \lambda_2 < \mu_2 < \dots < \lambda_K < \mu_K,$$

where for all $j = 1, \dots, K$,

$$\mu_j = 1 - \frac{n_{1,j} n_{2,j}}{(n_{1,j} + \alpha n_1)(n_{2,j} + \alpha n_2)}.$$

Theorem 2. Let X be the spectral embedding of dimension k , as defined by (3.2), for some k in the set $\{2, \dots, K\}$. Then $\text{sign}(X)$ gives the $k - 1$ largest blocks of each part of the graph.

Like Theorem 1, the assumption of decreasing block sizes can easily be relaxed. Assume that block pairs are indexed in decreasing order of μ_j . Then the spectral embedding of dimension k gives the $k - 1$ first block pairs for that order. It is interesting to notice that the order now depends on α :

- When $\alpha \rightarrow 0^+$, the block pairs j of highest value $(\frac{n_1}{n_{1,j}} + \frac{n_2}{n_{2,j}})^{-1}$ (equivalently, highest harmonic mean of proportions of nodes in each part of the graph) are isolated first.
- When $\alpha \rightarrow +\infty$, the block pairs j of highest value $\frac{n_{1,j}n_{2,j}}{n_1n_2}$ (equivalently, the highest geometric mean of proportions of nodes in each part of the graph) are isolated first.

The results also extend to non-block diagonal biadjacency matrices B and degree-corrected models, as for Theorem 1.

The proof of Theorem 2 follows the same workflow as that of Theorem 1. Let $Z_1 \in \mathbb{R}^{n_1 \times K}$ and $Z_2 \in \mathbb{R}^{n_2 \times K}$ be the left and right membership matrices for the block matrix $B \in \mathbb{R}^{n_1 \times n_2}$. The aggregated matrix is $\bar{B} = Z_1^\top B Z_2 \in \mathbb{R}^{K \times K}$. The diagonal matrices of block sizes are $W_1 = Z_1^\top Z_1$ and $W_2 = Z_2^\top Z_2$. We have the equivalent of Proposition 3:

Proposition 4. Let x_1, x_2 be a solution to the generalized singular value problem:

$$\begin{cases} Bx_2 = \sigma D_1 x_1 \\ B^\top x_1 = \sigma D_2 x_2 \end{cases}$$

Then either $Z_1^\top x_1 = Z_2^\top x_2 = 0$ and $\sigma = 0$ or $x_1 = Z_1 y_1$ and $x_2 = Z_2 y_2$ where y_1, y_2 is a solution to the generalized singular value problem:

$$\begin{cases} \bar{B}y_2 = \sigma \bar{D}_1 y_1, \\ \bar{B}^\top y_1 = \sigma \bar{D}_2 y_2. \end{cases}$$

Proof. Since the rank of B is equal to K , there are $n - K$ pairs of singular vectors (x_1, x_2) associated with the singular values 0, each satisfying $Z_1^\top x_1 = 0$ and $Z_2^\top x_2 = 0$. By orthogonality, the other pairs of singular vectors satisfy $x_1 = Z_1 y_1$ and $x_2 = Z_2 y_2$ for some vectors $y_1, y_2 \in \mathbb{R}^K$. By replacing these in the original generalized singular value problem, we get that (y_1, y_2) is a solution to the generalized singular value problem for the aggregate graph. \square

In the following, we focus on the block model described in Section 3.2.5, where $B = Z_1 Z_2^\top$.

Proof of Lemma 4. The generalized eigenvalue problem (3.6) associated with the regularized matrix A_α is equivalent to the generalized SVD of the regularized biadjacency matrix B_α :

$$\begin{cases} B_\alpha x_2 = \sigma D_{\alpha,1} x_1 \\ B_\alpha^\top x_1 = \sigma D_{\alpha,2} x_2, \end{cases}$$

with $\sigma = 1 - \lambda$.

In view of Proposition 4, the singular value $\sigma = 0$ has multiplicity $n - K$, meaning that the eigenvalue $\lambda = 1$ has multiplicity $n - K$. Since the graph is connected, the eigenvalue 0 has multiplicity 1. The proof then follows from the observation that if (x_1, x_2) is a pair of singular vectors for the singular value σ , then the vectors $x = (x_1, \pm x_2)^\top$ are eigenvectors for the eigenvalues $1 - \sigma, 1 + \sigma$.

Proof of Lemma 5. By Proposition 4, we can focus on the generalized singular value problem for the aggregate graph:

$$\begin{cases} \bar{B}_\alpha y_2 = \sigma \bar{D}_{\alpha,1} y_1 \\ \bar{B}_\alpha^\top y_1 = \sigma \bar{D}_{\alpha,2} y_2, \end{cases}$$

Since

$$\bar{B}_\alpha = W_1(I_K + \alpha J_K)W_2,$$

and

$$\begin{cases} \bar{D}_{\alpha,1} = W_1(W_2 + \alpha n_1 I), \\ \bar{D}_{\alpha,2} = W_2(W_1 + \alpha n_2 I), \end{cases}$$

we have:

$$\begin{cases} W_1(I_K + \alpha J_K)W_2 y_2 = W_1(W_2 + \alpha n_1 I)y_1 \sigma, \\ W_2(I_K + \alpha J_K)W_1 y_1 = W_2(W_1 + \alpha n_2 I)y_2 \sigma. \end{cases}$$

Observing that $J_K W_1 y_1 \propto 1_K$ and $J_K W_2 y_2 \propto 1_K$, we get:

$$\begin{cases} (W_2 + \alpha n_2 I_K)y_1 \sigma - W_2 y_2 \propto 1_K, \\ (W_1 + \alpha n_1 I_K)y_2 \sigma - W_1 y_1 \propto 1_K. \end{cases}$$

As two diagonal matrices commute, we obtain:

$$\begin{cases} (W_1 + \alpha n_1 I_K)(W_2 + \alpha n_2 I_K)y_1 \sigma - W_1 W_2 y_1 = (\eta_1(W_1 + \alpha n_1 I_K) + \eta_2 W_2)1_K, \\ (W_1 + \alpha n_1 I_K)(W_2 + \alpha n_2 I_K)y_2 \sigma - W_1 W_2 y_2 = (\eta_1 W_1 + \eta_2(W_2 + \alpha n_2 I_K))1_K, \end{cases}$$

for some constants η_1, η_2 , and

$$\begin{cases} y_{1,j} = \frac{\eta_1(n_{1,j} + \alpha n_1) + \eta_2 n_{2,j}}{(n_{1,j} + \alpha n_1)(n_{2,j} + \alpha n_2)\sigma - n_{1,j}n_{2,j}}, \\ y_{2,j} = \frac{\eta_1 n_{1,j} + \eta_2(n_{2,j} + \alpha n_2)}{(n_{1,j} + \alpha n_1)(n_{2,j} + \alpha n_2)\sigma - n_{1,j}n_{2,j}}. \end{cases}$$

Letting $s_1 = -\text{sign}(\eta_1(n_{1,j} + \alpha n_1) + \eta_2 n_{2,j})$ and $s_2 = -\text{sign}(\eta_1 n_{1,j} + \eta_2(n_{2,j} + \alpha n_2))$, we get:

$$\text{sign}(y_{1,j}) = s_1 \iff \text{sign}(y_{2,j}) = s_2 \iff \frac{n_{1,j}n_{2,j}}{(n_{1,j} + \alpha n_1)(n_{2,j} + \alpha n_2)} \geq \sigma = 1 - \lambda,$$

and the result follows from the fact that $x_1 = Z_1 y_1$ and $x_2 = Z_2 y_2$.

Proof of Lemma 6. The proof is the same as that of Lemma 3, where the threshold values follow from Lemma 5:

$$\mu_j = 1 - \frac{n_{1,j}n_{2,j}}{(n_{1,j} + \alpha n_1)(n_{2,j} + \alpha n_2)}.$$

Proof of Theorem 2. Let x be the j -th column of the matrix X , for some $j \in \{2, \dots, k\}$. In view of Lemma 6, this is the eigenvector associated with eigenvalue $\lambda_j \in (\mu_{j-1}, \mu_j)$. In view of Lemma 4, all entries of x corresponding to blocks of size $n_{1,1}, n_{1,2}, \dots, n_{1,j-1}$ have the same sign, the other having the opposite sign.

3.3 Interpretable Graph Embedding

Unlike some ad-hoc methods [Funke et al., 2020, Ou et al., 2016, Chen et al., 2007], vanilla spectral embedding is not applicable to *bipartite* or *directed* graphs. Moreover, the interpretation in terms of random walk is valid only for the full embedding, using *all* eigenvectors of the Laplacian. This is not feasible for large graphs (e.g., more than 10 000 nodes); in this case, the dimension of the embedding space *must* be much lower than the number of nodes and there is no clear interpretation of the geometry of the embedding.

In this section, we propose a graph embedding technique based on the generalized singular value decomposition (SVD) [Ewerbring et al., 1990] of the adjacency matrix that can be interpreted in terms of graph structure. Specifically, the distance between nodes in the embedding space corresponds to the distance between their respective neighborhoods in the graph, for some appropriate metric. Moreover, this interpretation is valid in any dimension k , considering the neighborhoods in the best rank- k approximation of the graph rather than in the graph itself. The proposed embedding applies to any type of graphs, including bipartite graphs and directed graphs. It turns out to be equivalent to the spectral embedding of the associate co-neighbor graph, after some appropriate scaling depending on the spectrum of the Laplacian matrix.

The rest of the section is organized as follows. We first present some useful matrix norm in Section 3.3.1 and introduce the *co-neighbor* graph in Section 3.3.2. In Section 3.3.3, we present the proposed embedding technique in the specific case of bipartite graphs, the general cases of directed and undirected graphs being considered in sections 3.3.4 and 3.3.5, respectively. The link with spectral embedding is explained in Section 3.3.6.

3.3.1 Matrix norm

For any positive vector $\alpha \in \mathbb{R}^n$, we denote by $\langle \cdot, \cdot \rangle_\alpha$ the following weighted dot product of \mathbb{R}^n :

$$\forall u, v \in \mathbb{R}^n, \quad \langle u, v \rangle_\alpha = \sum_{i=1}^n \frac{u_i v_i}{\alpha_i}.$$

We refer to the corresponding metric as the α metric. We denote by $\| \cdot \|_\alpha$ the corresponding norm:

$$\|u\|_\alpha^2 = u^\top \text{diag}(\alpha)^{-1} u = \|\text{diag}(\alpha)^{-\frac{1}{2}} u\|^2,$$

where $\| \cdot \|$ is the standard Euclidian norm of \mathbb{R}^n .

For any positive vectors $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}^m$, we define the weighted spectral norm of any matrix M of size $n_1 \times n_2$ for the weight vectors α, β as:

$$\|M\|_{\alpha, \beta} = \sup_{u \in \mathbb{R}^{n_2}: \|u\|_\beta = 1} \|Mu\|_\alpha.$$

Observe that $\|M\|_{\alpha, \beta} = \|\text{diag}(\alpha)^{-\frac{1}{2}} M \text{diag}(\beta)^{-\frac{1}{2}}\|_\sigma$, where $\| \cdot \|_\sigma$ denotes the spectral norm of $\mathbb{R}^{n_1 \times n_2}$, given by the largest singular value. The Froebenius norm is denoted by $\| \cdot \|_F$.

3.3.2 Co-neighbor graph

Consider any non-negative matrix A of size $n_1 \times n_2$. This may be viewed as the biadjacency matrix of a weighted bipartite graph G with n_1 nodes in one part and n_2 nodes in the other.

Equivalently, each row of A may be interpreted as a *data sample* and each column as a *feature*. Thus there are n_1 data samples and n_2 features, each row of A corresponding to the feature vector of a data sample. We denote by $d = A\mathbf{1}$ and $f = A^\top\mathbf{1}$ the weight vectors of data samples and features, respectively, and by $D = \text{diag}(d)$ and $F = \text{diag}(f)$ the corresponding diagonal matrices. If the matrix A is binary, the vectors d and f correspond to the degrees the nodes of each part of the bipartite graph G . We assume that the vectors d and f are positive (i.e., there is no all-zero row or column).

The proposed embedding is related to the spectral embedding of the *co-neighbor* graph. The co-neighbor graph is a weighted graph of n_1 nodes (the data samples) with adjacency matrix:

$$S = AF^{-1}A^\top.$$

The weight between nodes i and j can be interpreted as the *similarity* between the corresponding data samples i and j :

$$S_{ij} = \sum_{l=1}^{n_2} \frac{A_{il}A_{jl}}{f_l} = \frac{d_i d_j}{w} \langle p_i, p_j \rangle_\beta. \quad (3.11)$$

Note that the matrix S is symmetric positive semi-definite and has the same rank than A . Besides, the degree of a node in the co-neighbor graph is the same as in the original graph $S\mathbf{1} = A\mathbf{1} = d$.

3.3.3 Graph embedding

The normalized weight vectors define probability distributions over the data samples and the features, respectively:

$$\alpha = \frac{d}{w}, \quad \beta = \frac{f}{w},$$

where $w = d\mathbf{1} = f\mathbf{1} = \mathbf{1}^\top A\mathbf{1}$ is the total weight of the nodes. We have:

$$\beta^\top = \alpha^\top P, \quad (3.12)$$

where $P = D^{-1}A$. Observe that each row of P gives the feature distribution of a data sample, as a probability distribution over $\{1, \dots, n_2\}$.

Generalized SVD. Consider the generalized SVD of the matrix A associated with the diagonal matrices D and F :

$$\begin{cases} AV = DU\Sigma, \\ A^\top U = FV\Sigma, \end{cases} \quad \text{with} \quad \begin{cases} U^\top DU = I, \\ V^\top FV = I. \end{cases} \quad (3.13)$$

The matrices $U = (u_1, \dots, u_r)$ and $V = (v_1, \dots, v_r)$ are formed by the left and right generalized singular vectors of A associated with the generalized singular values $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$, where r is the rank of A , and $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r)$ is the diagonal matrix of positive generalized singular values. We have:

$$A = DU\Sigma V^\top F.$$

Now for any $k \leq r$, let $U^{(k)}, V^{(k)}$ be the respective restrictions of the matrices U, V to their first k columns, and let $\Sigma^{(k)} = \text{diag}(\sigma_1, \dots, \sigma_k)$. Define the matrix:

$$A^{(k)} = DU^{(k)}\Sigma^{(k)}V^{(k)\top}F.$$

This is an optimal rank- k approximation of the matrix A for the norm $\|\cdot\|_{\alpha,\beta}$ defined in section 3.3.1:

Proposition 5. *We have:*

$$\begin{aligned} \min_{M:\text{rank}(M)=k} \|A - M\|_{\alpha,\beta} &= \|A - A^{(k)}\|_{\alpha,\beta} \\ &= \begin{cases} \sigma_{k+1}w & \text{if } k < r, \\ 0 & \text{if } k = r. \end{cases} \end{aligned}$$

Proof. As the regular SVD provides an optimal low-rank approximation of a matrix, we have:

$$\begin{aligned} \min_{M:\text{rank}(M)=k} \|D^{-\frac{1}{2}}AF^{-\frac{1}{2}} - M\| &= \|D^{-\frac{1}{2}}AF^{-\frac{1}{2}} - \tilde{U}^{(k)}\Sigma\tilde{V}^{(k)\top}\| \\ &= \begin{cases} \sigma_{k+1} & \text{if } k < r, \\ 0 & \text{if } k = r, \end{cases} \end{aligned}$$

where $\tilde{U}^{(k)}, \tilde{V}^{(k)}$ are the restrictions of the matrices \tilde{U}, \tilde{V} to their first k columns. Observing that M has rank k if and only if $D^{\frac{1}{2}}MF^{\frac{1}{2}}$ has rank k , we get:

$$\begin{aligned} \min_{M:\text{rank}(M)=k} \|D^{-\frac{1}{2}}(A - M)F^{-\frac{1}{2}}\| &= \|D^{-\frac{1}{2}}(A - A^{(k)})F^{-\frac{1}{2}}\| \\ &= \begin{cases} \sigma_{k+1} & \text{if } k < r, \\ 0 & \text{if } k = r. \end{cases} \end{aligned}$$

The proof follows from the definition of the norm $\|\cdot\|_{\alpha,\beta}$. □

This is illustrated by Figure 3.4 for some toy graph, where the width of each edge in the low-rank approximation graph corresponds to its weight (the original graph is unweighted). It turns out that the weight vectors d and f are preserved by the low-rank approximation, provided k is larger than the number of connected components of the graph:

Proposition 6. *Let K be the number of connected components of the graph G . We have:*

1. $\sigma_1 = \dots = \sigma_K = 1 > \sigma_{K+1}$,
2. if $K = 1$, then $u_1 = 1/\sqrt{w}$ and $v_1 = 1/\sqrt{w}$,
3. $\alpha^\top u_k = 0$ and $\beta^\top v_k = 0$ for all $k > K$,
4. $A^{(k)}\mathbf{1} = d$ and $A^{(k)\top}\mathbf{1} = f$ for all $k > K$.

Proof. We have,

1. $D^{-1}SU = U\Sigma^2$, where S is the adjacency matrix of the co-neighbor graph. Thus the square generalized singular values are eigenvalues of the stochastic matrix $D^{-1}S$. In particular, we have $\sigma_1 = 1$, and the multiplicity of the generalized singular value 1 is the number of connected components of the co-neighbor graph. This is also the number of connected components of the bipartite graph G , because two nodes are connected in G if and only if they are connected in the co-neighbor graph.
2. If $K = 1$, then the generalized singular value σ_1 has multiplicity 1. It is then sufficient to verify that the vectors $u_1 = 1/\sqrt{w}$ and $v_1 = 1/\sqrt{w}$ satisfy $Au_1 = Dv_1, A^\top v_1 = Fu_1$ and $u_1^\top Du_1 = v_1^\top Fv_1 = 1$.

3. Let $u'_1 = 1/\sqrt{w}$ and $v'_1 = 1/\sqrt{w}$. Since u'_1, v'_1 are generalized singular vectors of A for the singular value 1, there exists some generalized singular decomposition U', V' of A such that u'_1, v'_1 are the respective first columns of U', V' . Now the matrices $D^{\frac{1}{2}}U', F^{\frac{1}{2}}V'$ form a regular singular decomposition of the normalized adjacency matrix $D^{-\frac{1}{2}}AF^{-\frac{1}{2}}$. We deduce that $U'^{(K)} = U^{(K)}G$ and $V'^{(K)} = V^{(K)}H$ for some orthogonal matrices G, H of size $K \times K$. Now for any $k > K$, we have $u_k^\top DU^{(K)} = 0$ and $v_k^\top FV^{(K)} = 0$ so that $u_k^\top DU'^{(K)} = 0$ and $v_k^\top FV'^{(K)} = 0$. In particular, $u_k^\top D1 = 0$ and $v_k^\top F1 = 0$.
4. Writing $A^{(k)} = \sum_{l=1}^k \sigma_l Du_l v_l^\top F$ and using the fact that $v_l^\top F1 = 0$ for all $l > K$, we get $A^{(k)}1 = A1 = d$ for all $k > K$. Similarly, $A^{(k)\top}1 = A^\top 1 = f$.

□

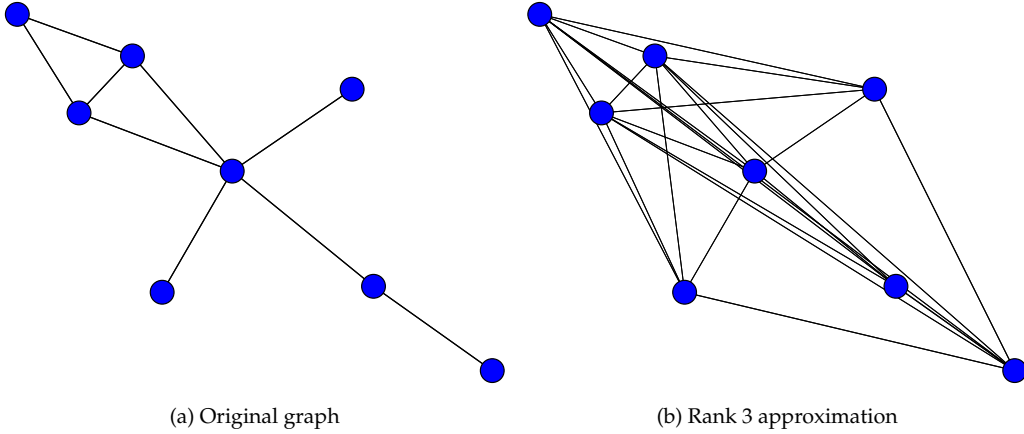


Figure 3.4: Low-rank approximation of a toy graph.

Full embedding. We first consider the full embedding, in dimension r (the rank of A). Define:

$$X = \sqrt{w}(I - 1\alpha^\top)U\Sigma.$$

Each column of X has its center of mass at the origin, for the weight vector α :

$$\alpha^\top X = 0.$$

The embedding is then given by the rows of X . We denote the corresponding vectors by x_1, \dots, x_n . Thus each data sample is embedded as a vector of dimension r .

Interestingly, the features can be embedded in the same space. Let:

$$Y = \sqrt{w}(I - 1\beta^\top)V.$$

Each column of Y has its center of mass at the origin, for the weight vector β :

$$\beta^\top Y = 0.$$

The co-embedding of features is given by the rows of Y . We denote the corresponding vectors by y_1, \dots, y_{n_2} .

Proposition 7. We have:

$$X = PY.$$

Proof. Using (3.12), we get:

$$\begin{aligned} PY &= \sqrt{w}P(I - 1\beta^\top)V, \\ &= \sqrt{w}(I - 1\alpha^\top)PV, \\ &= \sqrt{w}(I - 1\alpha^\top)U\Sigma = X. \end{aligned}$$

□

In view of Proposition 7, we have:

$$x_i = \sum_{j=1}^{n_2} p_{ij}y_j.$$

Thus each data sample i is embedded by some vector x_i which is the weighted average of its features. This is illustrated by Figure 3.5, for a toy bipartite graph connecting movies to actors. See how each movie is at the barycenter of its actors. In particular, two data samples with the same feature vectors (up to some multiplicative constant) have the same embedding.

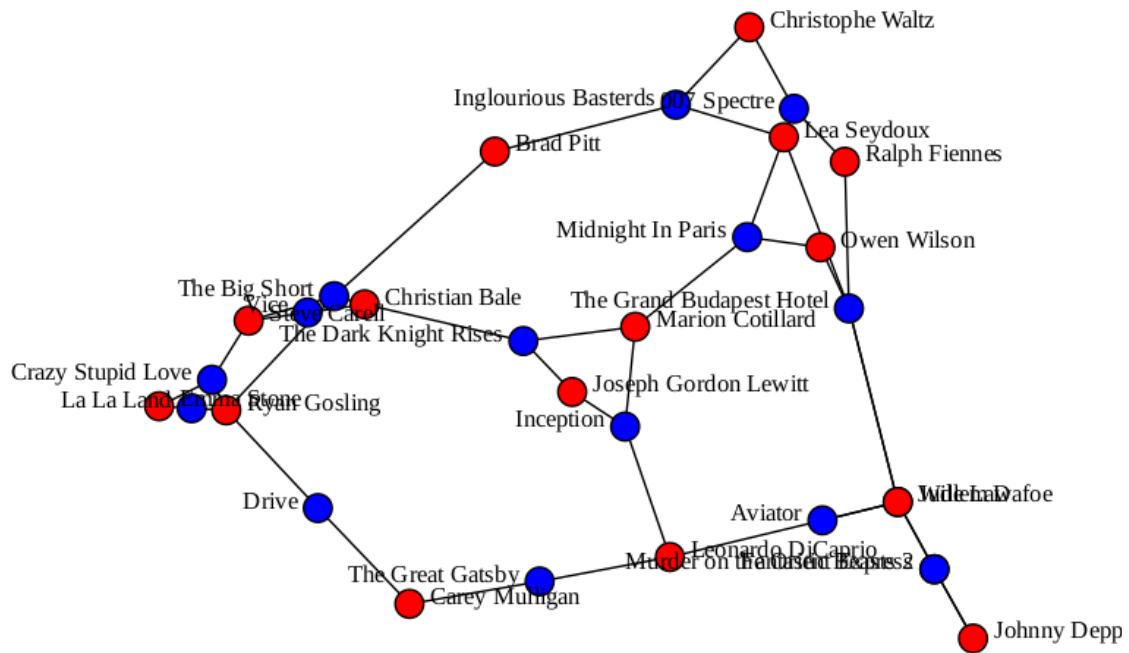


Figure 3.5: Co-embedding of movies (blue) and actors (red).

Geometry of the embedding. As noted earlier, the rows of P can be interpreted as feature distributions. Specifically, the feature distribution of data sample i is given by:

$$p_i = P^\top e_i.$$

In view of (3.12), the weighted average feature distribution is given by:

$$\beta = P^\top \alpha = \sum_{i=1}^{n_1} \alpha_i p_i.$$

The main result of the section is the following.

Theorem 3. *We have for all $i, j = 1, \dots, n_1$:*

$$\langle x_i, x_j \rangle = \langle p_i - \beta, p_j - \beta \rangle_\beta = \langle p_i, p_j \rangle_\beta - 1.$$

Proof. Using the fact that:

$$PF^{-1}P^\top = D^{-1}SD^{-1} = U\Sigma^2U^\top,$$

we get:

$$\begin{aligned} XX^\top &= w(I - 1\alpha^\top)U\Sigma^2U^\top(I - 1\alpha^\top)^\top, \\ &= w(I - 1\alpha^\top)PF^{-1}P^\top(I - 1\alpha^\top)^\top, \\ &= (P - 1\beta^\top)\text{diag}(\beta)^{-1}(P - 1\beta^\top)^\top, \end{aligned}$$

for which the proof follows. \square

Theorem 3, shows that the geometry of the embedding reflects the graph structure. In particular:

Vector norms. We have $\|x_i\|^2 = \|p_i - \beta\|_\beta^2 = \|p_i\|_\beta^2 - 1$. The norm of data sample i in the embedding space is equal to the distance between its feature distribution p_i and the average feature distribution β for the β metric. Data samples that have feature distributions close to the average are close to the origin in the embedding space; data samples that have unusual feature distributions (compared to the average) are far from the origin. Note that $\|x_i\|^2 \leq 1/\min_j \beta_j - 1$.

Distances. We have $\|x_i - x_j\|^2 = \|p_i - p_j\|_\beta^2$. The distance between two data samples in the embedding space is equal to the distance between their respective feature distributions for the β metric.

Angles. The cosine similarity between data samples i and j in the embedding space is given by:

$$s(i, j) = \frac{\langle x_i, x_j \rangle}{\|x_i\| \|x_j\|} = \frac{\langle p_i, p_j \rangle_\beta - 1}{\|p_i - \beta\|_\beta \|p_j - \beta\|_\beta}.$$

In particular, we have $s(i, j) = 1$ if and only if $p_i = p_j$ (same feature distributions) and $s(i, j) < 0$ whenever $\langle p_i, p_j \rangle_\beta = 0$ (disjoint feature distributions).

Illustration Consider the toy bipartite graph displayed in Figure 3.6. The movies, indexed by chronological order, are the feature nodes so that $\beta = (3/8, 3/8, 2/8)$. The sample nodes are the characters. For example, $p_{\text{Greedo}} = (1, 0, 0)$ and $p_{\text{Vador}} = (1/3, 1/3, 1/3)$. So, in the full embedding, we have

$$\begin{aligned} \langle x_{\text{Greedo}}, x_{\text{Vador}} \rangle &= (1 \times \frac{1}{3}) / (\frac{3}{8}) - 1, \\ &= -\frac{1}{9}. \end{aligned}$$

The same derivation with Jabba, $p_{\text{Jabba}} = (1/2, 0, 1/2)$, leads to

$$\begin{aligned}\langle x_{\text{Jabba}}, x_{\text{Vador}} \rangle &= \left(\frac{1}{2} \times \frac{1}{3} \right) \cdot \left(\frac{8}{3} + \frac{8}{2} \right) - 1, \\ &= \frac{1}{9}.\end{aligned}$$

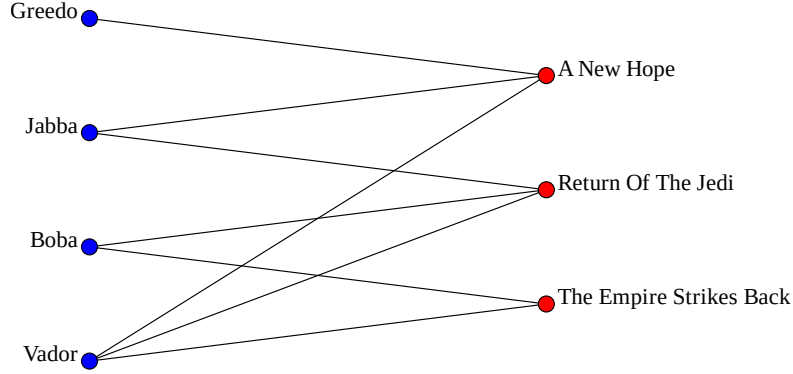


Figure 3.6: A toy bipartite graph.

Embedding in low dimension. Now consider the embedding in dimension k , for any $k \in \{K+1, \dots, r\}$. Let:

$$X^{(k)} = \sqrt{w}(I - 1\alpha^\top)U^{(k)}\Sigma^{(k)}.$$

Again, each column of $X^{(k)}$ has its center of mass at the origin:

$$\alpha^\top X^{(k)} = 0.$$

We embed the data samples as vectors of dimension k , corresponding to the rows of $X^{(k)}$. We denote the corresponding vectors by $x_1^{(k)}, \dots, x_n^{(k)}$.

Similarly, we co-embed the features as vectors of dimension k , corresponding to the rows of

$$Y^{(k)} = \sqrt{w}(I - 1\beta^\top)V^{(k)}.$$

We denote the corresponding vectors by $y_1^{(k)}, \dots, y_m^{(k)}$. We have:

$$X^{(k)} = P^{(k)}Y^{(k)}$$

with $P^{(k)} = D^{-1}A^{(k)}$. The proof is similar to that of Proposition 7, using the following extension of (3.12):

$$\beta^\top = \alpha^\top P^{(k)}. \tag{3.14}$$

Theorem 4. We have for all $i, j = 1, \dots, n$:

$$\langle x_i^{(k)}, x_j^{(k)} \rangle = \langle p_i^{(k)}, p_j^{(k)} \rangle_\beta - 1.$$

Proof. The proof is similar, using (3.14) and the fact that:

$$P^{(k)} F^{-1} P^{(k)\top} = D^{-1} S^{(k)} D^{-1} = U^{(k)} \Sigma^{(k)2} U^{(k)\top}.$$

□

In view of Theorem 4, the geometry of the embedding in dimension k can be interpreted as the full embedding but for the best rank- k approximation of the graph rather than for the graph itself.

Co-neighbor graph. As introduced in Section 3.3.2, this embedding is related to the spectral embedding of the *co-neighbor* graph S .

In view of (3.13), we have:

$$SU = DU\Sigma^2, \quad U^\top DU = I. \quad (3.15)$$

This is the generalized eigenvalue problem of S associated with the diagonal matrix of weights D . Observe that the spectral embedding of the co-neighbor graph is based on the generalized spectral decomposition of the associate Laplacian matrix $L = D - S$. In view of (3.15), its eigenvalues (in non-decreasing order) satisfy:

$$\begin{aligned} \lambda_1 = \dots = \lambda_K = 0 < \lambda_{K+1} = 1 - \sigma_{K+1}^2 \leq \dots \\ \leq \lambda_r = 1 - \sigma_r^2 < \lambda_{r+1} = \dots = \lambda_n = 1. \end{aligned}$$

Thus this embedding corresponds to the spectral embedding of the normalized co-neighbor graph after scaling by $1 - \lambda_1, \dots, 1 - \lambda_n$ and shift so that the center of mass is at the origin. Observe that only eigenvectors associated with eigenvalues less than 1 are kept (the other vanish), so that the dimension of the embedding is at most r .

For most real graphs, the biadjacency matrix A is sparse but the similarity matrix S is too dense to be stored. Consider for instance the node *Beyoncé* in the Twitter graph. This artist has 15.5 million followers, hence the corresponding node has an in-degree of 15.5 million. In the co-neighbor graphs, these followers form a clique of the same size, generating potentially 240 000 billion of nonzero entries in S . However, using techniques introduced in Section 1.4, there is no need to actually store the similarity matrix to compute the embedding.

3.3.4 Case of directed graphs

Consider a directed graph G of n nodes with adjacency matrix A . Observe that the matrix A is square but not symmetric in general. Let A_+ be the reduced adjacency matrix, obtained after removing all-zero rows and columns. The graph G can then be considered as a bipartite graph with biadjacency matrix A_+ . There are n^+ nodes in the first part (nodes of G having positive out-degrees) and n^- nodes in the second part (nodes of G having positive in-degrees). We simply apply the embedding described in section 3.3.3 to this bipartite graph.

The geometry of the embedding can be interpreted using Theorems 3 and 4, with p_i and $p_i^{(k)}$ corresponding to the distributions of successors of node i in the graph G and in its best rank- k approximation, respectively. The embedding also corresponds to the spectral embedding of the co-neighbor graph, after appropriate scaling and shifting. The similarity S_{ij} between two nodes i, j of G depends on their out-degrees and on the weighted dot-product of their distributions of successors p_i, p_j , according to (3.11).

3.3.5 Case of undirected graphs

Consider any undirected graph G of n nodes, with adjacency matrix A . Now A is both square and symmetric. Without loss of generality, we assume that all nodes have positive degrees. Similarly, the graph G may be seen as a bipartite graph with biadjacency matrix A . There are n nodes in each part. We then directly apply the embedding described in section 3.3.3 to this bipartite graph.

Again, the geometry of the embedding can be interpreted using Theorems 3 and 4, with p_i and $p_i^{(k)}$ corresponding to the distributions of successors of node i in the graph G and in its best rank- k approximation, respectively. The embedding also corresponds to the spectral embedding of the co-neighbor graph, after appropriate scaling and shifting. This is *not* the spectral embedding of the graph G . In particular, two nodes with the same neighborhoods have the same representation in the embedding space, which is not the case for the spectral embedding of the graph G .

3.3.6 Link with spectral embedding

We have seen that the proposed embedding is related to the spectral embedding of the co-citation graph, after appropriate scaling and shifting. This is not a practical method, however, as the similarity matrix S is most often too dense to be stored, meaning that the co-neighbor graph cannot be built. In this section, we show that the proposed embedding is also related to the spectral embedding of some extended graph, which can be built in practice.

Let A be any non-negative matrix of size $n_1 \times n_2$, as considered in section 3.3.3. This is the biadjacency matrix of some bipartite graph G . Now consider the extended graph \tilde{G} , with adjacency matrix and degree matrix:

$$\tilde{A} = \begin{pmatrix} 0 & A \\ A^\top & 0 \end{pmatrix}, \quad \tilde{D} = \begin{pmatrix} D & 0 \\ 0 & F \end{pmatrix}.$$

Note that the graph \tilde{G} has $n_1 + n_2$ nodes. In view of (3.13), we have:

$$\tilde{A} \begin{pmatrix} U \\ V \end{pmatrix} = \tilde{D} \begin{pmatrix} U \\ V \end{pmatrix} \Sigma, \quad \text{with} \quad \begin{pmatrix} U \\ V \end{pmatrix}^\top \tilde{D} \begin{pmatrix} U \\ V \end{pmatrix} = I.$$

So the generalized singular values $\sigma_1, \dots, \sigma_r$ of A are the generalized positive eigenvalues of \tilde{A} . The spectral embedding of the graph \tilde{G} is based on the spectral decomposition of the Laplacian matrix $\tilde{L} = \tilde{D} - \tilde{A}$. We get:

$$\tilde{L} \begin{pmatrix} U \\ V \end{pmatrix} = \tilde{D} \begin{pmatrix} U \\ V \end{pmatrix} \Lambda.$$

with $\Lambda = I - \Sigma$. So our embedding corresponds to the spectral embedding of the extended graph restricted to positive generalized eigenvalues, after scaling by $1 - \lambda_1, \dots, 1 - \lambda_r$ and shifting so that the center of mass is at the origin.

In the particular case where A is the adjacency matrix of some undirected graph G , the embedding differs from the spectral embedding of the graph G , which is based on the spectral decomposition of the Laplacian $L = D - A$ (instead of \tilde{L}).

3.4 About embedding metrics

Assessing the quality of an embedding without a supervised downstream task is not trivial. A good embedding should *fit* the data in the sense that nodes which are considered close according to the topology of the graph should also be close in the embedding space. Such fit functions \mathcal{F} include:

- **Spectral:** $\mathcal{F} = -\sum_{i,j} A_{ij} \|x_i - x_j\|^2$.
- **Node2Vec:** $\mathcal{F} = \sum_{(i,j) \in E} x_i^\top x_j$.

However, such functions have either trivial or ill-defined optimums. There are two main strategies to overcome this issue. Spectral methods rely on orthonormality constraints while iterative methods such as [Grover and Leskovec \[2016\]](#) or [Tang et al. \[2015b\]](#) rely on a penalization function \mathcal{D} . For example, in node2vec:

$$\mathcal{D} = -\sum_i \log \left(\sum_j \exp(x_i^\top x_j) \right).$$

This idea is similar to the one used for graph visualization where nodes apply repulsive forces on each other [[Fruchterman and Reingold, 1991](#)]. However, in both cases, the cost of evaluating \mathcal{D} is quadratic in the number of nodes. Node2vec relies on negative sampling in order to maintain tractable computations but this comes at the cost of an extra source of randomness and supplementary hyper-parameters. The final quality function is simply $\mathcal{Q} = \mathcal{F} - \mathcal{D}$.

Here, we propose a different one, as it is inspired by the modularity [[Newman, 2006a](#)] for graph clustering, we refer to it as *cosine modularity*:

$$\mathcal{Q} = \sum_{ij} \left(\frac{A_{ij}}{w} - \gamma \frac{d_i d_j}{w^2} \right) \left(\frac{1 + \cos(x_i, x_j)}{2} \right), \quad (3.16)$$

for some resolution parameter γ which is set to 1 by default. See [Example 3.4.1](#) for an illustration.

```
Example. 3.4.1: Cosine modularity

[1]: from sknetwork.embedding import cosine_modularity
      from sknetwork.data import karate_club

      karate = karate_club(metadata=True)
      A = karate.adjacency
      X = karate.position
      round(cosine_modularity(A, X), 2)

[1]: 0.35
```

This quality function has a few interesting properties:

1. It maintains the standard $\mathcal{Q} = \mathcal{F} - \mathcal{D}$ structure.
2. As for the standard modularity, $\mathcal{Q} \in [-\gamma, 1]$.
3. Its asymptotic complexity is only $O(m)$.

Proof. Let us denote $s_{ij} = \left(\frac{1 + \cos(x_i, x_j)}{2} \right)$.

1. In equation (3.16), we identify:

$$\mathcal{F} = \sum_{ij} \frac{A_{ij}}{w} s_{ij},$$

$$\mathcal{D} = \sum_{ij} \gamma \frac{d_i f_j}{w^2} s_{ij}.$$

2. As $s_{ij} \in [0, 1], \forall (i, j)$,

$$\mathcal{Q} \leq \sum_{ij} \frac{A_{ij}}{w} = 1,$$

$$\mathcal{Q} \geq - \sum_{ij} \gamma \frac{d_i f_j}{w^2} = -\gamma.$$

3. \mathcal{F} is a sum over m elements. For \mathcal{Q} , let us denote by \bar{x} the embedding projected onto the unit-sphere: $\bar{x}_i = x_i / \|x_i\|_2$. Then:

$$\mathcal{D} = \frac{\gamma}{2} \left[1 + \sum_{ij} \frac{d_i f_j}{w^2} \bar{x}_i^\top \bar{x}_j \right],$$

$$= \frac{\gamma}{2} \left[1 + \left(\sum_i \frac{d_i \bar{x}_i}{w} \right)^\top \left(\sum_j \frac{f_j \bar{x}_j}{w} \right) \right].$$

□

The proof of the last point highlights one major limitation of this quality function. It is sufficient for \bar{x} to satisfy $\langle \bar{x}, 1 \rangle_\alpha = 0$ or $\langle \bar{x}, 1 \rangle_\beta = 0$ in order to yield the minimal penalization.

Another approach to quantify the quality of an embedding is self-supervision through link-prediction. The idea is to hide a certain fraction of the edges, embed the nodes, predict the most likely missing edges based on the embedding and compare this prediction to the actually missing edges. The likelihood of an edge is usually either given by $\langle x_i, x_j \rangle$ or $\cos(x_i, x_j)$.

Still, evaluating the performance of a link prediction algorithm is also a challenging task. For example, the brute force computation of the edges sorted by likelihood has a complexity of $O(n^2 \log(n))$ which is prohibitive for large graphs. The selection of the hidden edges and the choice of the metric are also non-trivial. For more information on this topic, we refer to the work of [Yang et al. \[2015\]](#) as this lies beyond the scope of the present thesis.

3.5 Experiments

This section describes part of the experiments related to the embeddings. Sections 3.5.1 and 3.5.2 respectively introduce some metrics and the datasets. Section 3.5.3 justifies the choice of the normalized Laplacian. Section 3.5.4 outlines the influence of graph regularization for spectral clustering introduced in Section 3.2 while Section 3.5.5 studies the influence of the shift in the GSVD embedding. Finally, Section 3.5.6 highlights the importance of normalization for the spectral embedding. We defer general benchmarks to Chapters 4 and 5 respectively in order to focus on the different spectral embeddings in the present section.

3.5.1 Metrics

We consider a large set of metrics from the clustering literature. All metrics are upper-bounded by 1 and the higher the score the better. For supervised metrics, we denote by c the predicted clustering and by y the ground truth labels.

Homogeneity (H), Completeness (C) and V-measure score (V) [Rosenberg and Hirschberg, 2007]. Supervised metrics. A cluster is homogeneous if all its data points are members of a single class in the ground truth. A clustering is complete if all the members of a class in the ground truth belong to the same cluster in the prediction. Then, the V-measure score is the harmonic mean of homogeneity and completeness.

Adjusted Rand Index (ARI) [Hubert and Arabie, 1985]. Supervised metric. This is the corrected for chance version of the Rand Index (equation (3.17)) which is itself an accuracy on pairs of samples. A pair of samples is correctly classified if $\delta_{c_i c_j} = \delta_{y_i y_j}$. This means that either belong to the same cluster in the prediction and in the ground truth or they belong to different clusters in both partitions.

$$\text{RI}(c, y) = \frac{1}{n^2} \sum_{i,j} \mathbb{1}(\delta_{c_i c_j} = \delta_{y_i y_j}). \quad (3.17)$$

Then, the ARI is

$$\text{ARI} = \frac{\text{RI} - \mathbb{E}(\text{RI})}{\max(\text{RI}) - \mathbb{E}(\text{RI})},$$

where the expected value is calculated with respect to a random cluster assignment.

Adjusted Mutual Information (AMI) [Vinh et al., 2010] Supervised metric. Adjusted for chance version of the mutual information.

Fowlkes-Mallows Index (FMI) [Fowlkes and Mallows, 1983]. Supervised metric. Geometric mean between precision and recall on the edge classification task, as described for the ARI.

Modularity (\mathcal{M}) [Newman, 2006a]. Unsupervised metric as in equation (4.3). Fraction of edges within clusters compared to that is some null model where edges are shuffled at random.

Normalized Standard Deviation (NSD) Unsupervised metric. 1 minus normalized standard deviation in cluster size. This metric provides insight on the balance of the clustering.

3.5.2 Datasets

This section describes the datasets used in our experiments. Table 3.1 presents the main features of the graphs.

Stochastic Block-Model (SBM) We generate 100 instances of the same stochastic block model [Holland et al., 1983]. There are 100 blocks of size 20, with intra-block edge probability set to 0.5 for the first 50 blocks and 0.05 for the other blocks. The inter-block edge probability is set to 0.001. Other sets of parameters can be tested using the code available online. The ground-truth cluster of each node corresponds to its block.

20newsgroup (NG) This dataset consists of around 18 000 newsgroups posts on 20 topics. This defines a weighted bipartite graph between documents and words. The label of each document corresponds to the topic.

Wikipedia for Schools (WS) [Haruechaiyasak and Damrongrat, 2008]. This is the graph of hyperlinks between a subset of Wikipedia pages. The label of each page is its category (e.g., countries, mammals, physics).

Cora (CO) and CiteSeer (CS) Citation networks between scientific publications. These are standard datasets for node classification [Fey et al., 2018, Huang et al., 2018, Wijesinghe and Wang, 2019]. Labels correspond to social communities.

Wikipedia Vitals (WV) Graph of hyperlinks between a selection of Wikipedia pages¹, labeled by category (People, History, Geography...).

code	name	n	m	# classes
SBM	Stochastic Block Model	$2 \cdot 10^3$	$5 \cdot 10^3$	100
CO	Cora	$3 \cdot 10^3$	$1 \cdot 10^4$	7
CS	CiteSeer	$3 \cdot 10^3$	$1 \cdot 10^4$	6
NG	20newsgroup	$1 \cdot 10^4$	$2 \cdot 10^6$	19
WS	Wikipedia for Schools	$5 \cdot 10^3$	$2 \cdot 10^5$	14
WV	Wikipedia Vitals	$1 \cdot 10^4$	$1 \cdot 10^6$	11

Table 3.1: Some characteristics of the datasets.

3.5.3 Influence of Laplacian normalization

In this experiment, we justify our choice to favor the normalized Laplacian with respect to the regular one. In order to do so, we compare their performance on a semi-supervised task. We extract 16 eigenvectors and apply a 1-nearest-neighbor classifier with 1% of the nodes, selected uniformly at random, as labeled *seeds*. We do not use regularization nor unit-sphere normalization. For this experiment, all graphs are treated as undirected. The process is repeated 10 times for each graph. We report the resulting macro F1-scores in Table 3.2. Recall that, for a multi-label classification, the macro F1-score is the average of F1-scores obtained for the prediction of each class in a *one-against-all* setting. As we can see, the normalized Laplacian outperforms the regular one by a significant margin.

¹https://en.wikipedia.org/wiki/Wikipedia:Vital_articles/Level/4

Laplacian	CO	CS	WS	WV
regular	0.14 ± 0.01	0.17 ± 0.01	0.06 ± 0.00	0.10 ± 0.00
normalized	0.44 ± 0.05	0.28 ± 0.03	0.29 ± 0.06	0.50 ± 0.02

Table 3.2: Macro F1-scores (mean ± standard deviation).

3.5.4 Influence of regularization

We now illustrate the impact of regularization on the quality of spectral embedding. In all experiments, we skip the first dimension of the spectral embedding as it is not informative (the corresponding eigenvector is the all-ones vector, up to some multiplicative constant).

Experimental setup

All graphs are embedded in dimension 20, with different regularization parameters. To compare the impact of this parameter across different datasets, we use a relative regularization parameter $(w/n^2)\alpha$, where $w = \mathbf{1}_n^\top A \mathbf{1}_n$ is the total weight of the graph, as illustrated in Example 3.5.1.

Example. 3.5.1: Regularized spectral embedding

```
[1]: from sknetwork.embedding import Spectral
spectral = Spectral(n_components=20,
                    normalized_laplacian=True,
                    regularization=0.01,
                    relative_regularization=True)
```

We use the k-means algorithm to cluster the nodes in the embedding space. The parameter k is set to the ground-truth number of clusters. We use the Scikit-learn [Pedregosa et al., 2011a] implementation of the metrics, when available.

Results

We report the results in Table 3.3 for $\alpha \in \{0, 0.1, 1, 10\}$. We see that the regularization generally improves performance, the optimal value of α depending on both the dataset and the score function. As suggested by Lemma 3, the optimal value of the regularization parameter should depend on the distribution of cluster sizes, on which we do not have any prior knowledge.

To test the impact of noise on the spectral embedding, we add isolated nodes with self loop to the graph and compare the clustering performance with and without regularization. The number of isolated nodes is given as a fraction of the initial number of nodes in the graph. Scores are computed only on the initial nodes. The results are reported in Table 3.4 for the Wikipedia for Schools dataset. We observe that, in the absence of regularization, the scores drop even with only 1% noise. The computed clustering is a trivial partition with all initial nodes in the same cluster. This means that the 20 first dimensions of the spectral embedding focus on the isolated nodes. On the other hand, the scores remain approximately constant in the regularized case, which suggests that regularization makes the embedding robust to this type of noise.

SBM								
α	H	C	V	ARI	AMI	FMI	\mathcal{M}	NSD
0	0.19	0.27	0.22	0.0	0.01	0.03	0.45	0.76
0.1	0.33	0.35	0.34	0.0	0.01	0.01	0.52	0.91
1	0.36	0.37	0.36	0.0	0.01	0.01	0.50	0.92
10	0.28	0.34	0.30	0.0	0.00	0.02	0.36	0.78
NG								
α	H	C	V	ARI	AMI	FMI	\mathcal{M}	NSD
0	0.40	0.70	0.51	0.19	0.50	0.34	0.21	0.55
0.1	0.44	0.70	0.54	0.22	0.54	0.35	0.21	0.59
1	0.46	0.67	0.54	0.20	0.54	0.33	0.20	0.60
10	0.37	0.55	0.45	0.13	0.44	0.26	0.17	0.56
WS								
α	H	C	V	ARI	AMI	FMI	\mathcal{M}	NSD
0	0.23	0.29	0.25	0.05	0.25	0.26	0.25	0.49
0.1	0.26	0.29	0.28	0.10	0.27	0.26	0.29	0.61
1	0.23	0.24	0.23	0.04	0.23	0.20	0.30	0.65
10	0.19	0.22	0.20	0.00	0.19	0.20	0.23	0.53

Table 3.3: Impact of regularization on clustering performance.

3.5.5 Influence of embedding shift

The embedding described in Section 3.3.3 is shifted so that its center of mass is at the origin for the weight vector α . In this experiment, we intend to assess whether this shift leads to different results in practice. Note that, this shift does not change the nearest neighbor ordering with respect to the Euclidean distance, however, it might change it with respect to the cosine similarity.

The setup is similar to the one of Section 3.5.3, except that embeddings are normalized onto the unit-sphere so that the nearest neighbor is defined with respect to the cosine similarity instead of the Euclidean distance. In addition to the macro F1-score, we report the mean norm of the nodes before the shift and the norm of the shift vector $\Delta = (U\Sigma)^\top \alpha$ in Table 3.5. As the macro F1-score is not modified by the shift for any of the datasets, we only report it once. This can be surprising as the norm of the shift is not negligible with respect to the average norm of the node vectors. Thus, understanding the influence of this shift requires further investigation. We omit it in further experiments for simplicity.

3.5.6 Cosine similarity versus L2 norm

In this experiment, we highlight the importance of embedding normalization for spectral clustering. Recall that projecting the node vectors onto the unit sphere is equivalent to use the cosine similarity in the embedding space rather than the Euclidean distance.

We embed the nodes of the Cora dataset in dimension 16 and then apply k-means for $k = 5$. If the embedding is not normalized, this yields a V-score of 0.01 while the score is 0.21 for

$\alpha = 0$								
noise	H	C	V	ARI	AMI	FMI	\mathcal{M}	std
0 %	0.23	0.29	0.25	0.05	0.25	0.26	0.25	0.49
1 %	0.00	0.49	0.00	0.00	0.00	0.39	0.00	0
5 %	0.00	0.49	0.00	0.00	0.00	0.39	0.00	0
10 %	0.00	0.49	0.00	0.00	0.00	0.39	0.00	0

$\alpha = 1$								
noise	H	C	V	ARI	AMI	FMI	\mathcal{M}	std
0 %	0.23	0.24	0.23	0.04	0.23	0.2	0.3	0.65
1 %	0.24	0.24	0.24	0.04	0.23	0.2	0.3	0.66
5 %	0.23	0.23	0.23	0.05	0.22	0.2	0.3	0.67
10 %	0.24	0.23	0.23	0.05	0.23	0.2	0.3	0.67

Table 3.4: Impact of noise on clustering performance (WS dataset).

	CO	CS	WS	WV
$\frac{1}{n} \sum \ x_i\ _2$	2.76	3.96	2.43	2.70
$\ \Delta\ _2$	0.35	0.12	1.00	1.00
macro-F1 ($\mu \pm \sigma$)	0.43 ± 0.06	0.26 ± 0.02	0.28 ± 0.01	0.52 ± 0.02

Table 3.5: Some measurements. μ : mean. σ : standard deviation.

the normalized embedding. This is a direct consequence of the cluster sizes. Indeed, without normalization, the NSD of the clustering is 0.04, which means that most of the nodes belong to a unique giant cluster while the four other clusters have very few nodes. With the normalization, the NSD is 0.80, which means that the clusters sizes are very balanced.

The aggregate graphs are displayed in Figure 3.7. Our conclusion is that cosine similarity should be preferred for spectral clustering.

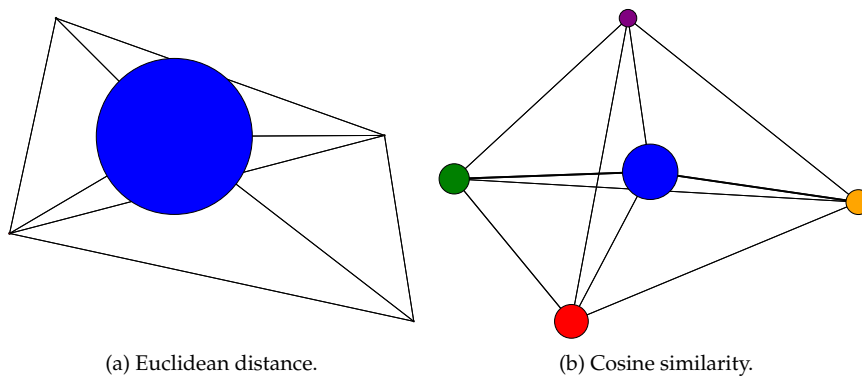


Figure 3.7: Aggregate graphs resulting from spectral clustering. Node sizes are proportional to cluster sizes.

Chapter 4

Node clustering

This chapter is dedicated to node clustering. As mentioned in Chapter 1, this task consists in grouping nodes into a certain number of coherent clusters.

Formally, we denote by $\{C_1, \dots, C_K\}$ a partition of the set of nodes V into K clusters. The associated cluster index is denoted by c i.e. if node i belongs to cluster C_k , then $c_i = k$ while δ is the coclustering variable i.e.,

$$\delta_{c_i c_j} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in the same cluster,} \\ 0 & \text{otherwise.} \end{cases}$$

Finally, $Z \in \mathbb{R}^{n \times K}$ is the membership matrix such that $Z_{ik} = \mathbf{1}(c_i = k)$. In particular,

$$\delta_{c_i c_j} = Z_i^\top Z_j.$$

The rest of this chapter is organized as follows. Section 4.1 presents a family of algorithms based on the Louvain heuristic while Section 4.2 is dedicated to experiments, making the connection with Section 3.3.

Note that, even though we consider graph clustering from a knowledge discovery point of view here, it is also a common tool from *graph coarsening* in graph neural networks [Simonovsky and Komodakis, 2017, Deng et al., 2019]. Graph coarsening is the graph-equivalent operation of *pooling* in convolutional neural networks.

4.1 Louvain algorithms

Initially introduced in Blondel et al. [2008], Louvain algorithm is a popular clustering method for graphs based on the greedy maximization of an objective function called modularity. Its excellent speed-performance trade-off has made this algorithm the common denominator for community detection of many graph softwares such as NetworkX, Gephi, IGraph, Neo4J...

Even-though this algorithm was originally designed for hard node clustering of undirected graphs, it can naturally be extended to handle bipartite, directed or signed graphs as well as

soft clustering. This section studies some of these variations. Especially, we show in Section 4.1.2 that a generalized version of the Louvain algorithm can be used to maximize different types of modularity function that we introduce in Section 4.1.1.

4.1.1 Modularity functions

In this section, we define the different modularity functions used for graph clustering and present some of their properties. Note that we do not discuss soft-modularity functions here i.e. modularity functions that take a soft partition as input. However, we refer the interested reader to the work of [Hollocoeu et al. \[2019\]](#).

Definitions

Consider a graph $G = (V, E)$, and two probability distributions p and q over $V \times V$. The distribution q is sometimes referred to as the *null model* of the graph. The general form of the modularity function, given a resolution parameter γ is

$$\mathcal{M} = \sum_{i,j} (p_{ij} - \gamma q_{ij}) \delta_{c_i c_j}. \quad (4.1)$$

Or, aggregating the terms cluster by cluster:

$$\mathcal{M} = \sum_k (P_k - \gamma Q_k), \quad (4.2)$$

where $P_k = \sum_{(i,j) \in \mathcal{C}_k \times \mathcal{C}_k} p_{ij}$ is the probability of sampling a pair of nodes in cluster k according to distribution p . Similar definition for Q_k .

The standard choice of p and q for undirected graphs are $p_{ij} \propto A_{ij}$ and $q_{ij} \propto d_i d_j$, which lead to the modularity as defined by [Newman \[2006a\]](#):

$$\mathcal{M} = \sum_{i,j} \left(\frac{A_{ij}}{w} - \gamma \frac{d_i d_j}{w^2} \right) \delta_{c_i c_j}. \quad (4.3)$$

Another choice is $p_{ij} \propto A_{ij}$ and $q_{ij} \propto 1$, which is referred to as the constant Potts model [[Traag et al., 2011](#)]:

$$\mathcal{M} = \sum_{i,j} \left(\frac{A_{ij}}{w} - \gamma \frac{1}{n^2} \right) \delta_{c_i c_j}. \quad (4.4)$$

If the graph is directed, two different approaches have been proposed. On the one hand, [Dugué and Perez \[2015\]](#) suggests to use $p_{ij} \propto A_{ij}$ and $q_{ij} \propto d_i f_j$, which leads to:

$$\mathcal{M} = \sum_{i,j} \left(\frac{A_{ij}}{w} - \gamma \frac{d_i f_j}{w^2} \right) \delta_{c_i c_j}. \quad (4.5)$$

On the other hand, [Lambiotte et al. \[2008\]](#) and [Kim et al. \[2010\]](#) suggest to use $p_{ij} = A_{ij} \pi_j / d_j$ and $q_{ij} = \pi_i \pi_j$, where π denotes the Pagerank probability distribution without damping factor. This leads to:

$$\mathcal{M} = \sum_{i,j} \left(\frac{A_{ij} \pi_j}{d_j} - \gamma \pi_i \pi_j \right) \delta_{c_i c_j}.$$

The authors prove that this modularity is the same as equation (4.3) applied to the modified adjacency

$$\tilde{A}_{ij} = \frac{1}{2} \left(\frac{A_{ij}\pi_j}{d_j} + \frac{A_{ji}\pi_i}{d_i} \right).$$

In the case of bipartite graphs, Barber [2007] proposed to use $p_{ij} = p_{ji} \propto B_{ij}$ and $q_{ij} \propto d_i f_j$ if $(i, j) \in V_1 \times V_2$ and $q_{ij} = 0$ otherwise. Which leads to the bimodularity function:

$$\mathcal{M} = \sum_{i \in V_1, j \in V_2} \left(\frac{B_{ij}}{w} - \gamma \frac{d_i f_j}{w^2} \right) \delta_{c_i c_j}.$$

Proposition 8. *Let B be the biadjacency matrix of a bipartite graph and let*

$$A = \begin{pmatrix} 0 & B \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{(n_1+n_2) \times (n_1+n_2)}$$

be the adjacency matrix of this graph, seen as a directed one. Given a clustering c , the bimodularity defined in equation (4.1.1) coincides with the directed-modularity as defined in equation (4.5).

Proof. First, let us note that A and B have the same total weight w . Next, note that $d^A = (0, d^B)$ and $f^A = (0, f^B)$. Let $Z = (Z_1, Z_2)^\top$ be the membership matrix. Then, starting from Equation 4.5:

$$\begin{aligned} \mathcal{M} &= \sum_{i,j} \left(\frac{A_{ij}}{w} - \gamma \frac{d_i^A f_j^A}{w^2} \right) \delta_{c_i c_j}, \\ &= \text{Tr} \left(Z^\top \left(\frac{A}{w} - \frac{\gamma}{w^2} d^A (f^A)^\top \right) Z \right), \\ &= \text{Tr} \left(Z_1^\top \left(\frac{B}{w} - \frac{\gamma}{w^2} d^B (f^B)^\top \right) Z_2 \right). \end{aligned}$$

□

Properties

A key property of the modularity functions, at the core of Louvain algorithm, is following the aggregation property described in Proposition 9.

Proposition 9. *The modularity of a partition of the nodes of a graph $G = (V, E)$ is equal to the modularity of the trivial partition where each node is assigned to its own cluster for the graph $\tilde{G} = (\tilde{V}, \tilde{E})$ such that \tilde{V} and \tilde{E} are the set of nodes and edges obtained by merging all nodes in the same cluster in the original partition.*

Proof. Let $\tilde{A} = Z^\top A Z$ be the adjacency matrix of the aggregated graph. We have

$$\begin{aligned} \tilde{d} &= \tilde{A} \mathbf{1} = Z^\top A Z \mathbf{1} = Z^\top d, \\ \tilde{f} &= \tilde{A}^\top \mathbf{1} = Z^\top A^\top Z \mathbf{1} = Z^\top f, \\ \tilde{w} &= \mathbf{1}^\top \tilde{A} \mathbf{1} = \mathbf{1}^\top Z^\top A Z \mathbf{1} = \mathbf{1}^\top A \mathbf{1} = w. \end{aligned}$$

Hence, $\tilde{p}_{kk'} = (Z^\top p Z)_{kk'}$ and $\tilde{q}_{kk'} = (Z^\top q Z)_{kk'}$ hold for all the probability distributions listed in Subsection 4.1.1:

$$\begin{aligned}
\mathcal{M} &= \sum_{i,j} (p_{ij} - \gamma q_{ij}) \delta_{c_i c_j}, \\
&= \sum_{i,j} (p_{ij} - \gamma q_{ij}) Z_i^\top Z_j, \\
&= \text{Tr} \left(Z^\top (p - \gamma q) Z \right), \\
&= \text{Tr} \left(I_K \left(Z^\top p Z - \gamma Z^\top q Z \right) I_K \right), \\
&= \sum_{k,k'} (\tilde{p}_{kk'} - \gamma \tilde{q}_{kk'}) \delta_{kk'}.
\end{aligned}$$

□

Another property of the generalized modularity function defined in equation (4.1) is its connection to block models for graphs. Consider a model in which, given a clustering and $\omega_{\text{in}} > \omega_{\text{out}} \geq 0$, the weight of an edge between i and j is

$$\hat{A}_{ij} = \begin{cases} \omega_{\text{in}}, & \text{if } \delta_{c_i c_j} = 1, \\ \omega_{\text{out}}, & \text{otherwise.} \end{cases}$$

We denote by $\mu(\delta)$ the edge sampling probability for this model. In particular,

$$\log(\mu_{ij}) = \log \left(\frac{\omega_{\text{in}}}{\omega_{\text{out}}} \right) \delta_{c_i c_j} + \log(\omega_{\text{out}}), \forall (i, j),$$

up to an additive constant.

We want to find a clustering such that the resulting distribution is both close to the edge sampling distribution p and far from the null model q . To do so, we seek to minimize over δ the following loss:

$$\mathcal{L} = \text{KL}(p, \mu(\delta)) - \gamma \text{KL}(q, \mu(\delta)), \tag{4.6}$$

where KL denotes the Kullback-Leibler divergence. Theorem 5 is the main result of this chapter.

Theorem 5. *The set of optimal clusterings for \mathcal{L} is equal to the set of optimal clustering for the generalized modularity metric:*

$$\arg \min_{\delta} \mathcal{L} = \arg \max_{\delta} \mathcal{M}. \tag{4.7}$$

Proof. Starting from Equation (4.6), we derive

$$\begin{aligned}
\arg \min_{\delta} \mathcal{L} &= \arg \min_{\delta} \text{KL}(p, \mu(\delta)) - \gamma \text{KL}(q, \mu(\delta)), \\
&= \arg \min_{\delta} \left(\sum_{i,j} p_{ij} \log \left(\frac{p_{ij}}{\mu_{ij}} \right) - \gamma \sum_{i,j} q_{ij} \log \left(\frac{q_{ij}}{\mu_{ij}} \right) \right), \\
&= \arg \min_{\delta} \left(- \sum_{i,j} p_{ij} \log(\mu_{ij}) + \gamma \sum_{i,j} q_{ij} \log(\mu_{ij}) \right), \\
&= \arg \max_{\delta} \left(\sum_{i,j} (p_{ij} - \gamma q_{ij}) \log(\mu_{ij}) \right).
\end{aligned}$$

As $\log(\mu_{ij}) = \log\left(\frac{\omega_{\text{in}}}{\omega_{\text{out}}}\right) \delta_{c_i c_j} + \log(\omega_{\text{out}})$ and $\log\left(\frac{\omega_{\text{in}}}{\omega_{\text{out}}}\right) > 0$:

$$\begin{aligned}
\arg \min_{\delta} \mathcal{L} &= \arg \max_{\delta} \sum_{i,j} (p_{ij} - \gamma q_{ij}) \delta_{c_i c_j}, \\
&= \arg \max_{\delta} \mathcal{M}.
\end{aligned}$$

□

Theorem 5 is complementary to the results of Newman [2016] which connect modularity maximization and block models through statistical inference. Now, the following property is related to the so-called *resolution limit* of modularity optimization.

Proposition 10. *The optimal partition at resolution γ for a graph with L connected components with respective total weights w_1, \dots, w_L is the concatenation of optimal partitions for each connected component at resolution $\gamma_l = \frac{w_l}{w} \gamma$.*

Proof. We denote by V_1, \dots, V_L the set of nodes corresponding to the connected components.

$$\begin{aligned}
\mathcal{M} &= \sum_l \frac{w_l}{w} \sum_{i,j \in V_l} \left(\frac{A_{ij}}{w_l} - \gamma_l \frac{d_i d_j}{w_l^2} \right) \delta_{c_i c_j} - \gamma \sum_{r \neq s} \sum_{(i,j) \in V_r \times V_s} \frac{d_i d_j}{w^2} \delta_{c_i c_j}, \\
&\leq \sum_l \frac{w_l}{w} \mathcal{M}_{|V_l, \gamma_l}, \\
&\leq \sum_l \frac{w_l}{w} \mathcal{M}_{|V_l, \gamma_l}^*.
\end{aligned}$$

$\mathcal{M}_{|V_l, \gamma_l}$ denotes the modularity of the partition of V_l at resolution γ_l and $\mathcal{M}_{|V_l, \gamma_l}^*$ the optimal one. This upper bound is reached by the concatenation of optimal partitions of V_1, \dots, V_L , hence the result. □

Finally, note that finding the partition that maximizes the modularity is an NP-complete problem [Brandes et al., 2006]. Still, many algorithms offer to find local optimums. Some methods rely on spectral division [Newman, 2006b, White and Smyth, 2005], some on simulated annealing [Guimera et al., 2004, Newman and Girvan, 2004], but the most commonly used is Louvain greedy optimization which is the subject of Section 4.1.2.

4.1.2 Generalized Louvain algorithm

This section describes the Generalized Louvain algorithm and discusses some implementation issues. Even though the Louvain algorithm [Blondel et al., 2008] was introduced to maximize the modularity for undirected networks as in equation (4.3), the general idea can be applied to any type of objective function that satisfies two properties:

- Easy computation of variation in score resulting from the modification of the cluster assignment of a single node.
- Aggregation property as described in Proposition 9.

The pipeline for the Louvain algorithm consists in two nested loops which we call "Macro-Louvain" and "Micro-Louvain", respectively described in Algorithms 4.1 and 4.2.

Local updates The efficiency of the Generalized Louvain Algorithm relies on the simple closed form solution of

$$\arg \max_c \Delta \mathcal{M}(c_i \leftarrow c)$$

in Algorithm 4.2. Note that it is not necessary to test all possible values of c but only the different labels among the neighbors of i .

From equation (4.2), we have:

$$\begin{aligned} \Delta \mathcal{M}(c_i \leftarrow c_j) &= \sum_k (\Delta P_k - \gamma \Delta Q_k), \\ &= (\Delta P_{c_i} - \gamma \Delta Q_{c_i}) + (\Delta P_{c_j} - \gamma \Delta Q_{c_j}), \end{aligned}$$

as i and j are the only two modified clusters. Actually, we only need to derive the variations for \mathcal{M} corresponding to equation (4.5) to cover all cases listed in subsection 4.1.1.

For the case of equation (4.5), we have, on the one hand,

$$\begin{aligned} \Delta P_{c_i} &= \frac{1}{w} \left[2A_{ii} - \sum_{c_l=c_i} (A_{il} + A_{li}) \right], \\ \Delta P_{c_j} &= \frac{1}{w} \sum_{c_l=c_j} (A_{il} + A_{li}). \end{aligned}$$

For the null model, on the other hand, it is convenient to rewrite

$$Q_k = \frac{1}{w^2} D_k F_k,$$

where $D_k = \sum_{c_i=k} d_i$ is the total out-degree of cluster k and $F_k = \sum_{c_i=k} f_i$ its total in-degree. Then,

$$\begin{aligned} \Delta Q_{c_i} &= \frac{1}{w^2} [f_i(d_i - D_{c_i}) + d_i(f_i - F_{c_i})], \\ \Delta Q_{c_j} &= \frac{1}{w^2} [f_i D_{c_j} + d_i F_{c_j}]. \end{aligned}$$

The case of equation (4.3) is simply the particular case where $A_{ij} = A_{ji}$ and $d_i = f_i$. Besides, it is sufficient to replace d and f by 1 to recover (4.4). Note that the values D_k and F_k can easily be stored and updated during the computations.

Algorithm 4.1 Macro-Louvain

Require: Adjacency A .

- 1: Initialize membership $Z = I$.
 - 2: **repeat**
 - 3: $\hat{Z} = \text{Micro-Louvain}(A)$
 - 4: $Z \leftarrow Z\hat{Z}$.
 - 5: $A \leftarrow Z^\top AZ$.
 - 6: **until** Convergence.
 - 7: **return** $c(Z)$.
-

Algorithm 4.2 Micro-Louvain

Require: Adjacency A .

- 1: Initialize $c = [1, 2, \dots, n]$.
 - 2: **repeat**
 - 3: **for** $i = 1 \leq \dots \leq n$ **do**
 - 4: $c_i = \arg \max_{c_j} \Delta \mathcal{M}(c_i \leftarrow c_j)$.
 - 5: **end for**
 - 6: **until** Convergence.
 - 7: **return** $Z(c)$.
-

Convergence The modularity is non-decreasing under the local updates performed in Micro-Louvain and the aggregation property 9 ensures that it is not decreasing under each update performed by Macro-Louvain. As the modularity is upper-bounded, the Generalized Louvain Algorithm converges towards a local maximum of the modularity. In practice, the stopping criterion for Macro and Micro-Louvain can be a maximum number of passes, a tolerance parameter for the increase in modularity, or both.

Parallel computing for Bilouvain In the case of bimodularity optimization, the local update for a node in V_1 only depends on the cluster assignment of the nodes in V_2 and vice versa. It is thus technically possible to perform all local updates for nodes of V_1 in parallel (the same goes for V_2). However, as Louvain is a greedy method, it is highly dependent on the order in which the cluster assignments are updated. Such an arbitrary order has no guarantee to result in better partitions than a random order.

Recall that, as for k-means initialization, a common practice to maximize modularity is to run several instances of Louvain with different node orderings and select the best partition afterwards. We illustrate this in Example 4.1.1. Two different orderings of the nodes yield distinct partitions with similar modularity.

Example. 4.1.1: Node shuffling for Louvain

```
[1]: from sknetwork.clustering import Louvain, modularity
      from sknetwork.data import karate_club
      from sklearn.metrics import v_measure_score

      A = karate_club()
      louvain = Louvain(shuffle_nodes=True)
      labels1 = louvain.fit_transform(A)
      labels2 = louvain.fit_transform(A)

      round(v_measure_score(labels1, labels2), 2)

[1]: 0.81

[2]: mod1 = modularity(A, labels1)
      mod2 = modularity(A, labels2)
      print(round(mod1, 2), round(mod2, 2))

0.39 0.39
```

4.2 Experiments

This section is dedicated to clustering experiments. Section 4.2.1 illustrates the generalized Louvain algorithm on the simple case. Section 4.2.2 provides a general clustering benchmark, while Section 4.2.3 demonstrates the efficiency of Louvain algorithm on a digit images clustering task. We use datasets and metrics already introduced in Chapter 3.

4.2.1 Illustration

Figure 4.1 illustrates two partitions of the same graph connecting movies to actors. The first partition is obtained by maximization of the modularity, seeing the graph as an undirected one, while the second is obtained by maximization of the bimodularity. As we can see, these partitions do not even have the same number of clusters even though some of them look similar.

4.2.2 General benchmark

In this section, we compare the performance of several clustering algorithms on labeled datasets. For this experiment, we compute the Adjusted Rand Index with respect to the ground truth labels for the following algorithms:

Louvain algorithms: We refer to them as *Louvain* for the maximization of (4.3), *DiLouvain* for (4.5), *BiLouvain* for (4.1.1) and *Potts* for (4.4). We try different resolutions values for each one of them: $\gamma \in \{0.1, 0.2, \dots, 1.5\}$. We omit (4.1.1) as the pagerank vector without damping is usually not defined on real directed graphs.

Spectral algorithms: We experiment with the standard Laplacian Eigenmaps [Belkin and Niyogi, 2002b] and with the algorithm described in Section 3.3 which we refer to as *GSVD*.

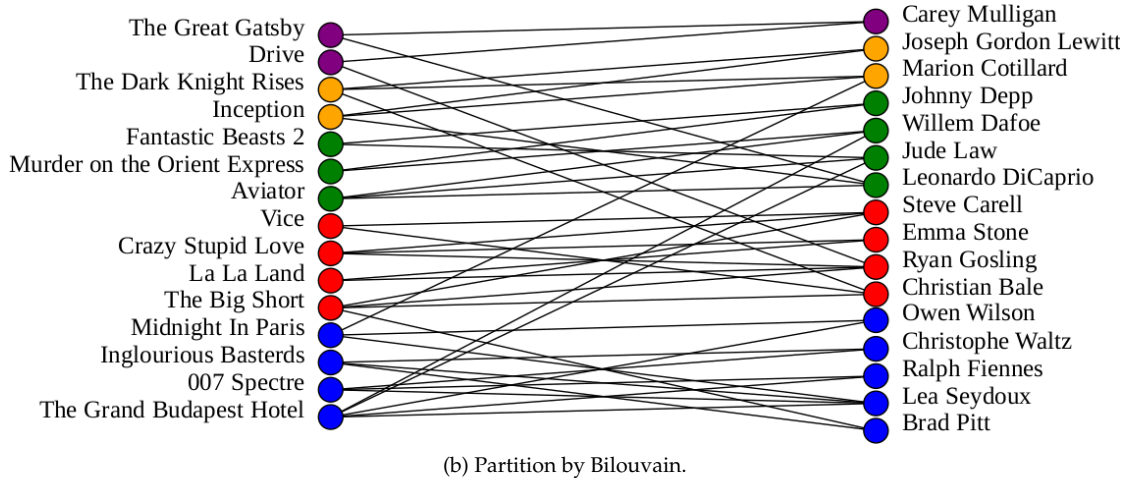
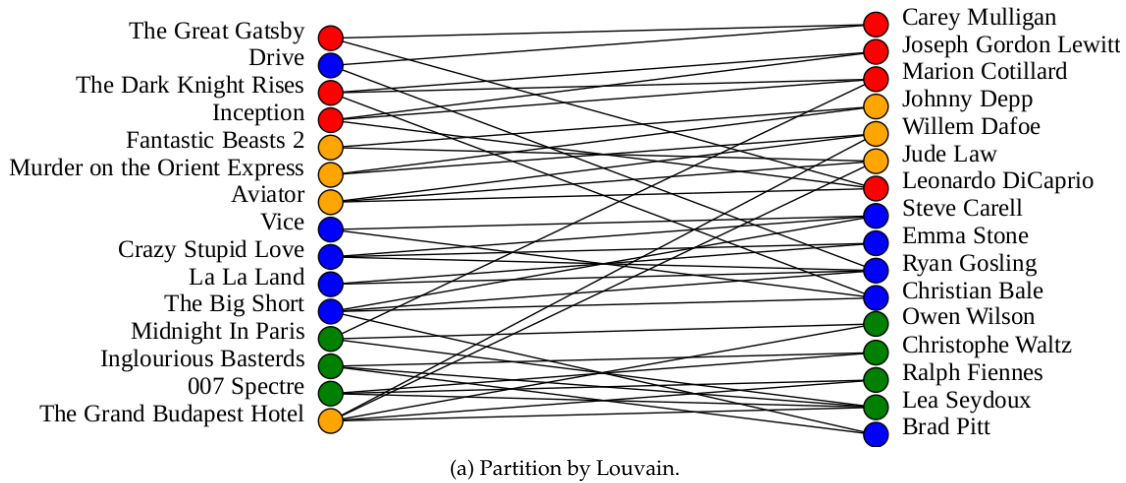


Figure 4.1: Movie-actor bipartite graph.

For these algorithms, we test different combinations of embedding dimension d and number of clusters k : $d \in \{8, 16, 32\}$ and $k \in \{0.8 \times K, K, 1.2 \times K\}$ where K is the number of clusters in the ground truth.

Propagation [Raghavan et al., 2007]. This method iterates over the nodes and assigns them the most represented label among their neighbors until convergence. The label of each node is initialized with its index.

We report the best ARI score obtained by each algorithm in Table 4.1 and the corresponding runtimes in Table 4.2. All computations are performed in the laptop setup. We use our Scikit-network implementations for all algorithms.

Our first observation is the generally good performance of all Louvain methods, except for *Potts*, combined with some of the shortest running times. Note that, among these methods, DiLouvain performs best on the directed graph WV and Bilouvain performs best on the bipartite graph NG, even though it is outperformed by GSVD. On the other hand, the standard Laplacian Eigenmaps

algorithm	CO	CS	NG	WS	WV
Louvain	0.39	0.22	0.31	0.23	0.31
DiLouvain	-	-	-	0.23	0.37
BiLouvain	0.31	0.20	0.39	0.22	0.35
Potts	0.28	0.23	0.00	0.16	0.19
Lap. Eigen.	0.09	0.03	0.20	0.09	0.16
GSVD	0.22	0.09	0.48	0.17	0.31
Propagation	0.16	0.14	0.00	0.00	0.00

Table 4.1: Best ARI for each algorithm.

algorithm	CO	CS	NG	WS	WV
Louvain	0.02	0.01	2.76	0.05	0.20
DiLouvain	-	-	-	0.05	0.25
BiLouvain	0.02	0.02	2.53	0.12	0.63
Potts	0.02	0.01	0.41	0.04	0.30
Lap. Eigen.	1.37	1.78	14.0	1.29	0.96
GSVD	0.32	0.15	6.67	0.67	1.49
Propagation	0.01	0.00	0.51	0.01	0.05

Table 4.2: Running time for best ARI (seconds).

has poor ARI on all graphs with the longest running times. Finally, the Propagation algorithm is very fast but yields very poor results.

In Chapter 5, we present a similar benchmark in a semi-supervised setting.

4.2.3 Digits classification

In order to further illustrate the interest of Louvain clustering, we perform an experiment on the MNIST dataset subset of Scikit-learn. The data is composed of 1 797 images in 8×8 gray scale pixels format. Each image represents a digit between 0 and 9. Some samples are displayed in Figure 4.2.

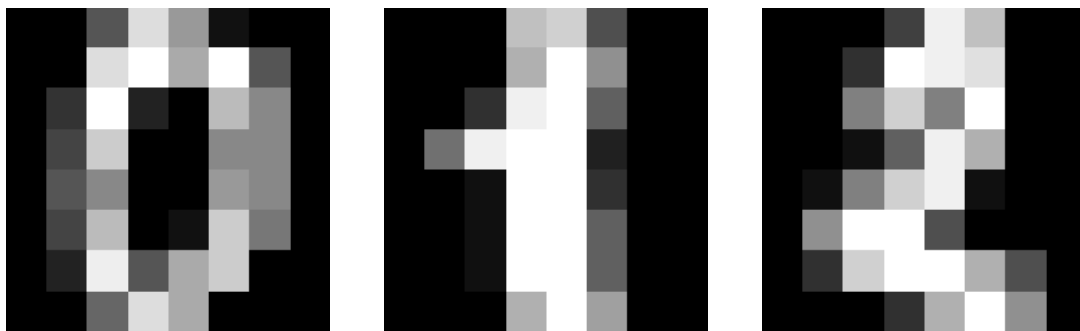


Figure 4.2: Some images from the dataset.

The goal is to recover these labels without supervision. We use the V-measure score to compare two different pipelines.

K-means The data is centered to have zero mean, then we apply a principal component analysis with whitening. In the *L2* version, we apply k-means directly on the output of the PCA. In the *cosine* version, we first normalize each row so that it has an l_2 norm equal to 1.

Louvain We apply a principal component analysis without whitening and construct a nearest-neighbor graph on which we apply DiLouvain, as the resulting graph is directed by construction.

We perform a grid-search optimization for the following hyper-parameters: dimension of the PCA $n_{\text{dim}} \in \{8, 16, 32\}$, number of clusters for k-means $k \in \{8, 9, 10, 11, 12\}$, number of nearest neighbors to construct the graph $n_{\text{neigh}} \in \{1, 3, 5\}$ and resolution for DiLouvain $\gamma \in \{0.1, 0.5, 1.\}$. We report the best score for each pipeline in Table 4.3. See how Louvain outperforms k-means by a significant margin.

pipeline	k-means (L2)	k-means (cosine)	Louvain
V	0.69	0.74	0.93

Table 4.3: V-measure scores.

Chapter 5

Semi-supervised node classification

This chapter is dedicated to the application of heat diffusion to node classification. Heat diffusion is a very intuitive and interpretable physical process which has inspired the heat kernel, a standard tool in graph mining with many applications, including semi-supervised classification of nodes. The idea is to compute the system state at some finite time t after free diffusion from a set of hot source nodes. In this work, we use a different diffusion based on the solution to a Dirichlet problem, with both hot and cold sources. These new boundary conditions lead to meaningful steady states, which can be exploited for node classification. This chapter covers the work presented in [De Lara and Bonald \[2020\]](#).

References

- De Lara, N., & Bonald, T. (2020). *A Consistent Diffusion-Based Algorithm for Semi-Supervised Classification on Graphs*. preprint.

5.1 Introduction

Heat diffusion, describing by the evolution of temperature T in an isotropic material, is governed by the following heat equation, initially developed by Joseph Fourier:

$$\frac{\partial T}{\partial t} = \alpha \Delta T, \quad (5.1)$$

where α is the thermal conductivity of the material and Δ is the Laplace operator. In steady state, this equation simplifies to $\Delta T = 0$ and the function T is said to be *harmonic*. The Dirichlet problem consists in finding the equilibrium in the presence of boundary conditions, that is when the temperature T is fixed on the boundary of the region of interest. [Figure 5.1](#) is a small tribute to these scientists.



Figure 5.1: Left to right: Joseph Fourier, Pierre-Simon de Laplace and Johann P.G.L. Dirichlet.

In this chapter, we show how to apply a discrete version of the Dirichlet problem to the semi-supervised classification of the nodes of a graph: given labels known for some nodes of the graph, referred to as the *seeds*, how to infer the labels of the other nodes? The number of seeds is typically small compared to the total number of nodes (e.g., 1%), hence the name of semi-supervised classification.

We propose to solve one Dirichlet problem per label by setting the temperatures of the seeds accordingly, in a one-against-all strategy, and to classify the nodes with respect to the deviation of temperature to the mean. We prove the consistency of our algorithm on a simple block model and its efficiency through experiments on real graphs.

The application of the heat equation to graph mining is well known in the literature. It is usually associated with kernel learning [Kondor and Lafferty, 2002] and referred to as heat kernel. It has been used to many different tasks like pattern matching [Thanou et al., 2017], node ranking [Ma et al., 2008, 2011], graph embedding [Donnat et al., 2018], graph clustering [Tremblay and Borgnat, 2014], and semi-supervised classification [Zhu, 2005, Merkurjev et al., 2016, Berberidis et al., 2018, Li et al., 2019]. In most cases, the learning task relies on the transient state of the diffusion process, using hot sources only. Our approach is different in that we solve one Dirichlet problem per label, using a one-against-all strategy, with both hot sources (the seeds of the considered label) and cold sources (the seeds of the other labels). The algorithm is parameter-free, unlike existing techniques based on the heat kernel, whose performance critically depends on some time parameter used to stop the diffusion. Our theoretical analysis also shows that temperature centering is critical, i.e., classification must rely on temperature deviations to the mean for each Dirichlet problem.

The rest of this chapter is organized as follows. In section 5.2, we introduce the Dirichlet problem on graphs. Section 5.3 describes our algorithm for node classification. The analysis showing the consistency of our algorithm on a simple block model is presented in section 5.4. Finally, Section 5.5 presents the experiments.

5.2 Dirichlet problem on graphs

In this section, we introduce the Dirichlet problem on graphs, show its interpretation in terms of random walk in the graph and characterize the solution. The difference with the heat kernel is also highlighted.

5.2.1 Heat equation

Consider a graph G with n nodes. We first assume that the graph is undirected and unweighted. We assume that nodes are indexed from 1 to n and denote by A the corresponding adjacency matrix. This is a symmetric, binary matrix. Let $d = A\mathbf{1}$ be the degree vector, which is assumed positive, and $D = \text{diag}(d)$. The Laplacian matrix is defined by

$$L = D - A.$$

Now let S be some strict subset of $\{1, \dots, n\}$ and assume that the temperature of each node $i \in S$ is set at some fixed value T_i . We are interested in the evolution of the temperatures of the other nodes. Heat exchanges occur through each edge of the graph proportionally to the temperature difference between the corresponding nodes. Then,

$$\forall i \notin S, \quad \frac{dT_i}{dt} = \sum_{j=1}^n A_{ij}(T_j - T_i),$$

that is

$$\forall i \notin S, \quad \frac{dT_i}{dt} = -(LT)_i,$$

where T is the vector of temperatures. This is the heat equation in discrete space, where $-L$ plays the role of the Laplace operator in (5.1). At equilibrium, T satisfies Laplace's equation:

$$\forall i \notin S, \quad (LT)_i = 0. \quad (5.2)$$

We say that the vector T is *harmonic*. With the boundary conditions T_i for all $i \in S$, this defines a Dirichlet problem in discrete space.

5.2.2 Random walk

Consider a random walk in the graph G with a probability of moving from node i to node j equal to A_{ij}/d_i . Let X_0, X_1, X_2, \dots be the sequence of nodes visited by the random walk. This defines a Markov chain on $\{1, \dots, n\}$ with transition matrix $P = D^{-1}A$.

Observing that $L = D(I - P)$, Laplace's equation can be written equivalently

$$\forall i \notin S, \quad T_i = (PT)_i. \quad (5.3)$$

In other words, the temperature of each node $i \notin S$ at equilibrium is the average of the temperature of its neighbors. This implies in particular that the solution to the Dirichlet problem is unique, provided that the graph is connected. For the sake of completeness, we provide a proof of this result in the considered discrete case:

Proposition 11. *If the graph is connected, there is at most one solution to the Dirichlet problem.*

Proof. We first prove that the maximum and the minimum of the vector T are achieved on the boundary S . Let i be any node such that T_i is maximum. If $i \notin S$, it follows from (5.3) that T_j is maximum for all neighbors j of i . If no such node belongs to S , we apply again this argument until we reach a node in S . Such a node exists because the graph is connected. It achieves the maximum of the vector T . The proof is similar for the minimum.

Now consider two solutions T, T' to Laplace's equation. Then $\Delta = T' - T$ is a solution of Laplace's equation with the boundary condition $\Delta_i = 0$ for all $i \in S$. We deduce that $\Delta_i = 0$ for all i (because both the maximum and the minimum are equal to 0), that is $T' = T$. \square

Now let P_{ij}^S be the probability that the random walk first hits the set S in node j when starting from node i . Observe that P^S is a stochastic matrix, with $P_{ij}^S = \delta_{ij}$ (Kronecker delta) for all $i \in S$. By first-step analysis, we have:

$$\forall i \notin S, \quad P_{ij}^S = \sum_{k=1}^n P_{ik} P_{kj}^S. \quad (5.4)$$

The following result provides a simple interpretation of the solution to the Dirichlet problem in terms of random walk in the graph: the temperature of any node is the average of the temperatures of the nodes at the boundary, weighted by the probabilities of reaching each of these nodes first:

Proposition 12. *The solution to the Dirichlet problem is*

$$\forall i \notin S, \quad T_i = \sum_{j \in S} P_{ij}^S T_j. \quad (5.5)$$

Proof. The vector T defined by (5.5) satisfies for all $i \notin S$:

$$\sum_{j=1}^n P_{ij} T_j = \sum_{j=1}^n P_{ij} \sum_{k \in S} P_{jk}^S T_k = \sum_{k \in S} P_{ik}^S T_k = T_i,$$

where we have used (5.4). Thus, T satisfies (5.3). The proof then follows from Proposition 11. \square

5.2.3 Solution to the Dirichlet problem

We now characterize the solution to the Dirichlet problem in discrete space. Without any loss of generality, we assume that nodes with unknown temperatures (i.e., not in S) are indexed from 1 to $n - s$ so that the vector of temperatures can be written

$$T = \begin{bmatrix} X \\ Y \end{bmatrix},$$

where X is the unknown vector of temperatures at equilibrium, of dimension $n - s$. Writing the transition matrix in block form as

$$P = \begin{bmatrix} Q & R \\ \cdot & \cdot \end{bmatrix},$$

it follows from (5.3) that:

$$X = QX + RY, \quad (5.6)$$

so that:

$$X = (I - Q)^{-1}RY. \quad (5.7)$$

Note that the inverse of the matrix $I - Q$ exists whenever the graph is connected, which implies that the matrix Q is sub-stochastic with spectral radius strictly less than 1 [Chung, 1997].

The exact solution of (5.7) requires to solve a (potentially large) linear system. In practice, a very good approximation is provided by a few iterations of (5.6), the rate of convergence depending on the spectral radius of the matrix Q . The small-world property of real graphs suggests that a few iterations are enough for real graphs [Watts and Strogatz, 1998]. This will be confirmed by the experiments.

5.2.4 Heat kernel

Now consider the heat diffusion in the absence of boundary conditions:

$$\frac{dT}{dt} = -LT. \quad (5.8)$$

First, note that, in this case, the average temperature of the nodes \bar{T} is preserved over time:

$$\frac{d\bar{T}}{dt} = \frac{1^\top}{n} \cdot \frac{dT}{dt} = - \left(\frac{1^\top}{n} L \right) T = 0.$$

The solution of this differential equation is given by:

$$T(t) = e^{-Lt}T(0).$$

The matrix $H(t) = e^{-Lt}$ is referred to as the *heat kernel*. It can be expressed through the spectral decomposition of L , $L = U\Lambda U^\top$ with $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ the diagonal matrix of eigenvalues $\lambda_1 = 0 \leq \lambda_2 \leq \dots \leq \lambda_n$ and $U = (u_1, \dots, u_n)$ an orthogonal matrix of eigenvectors:

$$H(t) = \sum_{k=1}^n e^{-\lambda_k t} u_k u_k^\top.$$

If the graph is connected, then $\lambda_2 > 0$ and

$$H(t) \rightarrow u_1 u_1^\top \quad \text{when } t \rightarrow +\infty.$$

Since $u_1 \propto 1$, the temperature becomes uniform at equilibrium. The interest of the diffusion lies in the transient states, for some finite time t .

Note that the exact computation of the heat kernel requires the spectral decomposition of the Laplacian matrix, which is not feasible for large graphs. In practice, an approximation of the system state at some finite time t can be derived directly from the heat diffusion (5.8). For some sufficiently small time step $\delta > 0$, the state at time t follows from t/δ iterations of the following updates:

$$T \leftarrow T - \delta LT. \quad (5.9)$$

Another approach consists in applying the diffusion in discrete time:

$$T \leftarrow PT, \quad (5.10)$$

which is similar to iteration (5.3) but in the absence of boundary. In both (5.9) and (5.10), the number of iterations, say N , plays a key role, as the state T converges to the equilibrium with uniform temperatures when $N \rightarrow +\infty$.

Note that, in the case of the diffusion in discrete time, it is the weighted average temperature $\hat{T} = \frac{1}{w} \sum_i d_i T_i$ that is preserved over time:

$$\hat{T}(t+1) = \frac{d^\top}{w} T(t+1) = \left(\frac{d^\top}{w} P \right) T(t) = \frac{d^\top}{w} T(t) = \hat{T}(t).$$

5.2.5 Extensions

All results extend to weighted graphs, with a positive weight assigned to each edge; this weight can then be interpreted as the *thermal conductivity* of the edge in the diffusion process. The results also apply to directed graphs. Indeed, a directed graph G of n nodes, with adjacency matrix A , can be considered as a bipartite graph of $2n$ nodes, with adjacency matrix:

$$\begin{bmatrix} 0 & A \\ A^\top & 0 \end{bmatrix}$$

The diffusion can be applied to this bipartite graph, which is undirected. Observe that each node of the directed graph G is duplicated in the bipartite graph and is thus characterized by 2 temperatures, one as heat source (outgoing edges) and one as heat destination (incoming edges). It is not necessary for the directed graph to be strongly connected; only the associate bipartite graph needs to be connected.

5.3 Node classification algorithm

In this section, we introduce a node classification algorithm based on the Dirichlet problem. The objective is to infer the labels of all nodes given the labels of a few nodes called the *seeds*. Our algorithm is a simple modification of the popular method proposed by [Zhu et al. \[2003\]](#). Specifically, we propose to center temperatures before classification.

5.3.1 Binary classification

When there are only two different labels, the classification can be done by solving one Dirichlet problem. The idea is to use the seeds with the first label as hot sources, setting their temperature at 1, and the seeds with the second label as cold sources, setting their temperature at 0. The solution to this Dirichlet problem gives temperatures between 0 and 1.

A natural approach, proposed by [Zhu et al. \[2003\]](#), consists in assigning label 1 to all nodes with temperature above 0.5 and label 2 to other nodes. The analysis of section 5.4 suggests that it is preferable to set the threshold to the mean temperature,

$$\bar{T} = \frac{1}{n} \sum_{i=1}^n T_i.$$

Specifically, all nodes with temperature above \bar{T} are assigned label 1, the other are assigned label 2. Equivalently, temperatures are centered before classification: after centering, nodes with positive temperature are assigned label 1, the others are assigned label 2.

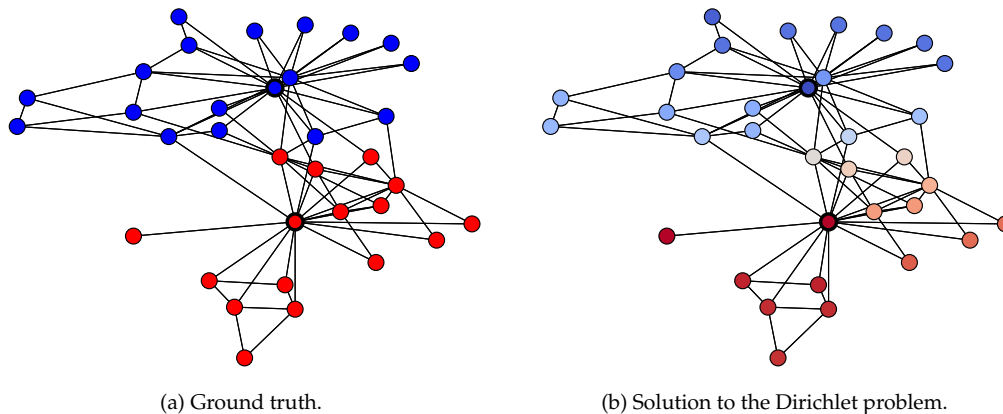


Figure 5.2: Binary classification of the Karate Club graph [Zachary, 1977] with 2 seeds (indicated with a black circle). Red nodes have label 1, blue nodes have label 2.

Note that the temperature of each node can be used to assess the confidence in the classification: the closer the temperature to the mean, the lower the confidence. This is illustrated by Figure 5.2 (the lighter the color, the lower the confidence). In this case, only one node is misclassified and has indeed a temperature close to the mean, as illustrated in Example 5.3.1.

```

Example. 5.3.1: Binary classification

[1]: from sknetwork.data import karate_club
      from sknetwork.ranking import Dirichlet

      karate = karate_club(metadata=True)
      A = karate.adjacency
      labels_true = karate.labels

      dirichlet = Dirichlet()
      T = dirichlet.fit_transform(A, {0: 1, 33: 0})
      labels_pred = (T > T.mean()).astype(int)

      (labels_pred != labels_true).sum()

[1]: 1

[2]: T[(labels_pred != labels_true)][0], T.mean()

[2]: (0.41024768968535086, 0.4913702992504278)

```

5.3.2 Multi-class classification

In the presence of more than 2 labels, we use a *one-against-all* strategy: the seeds of each label alternately serve as hot sources (temperature 1) while all the other seeds serve as cold sources

(temperature 0). After centering the temperatures (so that the mean temperature of each diffusion is equal to 0), each node is assigned the label that maximizes its temperature. This algorithm, we refer to as Dirichlet classifier, is parameter-free.

Algorithm 5.1 Dirichlet classifier

Require: Seed set S and associated labels $y \in \{1, \dots, K\}$.

```

1: for  $k$  in  $\{1, \dots, K\}$  do
2:    $T^S = 0$ .
3:   for  $i \in S$  do
4:     if  $y_i = k$  then
5:        $T_i^S = 1$ .
6:     end if
7:   end for
8:    $T^{(k)} \leftarrow \text{Dirichlet}(S, T^S)$ .
9:    $\Delta^{(k)} \leftarrow T^{(k)} - \text{mean}(T^{(k)})$ 
10: end for
11: for  $i \notin S$  do
12:    $x_i = \arg \max_{k=1, \dots, K} (\Delta_i^{(k)})$ 
13: end for
14: return  $x$ , labels of nodes outside  $S$ 

```

The key difference with the *vanilla* method lies in temperature centering (line 9 of the algorithm). Another variant proposed by [Zhu et al. \[2003\]](#) consists in rescaling the temperature vector by the weight of the considered label in the seeds (see equation (9) in their paper).

5.3.3 Time complexity

The time complexity depends on the algorithm used to solve the Dirichlet problem. We here focus on the approximate solution by successive iterations of (5.6). Let m be the number of nonzero entries in A . Using the Compressed Sparse Row format for the adjacency matrix, each matrix-vector product has a complexity of $O(m)$. The complexity of Algorithm 5.1 is then $O(NKm)$, where N is the number of iterations. Note that the K Dirichlet problems are independent and can thus be computed in parallel.

5.4 Analysis

In this section, we prove the consistency of Algorithm 5.1 on a simple block model. In particular, we highlight the importance of temperature centering in the analysis.

5.4.1 Block model

Consider a graph of n nodes consisting of K blocks of respective sizes n_1, \dots, n_K , forming a partition of the set of nodes. There are s_1, \dots, s_K seeds in these blocks, which are respectively assigned labels $1, \dots, K$. Intra-block edges have weight p and inter-block edges have weight q . We expect the algorithm to assign label k to all nodes of block k whenever $p > q$, for all $k = 1, \dots, K$.

5.4.2 Dirichlet problem

Consider the Dirichlet problem when the temperature of the s_1 seeds of block 1 is set to 1 and the temperature of the other seeds is set to 0. We have an explicit solution to this Dirichlet problem.

Lemma 7. *Let T_k be the temperature of non-seed nodes of block k at equilibrium. We have:*

$$\begin{aligned}(s_1(p-q) + nq)T_1 &= s_1(p-q) + n\bar{T}q, \\ (s_k(p-q) + nq)T_k &= n\bar{T}q \quad k = 2, \dots, K,\end{aligned}$$

where \bar{T} is the average temperature, given by:

$$\bar{T} = \left(\frac{s_1 n_1(p-q) + nq}{n s_1(p-q) + nq} \right) / \left(1 - \sum_{k=1}^K \frac{(n_k - s_k)q}{s_k(p-q) + nq} \right).$$

Proof. In view of (5.2), we have:

$$\begin{aligned}(n_1(p-q) + nq)T_1 &= s_1p + (n_1 - s_1)pT_1 + \sum_{j \neq 1} (n_j - s_j)qT_j, \\ (n_k(p-q) + nq)T_k &= s_1q + (n_k - s_k)pT_k + \sum_{j \neq k} (n_j - s_j)qT_j, \quad k = 2, \dots, K.\end{aligned}$$

We deduce:

$$\begin{aligned}(s_1(p-q) + nq)T_1 &= s_1p + Vq, \\ (s_k(p-q) + nq)T_k &= s_1q + Vq \quad k = 2, \dots, K.\end{aligned}$$

with

$$V = \sum_{j=1}^K (n_j - s_j)T_j.$$

The proof then follows from the fact that

$$n\bar{T} = s_1 + \sum_{j=1}^K (n_j - s_j)T_j = s_1 + V.$$

□

5.4.3 Classification

We now state the main result of the chapter: the Dirichlet classifier is a consistent algorithm for the block model, in the sense that all nodes are correctly classified whenever $p > q$.

Theorem 6. *If $p > q$, then $x_i = k$ for all non-seed nodes i of each block k , for any parameters n_1, \dots, n_K (block sizes) and s_1, \dots, s_K (numbers of seeds).*

Proof. Let $\delta_k^{(1)} = T_k - \bar{T}$ be the deviation of temperature of non-seed nodes of block k for the Dirichlet problem associated with label 1. In view of Lemma 7, we have:

$$\begin{aligned}(s_1(p-q) + nq)\delta_1^{(1)} &= s_1(p-q)(1 - \bar{T}), \\ (s_k(p-q) + nq)\delta_k^{(1)} &= -s_k(p-q)\bar{T} \quad k = 2, \dots, K,\end{aligned}$$

For $p > q$, using the fact that $\bar{T} \in (0, 1)$, we get $\delta_1^{(1)} > 0$ and $\delta_k^{(1)} < 0$ for all $k = 2, \dots, K$. By symmetry, for each label $l = 1, \dots, K$, $\delta_l^{(l)} > 0$ and $\delta_k^{(l)} < 0$ for all $k \neq l$. We deduce that for each block k , $x_i = \arg \max_l \delta_k^{(l)} = k$ for all non-seed nodes i of block k . \square

Observe that the temperature centering is critical for consistency. In the absence of centering, non-seed nodes of block 1 are correctly classified if and only if their temperature is the highest in the Dirichlet problem associated with label 1. In view of Lemma 7, this means that for all $k = 2, \dots, K$,

$$s_1 q \frac{n_1(p-q) + nq}{s_1(p-q) + nq} + s_1(p-q) \left(1 - \sum_{j=1}^K \frac{(n_j - s_j)q}{s_j(p-q) + nq} \right) > s_k q \frac{n_k(p-q) + nq}{s_k(p-q) + nq}.$$

This condition might be violated even if $p > q$, depending on the parameters n_1, \dots, n_K and s_1, \dots, s_K . In the practically interesting case where $s_1 \ll n_1, \dots, s_K \ll n_K$ for instance (low fractions of seeds), the condition requires:

$$s_1(n_1(p-q) + nq) > s_k(n_k(p-q) + nq).$$

For blocks of same size, this means that only blocks with the largest number of seeds are correctly classified. The classifier is biased towards labels with a large number of seeds. This sensitivity of the *vanilla* algorithm to the label distribution of seeds will be confirmed in the experiments on real graphs.

5.5 Experiments

In this section, we show the impact of temperature centering on the quality of classification using both synthetic and real data. First, in Sections 5.5.1 and 5.5.2, we only focus on 3 algorithms: the vanilla algorithm (without temperature centering), the weighted version proposed by [Zhu et al., 2003] (also without temperature centering) and our algorithm (with temperature centering). Then, in Section 5.5.3, we provide a general benchmark of classification methods to assess the performance of our algorithm.

5.5.1 Synthetic data

We first use the stochastic block model (SBM) [Airoldi et al., 2008] to generate graphs with an underlying structure in clusters. This is the stochastic version of the block model used in the analysis. There are K blocks of respective sizes n_1, \dots, n_K . Nodes of the same block are connected with probability p while nodes in different blocks are connected probability q . We denote by s_k the number of seeds in block k and by s the total number of seeds.

We first compare the performance of the algorithms on a binary classification task ($K = 2$) for a graph of $n = 10\,000$ nodes with $p = 10^{-3}$ and $q = 10^{-4}$, in two different settings:

- **Seed asymmetry:** Both blocks have the same size $n_1 = n_2 = 5000$ but different numbers of seeds, with $s_1/s_2 \in \{1, 2, \dots, 10\}$ and $s_2 = 250$ (5% of nodes in block 2).
- **Block size asymmetry:** The blocks have different sizes with ratio $n_1/n_2 \in \{1, 2, \dots, 10\}$ and seeds in proportion to these sizes, with a total of $s = 1\,000$ seeds (10% of nodes).

For each configuration, the experiment is repeated 10 times. Randomness comes both from the generation of the graph and from the selection of the seeds. We report the F1-scores in Figure 5.3 (average \pm standard deviation). Observe that the variability of the results is very low due to the relatively large size of the graph. As expected, the centered version is much more robust to both types of asymmetry. Besides, in case of asymmetry in the seeds, the weighted version of the algorithm tends to amplify the bias and leads to lower scores than the vanilla version.

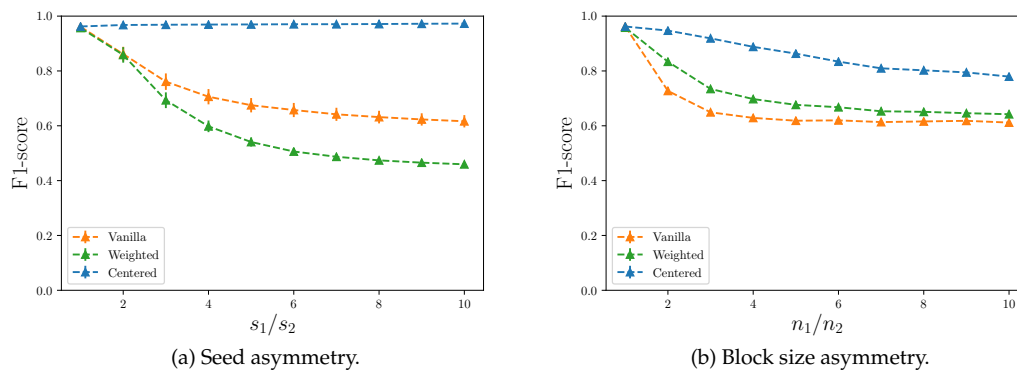


Figure 5.3: Binary classification performance on the SBM.

We show in Figure 5.4 the same type of results for $K = 10$ blocks and $p = 5 \cdot 10^{-2}$. For the block size asymmetry, the size of blocks 2, \dots , 10 is set to 1 000.

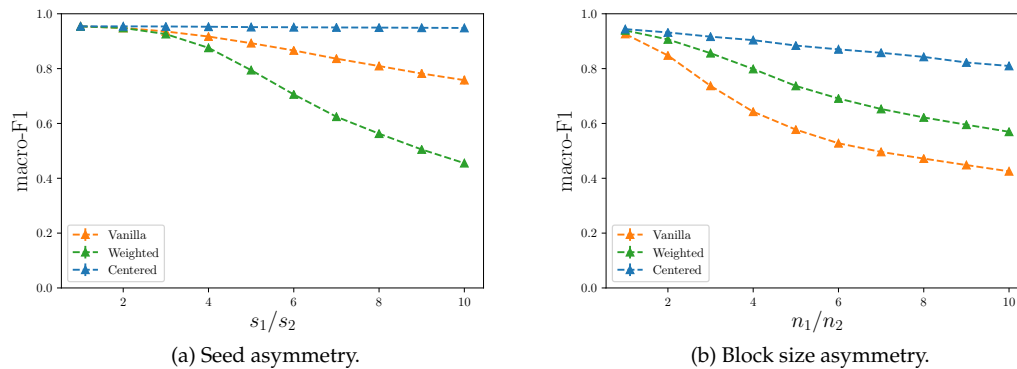


Figure 5.4: Multi-label classification performance on the SBM.

5.5.2 Real data

We use datasets from the NetSet¹ and SNAP² collections (see Table 5.1). Some of them have already been introduced in Chapter 3.

¹<https://netset.telecom-paris.fr/>

²<https://snap.stanford.edu/data/>

dataset	n	m	# classes	(%) labeled
CO	2 708	10 556	7	100
CS	3 264	9 072	6	100
WS	4 403	112 834	16	100
WV	10 012	792 091	11	100
WL	3 210 346	67 196 296	10	100
DBLP	317 080	2 099 732	5000	29
Amazon	334 863	1 851 744	5000	5

Table 5.1: Datasets.

These datasets can be categorized into 3 groups:

- **Citations networks:** Cora (CO) and CiteSeer (CS) are citation networks between scientific publications. These are standard datasets for node classification [Fey et al., 2018, Huang et al., 2018, Wijesinghe and Wang, 2019].
- **Wikipedia graphs:** Wikipedia for schools (WS) [Haruechaiyasak and Damrongrat, 2008], Wikipedia vitals (WV) and Wikilinks (WL) are graphs of hyperlinks between different selections of Wikipedia pages. In WS and WV, pages are labeled by category (People, History, Geography...). For WL, pages are labeled through clusters of words used in these articles. As these graphs are directed, we use the extension of the algorithm described in §5.2.5, with nodes considered as heat sources.
- **Social networks:** DBLP and Amazon are social networks with partial ground-truth communities [Leskovec and Krevl, 2014]. As nodes are partially labeled and some nodes have several labels, the results for these datasets are presented separately, with specific experiments based on binary classification.

For the citation networks and the Wikipedia graphs, we compare the classification performance of the algorithms in terms of macro-F1 score and two seeding policies:

- **Uniform sampling**, where seeds are sampled uniformly at random.
- **Degree sampling**, where seeds are sampled in proportion to their degrees.

In both cases, the seeds represent 1% of the total number of nodes in the graph. The process is repeated 10 times for each configuration. We do not display the results for the weighted version of the algorithm as they are very close to those obtained with the vanilla algorithm.

We report the results in Tables 5.2 and 5.3 for uniform sampling and degree sampling, respectively. We see that centered version outperforms the vanilla one by a significant margin.

algorithm	CO	CS	WS	WV	WL
Vanilla	0.19 ± 0.12	0.17 ± 0.04	0.04 ± 0.02	0.09 ± 0.04	0.19 ± 0.01
Centered	0.42 ± 0.18	0.36 ± 0.04	0.16 ± 0.11	0.55 ± 0.03	0.51 ± 0.01

Table 5.2: Macro-F1 scores (mean \pm standard deviation) with uniform seed sampling.

For the social networks, we perform independent binary classifications for each of the 3 dominant labels and average the scores. As these datasets have only a few labeled nodes, we consider

algorithm	CO	CS	WS	WV	WL
Vanilla	0.30 ± 0.08	0.16 ± 0.07	0.02 ± 0.01	0.10 ± 0.04	0.34 ± 0.00
Centered	0.51 ± 0.08	0.26 ± 0.13	0.06 ± 0.04	0.48 ± 0.02	0.45 ± 0.00

Table 5.3: Macro-F1 scores (mean \pm standard deviation) with degree seed sampling.

the most favorable scenario where seeds are sampled in proportion to the labels. Seeds still represent 1% of the nodes. The results are shown in Table 5.4.

algorithm	DBLP	Amazon
Vanilla	0.04 ± 0.00	0.05 ± 0.01
Centered	0.19 ± 0.01	0.18 ± 0.02

Table 5.4: Macro-F1 scores (mean \pm standard deviation) with balanced seed sampling.

Finally, we assess the classification performance of the algorithms in the case of seed asymmetry. Specifically, we first sample $s = 1\%$ of the nodes uniformly at random and progressively increase the number of seeds for the dominant class of each dataset, say label 1.

The process is repeated 10 times for each configuration. Figure 5.5 shows the macro-F1 scores. We see that the performance of the centered algorithm remains steady in the presence of seed asymmetry.

5.5.3 General benchmark

We evaluate our Dirichlet algorithm against the following ones from the literature based on the macro F1-score. For each dataset, we generate 10 different seed sets by selecting 1% of the nodes uniformly at random. Each algorithm is tested using all the seed sets and the results are averaged.

We add some datasets from Chapter 4 and experiment with the following algorithms:

GSVD + KNN This two step method first performs the spectral embedding of the nodes of the graph [von Luxburg, 2007, Belkin and Niyogi, 2002c], then applies a k-nearest neighbors classification [Omohundro, 1989, Bentley, 1975] in the embedding space, using the labels of the seeds. Here, we use the GSVD embedding projected onto the unit-sphere as described in Chapter 3. We test this algorithm with a dimension of the embedding space $n_{\text{dim}} \in \{8, 16, 32\}$ and the number of nearest neighbors $k \in \{1, 3, 5\}$.

Propagation [Raghavan et al., 2007]. This method iterates over the nodes and assigns them the most represented label among their neighbors until convergence. It has the benefit of being hyper-parameter free and does not require the *one-versus-all* strategy in the case of a multi-class problem.

Pagerank [Lin and Cohen, 2010]. This method is similar to the Heat Kernel except that the diffusion is replaced by a Personalized PageRank [Page et al., 1999b], with a restart distribution taken uniform over the hot sources. The pagerank is approximated using an iteration similar to (5.3) for a number of iterations $N \in \{5, 15, 25\}$ and a damping factor $\alpha \in \{0.75, 0.85, 0.95\}$.

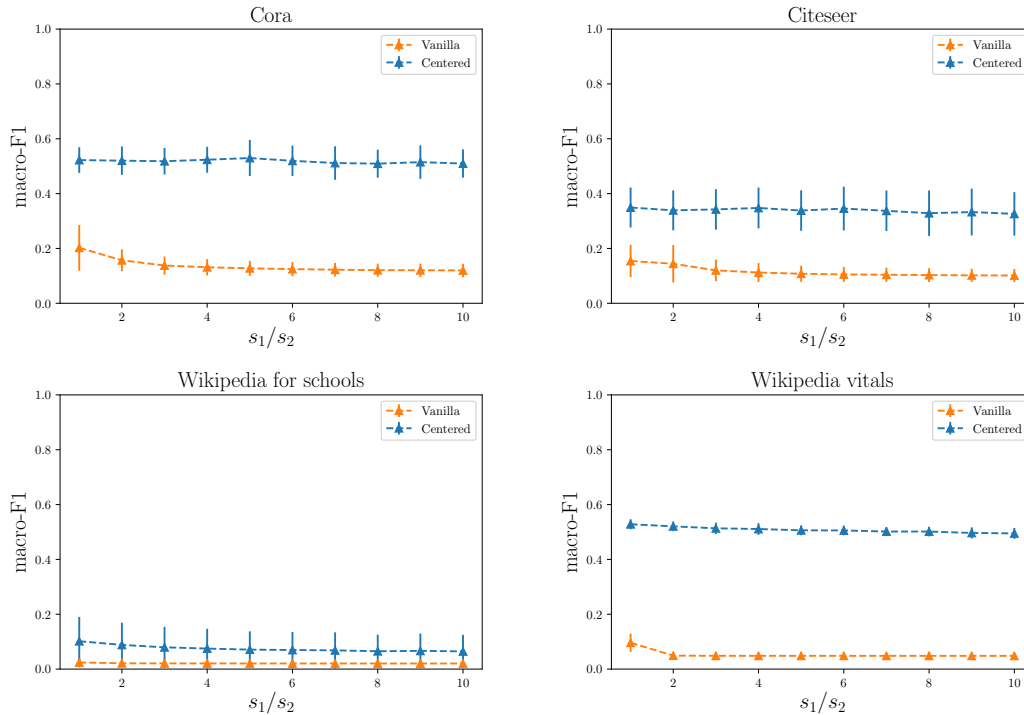


Figure 5.5: Impact of the fraction of seeds for label 1 on macro-F1 score (mean \pm standard deviation).

Graph Convolutional Network (GCN) [Kipf and Welling, 2016]. These neural networks perform nonlinear message passing between the nodes of the graph in order to minimize both a classification loss and an embedding loss. Here, we use the original network design with one hidden layer for which we test dimensions $h \in \{16, 32, 64\}$. The network is implemented with the DGL Python library [Wang et al., 2019] and trained for 500 epochs using Adam stochastic optimization [Kingma and Ba, 2014]. In the absence of exogenous node features, we use a one-hot encoding of node degrees as features, as suggested by Xu et al. [2018], combined with binning in dimension $n_{\text{feat}} \in \{16, 32, 64\}$.

For Dirichlet and the standard Heat Kernel based on the free diffusion, we test $N \in \{5, 15, 25\}$.

We report the best scores obtained by each algorithm in Table 5.5. We also report the corresponding median runtimes, computed in the server setup, in Table 5.6. Computations longer than 6 hours trigger a Time Out.

As we can see, Dirichlet and Pagerank perform very well on most graphs, however, recall that Pagerank has an extra hyper-parameter to tune with respect to Dirichlet. GSVD also obtains good results, especially on the bipartite and directed graphs which is coherent with the results of Chapter 4. On the other hand, despite being the fastest algorithm on all datasets (probably because of the small world property and the hard stopping criterion), Propagation yields some of the poorest results, especially on the bipartite and directed graphs. Finally, GCN gives globally very poor results while being slower than other algorithms by several orders of magnitude. Some explanation about this poor performance can be found in Hou et al. [2020], it is possible

that the GCN overfits the node attributes instead of actually learning from the graph structure.



algorithm	CO	CS	NG	WS	WV
GSVD + KNN	0.47 ± 0.08	0.30 ± 0.03	0.77 ± 0.02	0.33 ± 0.04	0.55 ± 0.02
Propagation	0.31 ± 0.09	0.28 ± 0.05	0.07 ± 0.00	0.04 ± 0.00	0.03 ± 0.01
Pagerank	0.54 ± 0.09	0.38 ± 0.04	0.62 ± 0.01	0.24 ± 0.03	0.50 ± 0.02
GCN	0.15 ± 0.02	0.20 ± 0.03		0.12 ± 0.02	0.24 ± 0.06
Heat Kernel	0.50 ± 0.09	0.37 ± 0.05	0.02 ± 0.01	0.19 ± 0.03	0.39 ± 0.04
Dirichlet	0.53 ± 0.09	0.39 ± 0.04	0.60 ± 0.06	0.22 ± 0.04	0.51 ± 0.03

Table 5.5: Macro F1-score (mean \pm standard deviation). : Time Out.



algorithm	CO	CS	NG	WS	WV
GSVD + KNN	0.10	0.11	1.89	0.18	0.62
Propagation	0.00	0.00	0.25	0.00	0.02
Pagerank	0.41	0.40	0.96	0.40	0.69
GCN	3.69	5.31		28.4	110
Heat Kernel	0.12	0.10	1.34	0.17	0.41
Dirichlet	0.04	0.05	1.54	0.23	0.74

Table 5.6: Running time for best score (seconds). : Time Out.

Chapter 6

Graph classification

This chapter is dedicated to graph classification. Given a collection of graphs, the goal is to infer the class of unlabeled ones based on a labeled training set. Standard applications are molecular compounds classification in biology [Childs et al., 2009, Kudo et al., 2005] or malware detection in cyber-security [Hu et al., 2009, Gascon et al., 2013, Kinable and Kostakis, 2011].

The rest of this chapter is organized as follows, Section 6.1 introduces the specificity of graph classification with respect to supervised classification on standard vector data, Section 6.2 presents some related work, Section 6.3 presents a first algorithm published in [de Lara and Pineau, 2018] while Section 6.4 presents a second algorithm published in [Pineau and de Lara, 2019a]. Finally, Section 6.5 covers the experiments.

References

- De Lara, N., & Pineau, E. (2018). *A simple baseline algorithm for graph classification*. Relational Representation Learning, NeurIPS 2018 Workshop.
- Pineau, E., & De Lara, N. (2019). *Variational recurrent neural networks for graph classification*. In Representation Learning on Graphs and Manifolds Workshop.

6.1 Challenges

Many natural or synthetic systems have a natural graph representation where entities are described through their mutual connections: chemical compounds (see Figure 6.1), social or biological networks, for example. Therefore, automatic mining of such structures is useful in a variety of applications. However, graph classification raises two main difficulties to leverage standard machine learning algorithms.

First, most of these algorithms take vectors of fixed size as inputs. In the case of graphs, usual representations such as edge list or adjacency matrix do not match this constraint. The size of the representations is graph dependent (number of edges in the first case, number of nodes squared in the second).

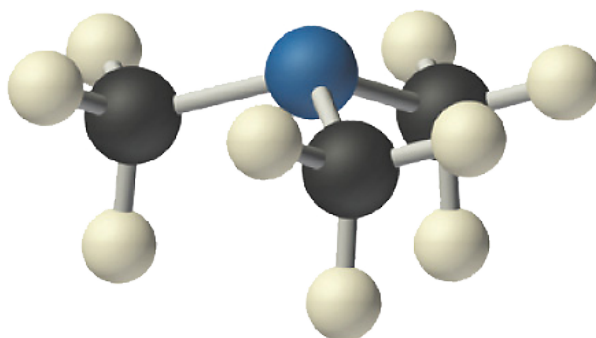


Figure 6.1: Trimethylamine molecule represented as a graph.

Secondly, these representations are index dependent: up to indexing of its nodes, a same graph admits $n!$ equivalent representations. In a classification task, the label of a graph is independent from the indices of its nodes, so the model used for prediction should be invariant to node ordering as well. Some neural network architectures have been specifically designed to tackle this issue for standard vector data such as clouds of points [Qi et al., 2017, Segol and Lipman, 2019]. However, these networks are not able to learn directly from graph data.

Section 6.2 presents some classic techniques to overcome these difficulties. Besides, due to this specific type of input, graph classification is not included in Scikit-network. However, our paper de Lara and Pineau [2018] has been implemented in another open-source graph library, *Karate Club* [Rozemberczki et al., 2020].

6.2 Related work

Graph classification methods can schematically be divided into three categories: graph kernels, sequential methods and embedding methods. In this section, we briefly present these different approaches, focusing on methods that only use the structure of the graph and no exogenous information, such as node features, to perform classification as we only want to compare the capacity of the algorithms to capture structural information.

Kernel methods Kernel methods [Nikolentzos et al., 2017a,b, 2018, Neumann et al., 2016] perform pairwise comparisons between the graphs of the dataset and apply a classifier, usually a support vector machine (SVM), on the similarity matrix. In order to maintain the number of comparisons tractable when the number of graphs is large, they often use Nyström algorithm [Williams and Seeger, 2001] to compute a low rank approximation of the similarity matrix. The key is to construct an efficient kernel that can be applied to graphs of varying sizes and captures useful features for the downstream classification.

Sequential methods Some methods tackle the varying sizes of graphs by processing them as a sequence of nodes. Earliest models used random walk based representations [Callut et al., 2008, Xu et al., 2012]. More recently, Jin and JaJa [2018] or You et al. [2018] transform a graph into a sequence of fixed size vectors, corresponding to its nodes, which is fed to a recurrent neural network. The two main challenges in this approach are the design of the embedding

function for the nodes and the order in which the embeddings are given to the recurrent neural network. The algorithm of Section 6.4 falls into this category.

Embedding methods Embedding methods [Gomez et al., 2017, Barnett et al., 2016, Dutta and Sahbi, 2017, Narayanan et al., 2017], derive a fixed number of features for each graph which is used as a vector representation for classification. Even though deriving a good set of features is often a difficult task, this approach has the benefit of being compatible with any standard classifier in a *plug and play* fashion (SVM, random forest, multilayer perceptron...). The model of Section 6.3 belongs to this class of methods as it relies on spectral features of the graph.

6.3 Spectral method

In this section, we propose a simple and fast algorithm based on the spectral decomposition of graph Laplacian to perform graph classification and get a first reference score for a dataset. We show in Section 6.5 that this method obtains competitive results compared to state-of-the-art algorithms. Note that, even though we present a very simple algorithm here, spectral methods have been incorporated into more complex ones with good experimental results [Chauhan et al., 2020, Verma and Zhang, 2017].

Let $G = (V, E)$ be an undirected and unweighted graph and $A \in \{0, 1\}^{n \times n}$ its Boolean adjacency matrix with respect to an arbitrary indexing of the nodes. G is assumed to be connected, otherwise, we extract its largest connected component. Let $D = \text{diag}(A\mathbf{1})$ be the matrix of node degrees, the normalized Laplacian of G is defined as

$$\mathcal{L} = I - D^{-1/2}AD^{-1/2}. \quad (6.1)$$

We use the k smallest positive eigenvalues of \mathcal{L} in ascending order as input of the classifier:

$$X = (\sigma_1, \dots, \sigma_k).$$

If the graph has less than k nodes, we use right zero padding to get a vector of appropriate dimensions: $X = (\sigma_1, \dots, \sigma_{n-1}, 0, \dots, 0)$.

The normalized Laplacian matrix of a graph is a well-known object in spectral learning [Belkin and Niyogi, 2002a, Kamvar et al., 2003]. However, for node clustering or classification most of the attention is usually directed to its eigenvectors and not its spectrum. A major benefit of the ordered spectrum representation for graph classification is that it does not depend on the indexing of the nodes.

Some Laplacian eigenvalues properties The eigenvalues of the normalized Laplacian matrix lie between 0 and 2. Such a property is very convenient for the downstream use of a standard classifier without heavy rescaling or preprocessing. The multiplicity of the eigenvalue 0 corresponds to the number of connected components in the graph, hence the omission of σ_0 in our representation as we only consider the largest connected component. Other values are also known to denote the presence of specific structures in the graph [Chung and Graham, 1997]. For example, an eigenvalue equal to 2 denotes a bipartite structure.

Physical interpretations In [Bonald et al. \[2018b\]](#), each eigenvalue of the Laplacian corresponds to the energy level of a *stable* configuration of the nodes in the embedding space. The lower the energy, the stabler the configuration. In [Shuman et al. \[2016\]](#), these eigenvalues correspond to frequencies associated to a Fourier decomposition of any signal living on the vertices of the graph. Thus, the truncation of the Fourier decomposition acts as low-pass filter on the signal. Characterizing a graph by the smallest eigenvalues of its normalized Laplacian is thus comparable to characterizing a melody by its lowest fundamental frequencies.

Finally, there have been some attempts to connect spectral decomposition to graph isomorphism [[Van Dam and Haemers, 2003](#), [Kolla et al., 2017](#)], however, to the best of our knowledge, this is still an open problem.

The choice of the classifier is left to the discretion of the user. In our experiments, we chose a random forest classifier (RFC) which offers a good computational speed versus accuracy trade-off.

An illustration of the model is proposed in figure 6.2.

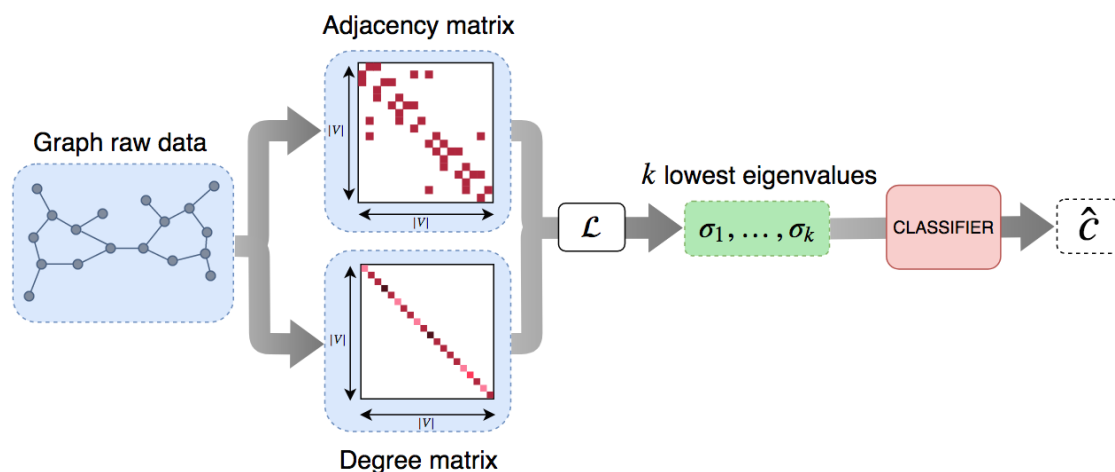


Figure 6.2: Spectral model. \mathcal{L} refers to equation (6.1) and \hat{c} is the predicted class.

6.4 Variational RNN method

In this section, we propose a method to sequentially embed graph information in order to perform classification. By construction, this recurrent graph classifier overcomes the common difficulties listed in Section 6.1. Besides, we propose to use an additional node prediction block to help the model to capture the intrinsic structure of the graphs. The complete model is denoted *variational recurrent graph classifier* (VRGC).

We propose to use a sequential approach to embed graphs with a variable number of nodes and edges into a vector space of a chosen dimension. This latent representation is then used for classification. Node index invariance is approximated through specific pre-processing and aggregation.

Let $G = (V, E)$ be an undirected and unweighted graph with V a set of nodes and E a set of edges. The graph G can be represented, modulo any permutation π over its nodes

$i \in \{1, \dots, n\}$, by its boolean adjacency matrix A^π such that $A_{ij}^\pi = 1$ if nodes indexed by i and j are connected in the graph and $A_{ij}^\pi = 0$ otherwise. We use this adjacency matrix as a raw representation of the graph.

The proposed VRGC model is composed of three main parts: node ordering and embedding, classification and regularization with variational auto-regression (VAR). See Figure 6.3 for an illustration.

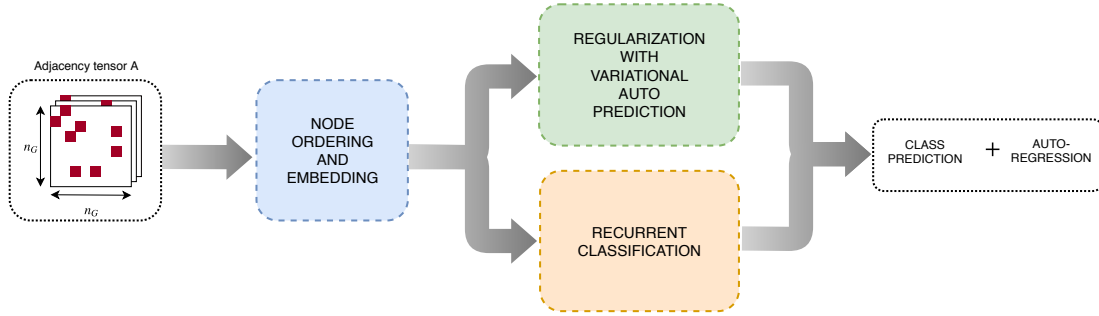


Figure 6.3: Macroscopic representation of VRGC.

Node ordering and embedding Before being processed by the neural network, the adjacency matrix of a graph is transformed on-the-fly [You et al., 2018]. First, a node is selected at random and used as root for a breadth first search (BFS) over the graph, as illustrated in Figure 6.4.

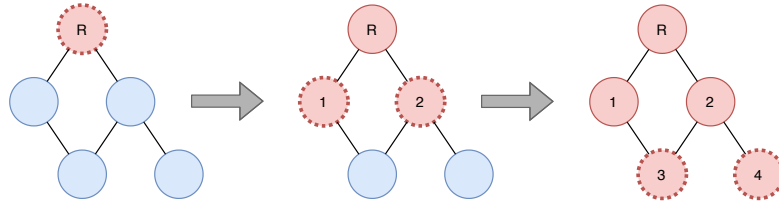


Figure 6.4: Node indexing using breadth first search. R: root node.

The rows and columns of the adjacency matrix are then reordered according to the sequence of nodes returned by the BFS. Next, each row i (corresponding to the i^{th} node in the BFS ordering) is truncated to keep only the connections of node i with the $\min(i, d)$ nodes that preceded in the BFS. This way, each node is d -dimensional, and each truncated matrix is zero-padded in order to have dimensions (n_G, d) , with n_G the size of the larger graph in the dataset. This can be seen as applying a sliding window of width d to the adjacency matrix. See Figure 6.5 for an illustration.

After node ordering and pre-embedding, each graph is processed as a sequence of d -dimensional nodes by a gated recurrent unit (GRU) neural network [Cho et al., 2014]. The GRU is a special RNN able to learn long term dependencies by solving vanishing gradient effect. The choice of GRU over Long Short Term Memory networks is arbitrary as they have equivalent long-term modeling power [Chung et al., 2014]. In order to help the recurrent network training, we propose to add a simple fully connected network between pre-embedding

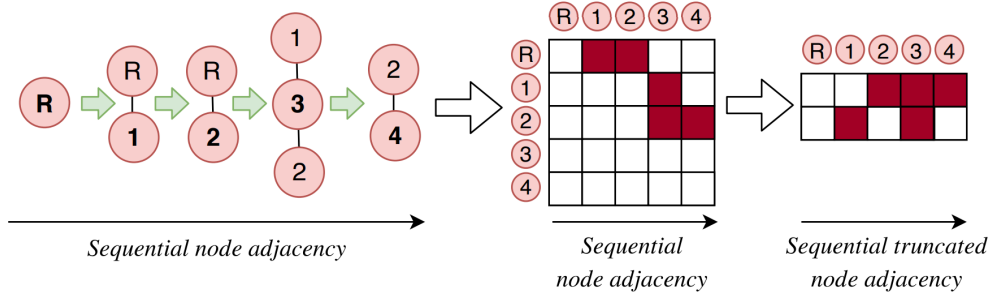


Figure 6.5: Truncation procedure from the root node R.

and recurrent embedding. Therefore, the node will be presented to the GRU in the shape of continuous vectors instead of binary adjacency vectors.

Finally the GRU sequentially embeds each node i by using $i - 1$ and information contained in a memory cell h_{i-1} that theoretically embeds all previously seen information. The embedded node sequence $\{h_i\}_{i=1}^{n_G}$ then feeds both the VAR and the classifier as discussed in subsequent sections. See Figure 6.6 for an illustration.

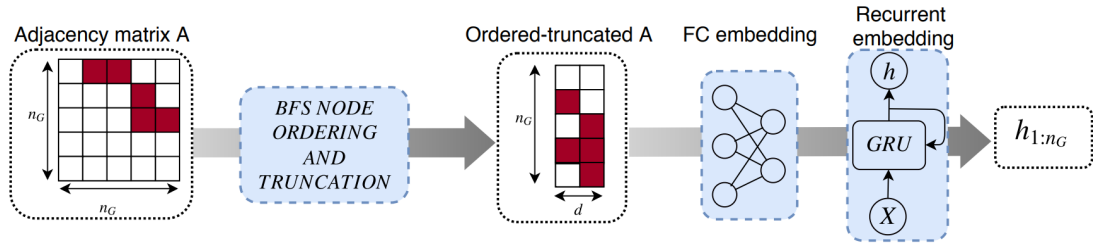


Figure 6.6: Node ordering and embedding.

Classification After the embedding step, we use an additional GRU dedicated to classification that takes $\{h_i\}_{i=1}^{n_G}$ as input. Its last memory cell, denoted \tilde{h}_{n_G} , feeds a softmax multilayer perceptron (MLP) which performs class prediction. Formally, let c be the class index, the classifier is trained by minimizing the cross-entropy loss between ground-truth and $\hat{p}(G, r)$ the softmax class membership probability vector for a given graph G that has been sorted by a BFS rooted with node r . We call this objective term $\mathcal{L}_{classif}$. As discriminating patterns might be spread across the whole graph, the network is required to model long-term dependencies. By construction, GRUs have such ability. See Figure 6.7 for an illustration.

Regularization with variational auto-regression As the structure of a graph is the concatenation of the interactions between all nodes and their respective neighbors, learning a good representation without using node attributes requires for the model to capture the structure of the graph while classifying. Accordingly, we add an auto-regression block to our model. Given a node to process, the network makes a prediction for the neighborhood of the next node. Multi-task learning is a powerful leverage to learn rich representation in NLP [Sanh et al., 2018]. In particular, such representation for sequence classification has already been used for sentiment analysis [Latif et al., 2017, Xu et al., 2017].

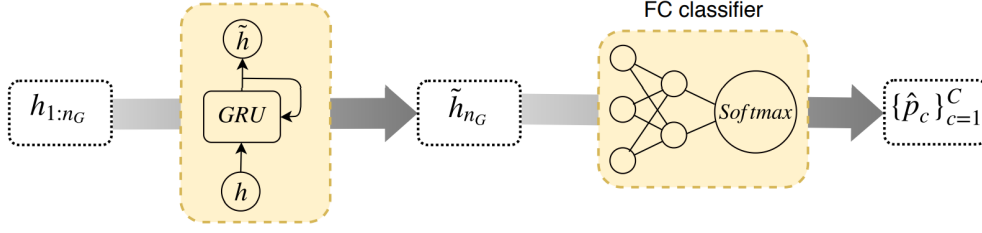


Figure 6.7: Classification block.

We use a variational auto-encoder (VAE) [Kingma and Welling, 2013] to learn a representation of each node i given h_{i-1} . The first layers of the encoder are shared with the classifier and corresponds to the graphs preprocessing (blue part in Figures 6.3 and 6.6). The subsequent encoder layers, the latent sampling and the decoder constitute the VAR. For each graph G with embedded nodes $i \in \{1, \dots, n_G\}$, the fully connected variational auto-encoder takes the latent states $\{h_i\}_{i \in \{1, \dots, n_G\}}$ as input. Let $\{z_i\}_{i \in \{1, \dots, n_G\}}$ be the latent random variables for the following model

$$p(G, z|h) = p(i = 1, z_1) \prod_{i=2}^{n_G} p_\theta(i|z_i) q_\phi(z_i|h_{i-1}).$$

In practice, p_θ and q_ϕ are modelled by neural networks parameterized by θ and ϕ , which require differentiable functions for training. However, $p_\theta(i|z_i)$ models a binary adjacency vector representing the connections between node i and previously visited nodes $j < i$. Therefore, we use sigmoid continuous relaxation to train our model, and hard binary sampling at test time. We use a Gaussian variational posterior distribution. Training is done by maximizing the variational lower bound of the log-likelihood of the observation as in Kingma's VAE.

The VAE-like loss for VAR regularization is the following:

$$\mathcal{L}_{pred} = \mathbb{E}_{p_d(G)} \left[\sum_{i=2}^{n_G} \text{KL} (q_\phi(z_i|h_{i-1}) || q(z_i)) \right] - \mathbb{E}_{p_d(G)} \left[\sum_{i=2}^{n_G} \mathbb{E}_{q_\phi(z_i|h_{i-1})} [\log p_\theta(i|z_i)] \right],$$

which is a lower bound of the negative marginal log-likelihood $\mathbb{E}_{p_d(G)} [\log p_\theta(G)]$. p_θ and q_ϕ are the respective densities of $i|z$ and $z|h$, whose distribution are parameterized by θ and ϕ , respectively. KL denotes the Kullback-Leibler divergence, p_d is the empirical distribution of G and $q(z_i)$ is the density of the prior distribution of latent variables $\{z_i\}_{i=2}^{n_G}$. We chose the standard Gaussian prior for $q(z_i)$.

The regularization part is illustrated in Figure 6.8.

In the end, the model is trained by minimizing the total loss $\mathcal{L} = \mathcal{L}_{classif} + \alpha \mathcal{L}_{pred}$, where α is a hyper-parameter.

Aggregation of the results at test time The node ordering step introduces randomness to our model. On the one hand, it helps learn more general graph representations during the training phase, but on the other hand, it might produce different outputs for the same graph during the testing phase, depending on the root of the BFS. In order to counter this side effect, we add the following aggregation step for the testing phase. Each graph is processed N times by the model with N different roots for BFS ordering. The N class membership probability vectors are

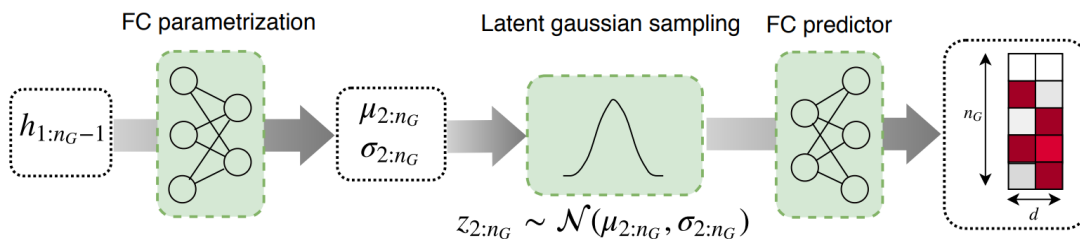


Figure 6.8: Regularization with VAR plus final aggregation.

extracted and averaged. The average score vector is noted \bar{p} and computed as follows with an element-wise sum:

$$\bar{p}(G) = \frac{1}{N} \sum_{r \sim \mathcal{U}(\llbracket 1, n_G \rrbracket)} \hat{p}(G, r).$$

This soft vote is repeated K times resulting in K probability vectors $\{\bar{p}_{\cdot, k}(G)\}_{k=1}^K$ for each graph G . The final class attributed to a graph corresponds to the highest probability among the K vectors. This second hard vote enables to choose the batch of votes for which the model is the most confident:

$$\hat{c}(G) = \arg \max_{c \in \llbracket 1, C \rrbracket} \{ \|\bar{p}_{c, \cdot}(G)\|_{\infty} \}.$$

6.5 Experiments

This section is dedicated to our experiments on graph classification. First, we introduce the datasets in Section 6.5.1 and the results in sections 6.5.2, 6.5.3, 6.5.4 and 6.5.5.

6.5.1 Datasets

We evaluated our model against four standard datasets from biology: Mutag (MT), Enzymes (EZ), Proteins Full (PF) and National Cancer Institute (NCI1) [Kersting et al., 2016]. Table 6.1 presents some characteristics of these datasets.

	MT	EZ	PF	NCI1
# graphs	188	600	1113	4110
# classes	2	6	2	2
bias	0.66	0.17	0.60	0.5
avg. $ V $	18	33	39	30
min $ V $ / max $ V $	10/28	2/125	4/620	3/106
avg. $ E $	39	124	146	64.6

Table 6.1: Basic characteristics of the datasets. Bias indicates the proportion of the largest class.

Step	Architecture
BFS	1-layer FC. $d_n \times 64$
Embedding	2-layer GRU. 64×128
VAR	<u>Encoder</u> 1-layer FC. $128 \times 2 \times 8$ Gaussian sampling <u>Predictor</u> 2-layer ReLU FC. $8 \times d_n$
Classifier	2-layer GRU. $128 \times 128 + \text{DP}(0.25)$ 2-layer ReLU FC. $128 \times C + \text{SoftMax}$

Table 6.2: Generic architecture used in our experiments. ReLU FC stands for fully-connected network with ReLU activation. DP stands for dropout.

All graphs represent chemical compounds, nodes are molecular substructures (typically atoms) and edges represent connections between these substructures (chemical bond or spatial proximity). In MT, the compounds are either mutagenic or not mutagenic. EZ contains tertiary structures of proteins from the 6 Enzyme Commission top level classes; it is the only multiclass dataset of this section. PF is a subset of the Dobson and Doig dataset representing secondary structures of proteins being either enzyme or not enzyme. In NCI1, compounds either have an anti-cancer activity or do not.

6.5.2 General benchmark

Each dataset is divided into 10 folds such that the class proportions are preserved in each fold for all datasets. These folds are then used for cross-validation i.e, one fold serves as the testing set while the other ones compose the training set. Results are averaged over all testing sets. We built the folds using Scikit-learn [Pedregosa et al. \[2011b\]](#) *StratifiedKFold* function with the random seed fixed to 1 in order to get reproducible results.

Spectral method For the general benchmark, the embedding dimension is set to the average number of nodes for each dataset. Another experiment illustrates the influence of k , see [Table 6.5](#). We use a unique set of hyper-parameters for the classifier is used for all datasets. We used the random forest classifier from Scikit-learn with *class_weights*: balanced. The other non-default hyper parameters were selected by randomized cross validation over the different datasets: 500 decision trees with 1 sample minimum per leaf, a maximum depth of 100 and use of bootstrap.

VRGC This model is implemented in Pytorch [[Paszke et al., 2017](#)] and trained with the Adam stochastic optimization method [[Kingma and Ba, 2014](#)] in the server setup [1.3.2](#). The input size d_n of the recurrent neural network is chosen for each dataset according to the algorithm described in [[You et al., 2018](#)], namely 11 for MT, 25 for EZ, 80 for PF and 11 for NCI1. α is set to 0.1. For training, batch size is set to 64, and the learning rate to 10^{-3} , decreased by 0.3 at iterations 400 and 1000. We use the same hyper-parameters for every dataset. [Table 6.2](#) details the architecture of the network.

We compare our results to those obtained by Earth Mover’s Distance [[Nikolentzos et al., 2017b](#)] (EMD), Pyramid Match [[Nikolentzos et al., 2017b](#)] (PM), Feature-Based [[Barnett et al.,](#)

2016] (FB), Dynamic-Based Features [Gomez et al., 2017] (DyF), Stochastic Graphlet Embedding [Dutta and Sahbi, 2017] (SGE) and Family of Graph Spectral Distances [Verma and Zhang, 2017] (FGSD).

All values are directly taken from the aforementioned papers as they use a setup similar to ours. For algorithms presenting results with and without node features, we reported the results without node features. For those presenting results with several sets of hyper-parameters, we reported the results for the parameters that performed best on the largest number of datasets. Results are reported in Table 6.3.

VRGC obtains state-of-the-art results on three out of these four datasets and the second best result on the fourth one. However, its standard deviation is usually higher than for other algorithms. The spectral method also gets competitive scores while it is much faster to train.

	MT	EZ	PF	NCI1
EMD	86.1 ± 0.8	36.8 ± 0.8	-	72.7 ± 0.2
PM	85.6 ± 0.6	28.2 ± 0.4	-	69.7 ± 0.1
FB	84.7 ± 2.0	29.0 ± 1.2	70.0 ± 1.3	62.9 ± 1.0
DyF	86.3 ± 1.3	26.6 ± 1.2	73.1 ± 0.4	66.6 ± 0.3
SGE	87.3	40.7	71.9	-
FGSD	92.1	-	73.4	79.8
Spectral	88.4 ± 7.0	43.7 ± 6.3	73.6 ± 3.5	75.2 ± 2.1
VRGC	86.3 ± 8.6	48.4 ± 6.2	74.8 ± 3.0	80.7 ± 2.2

Table 6.3: Mean accuracy (%) and standard deviation.

6.5.3 Influence of the classifier for the spectral model

Besides RFC, we experimented with different standard classifiers combined to our spectral embedding. Namely: k -nearest neighbors classifier (k NN), 2-layers perceptron with Relu non-linearity (MLP), support vector machine with *one versus one* classification (SVM) and ridge regression classifier (RRC). Results are reported in Table 6.4.

	MT	PTC	EZ	PF	DD	NCI1
RFC	88.4	62.8	43.7	73.6	75.4	75.2
1NN	86.8	59.3	37.3	65.6	69.6	68.3
15NN	85.7	61.9	33.7	70.4	75.0	69.6
MLP	86.3	60.5	31.8	71.6	75.6	62.3
SVM	85.3	60.8	31.3	73.0	75.0	63.9
RRC	84.2	59.6	26.7	71.5	75.0	62.2

Table 6.4: Mean accuracy (%) of some classifiers combined to the spectral model.

As we can see, RFC provides the best results for all datasets except DD where MLP has an accuracy of 75.6 against 75.4. Our intuition to explain these good results is that the decision tree classifier, which is at the core of RFC, is an algorithm based on level thresholding. As explained

in section 6.3, our embedding represents a sequence of energy levels, being above or below a certain level is thus likely to be meaningful for classification.

6.5.4 Influence of k for the spectral model

We experimented with different embedding dimensions for RFC: $k \in \{1, 5, 10, 25, 50\}$. The hyper-parameters are the same as in the other experiments. Results are reported in Table 6.5.

k	MT	PTC	EZ	PF	DD	NCI1
1	76.2	56.1	23.8	64.0	57.2	58.2
5	86.8	62.5	39.0	69.6	73.9	72.5
10	86.8	61.4	42.8	71.7	75.5	75.5
25	88.4	62.8	42.7	72.8	75.7	75.2
50	88.4	62.8	43.7	73.6	75.1	75.2

Table 6.5: Mean accuracy (%) of the spectral model for different dimensions.

We see that even the first energy level is sufficient to obtain a non-trivial classification. $k = 5$ provides results competitive with the state of the art while $k = 50$ provides results relatively similar to $k = \text{avg}(|V|)$. We did not experiment with larger values of k as it would mostly result into additional zero padding for most graphs. Note that, embedding all graphs for $k = 50$ took less than a minute in the laptop setting.

6.5.5 Node indexing invariance for VRGC

The model is designed to be independent from node ordering of the graph with respect to different BFS roots. Inputs representing the same graph (up to node ordering) should be close from one another in the latent embedding space. As the preprocessing is performed on each graph at each epoch, a same graph is processed many times by the model during training with different embeddings. This creates a natural regularization for the network.

We illustrate this in Figure 6.9. We embed five graphs from the dataset EZ with 20 different BFS each and plot their TSNE projections. As we can see, the projections corresponding to the same graphs form a heap in the low dimensional representation of the latent space.

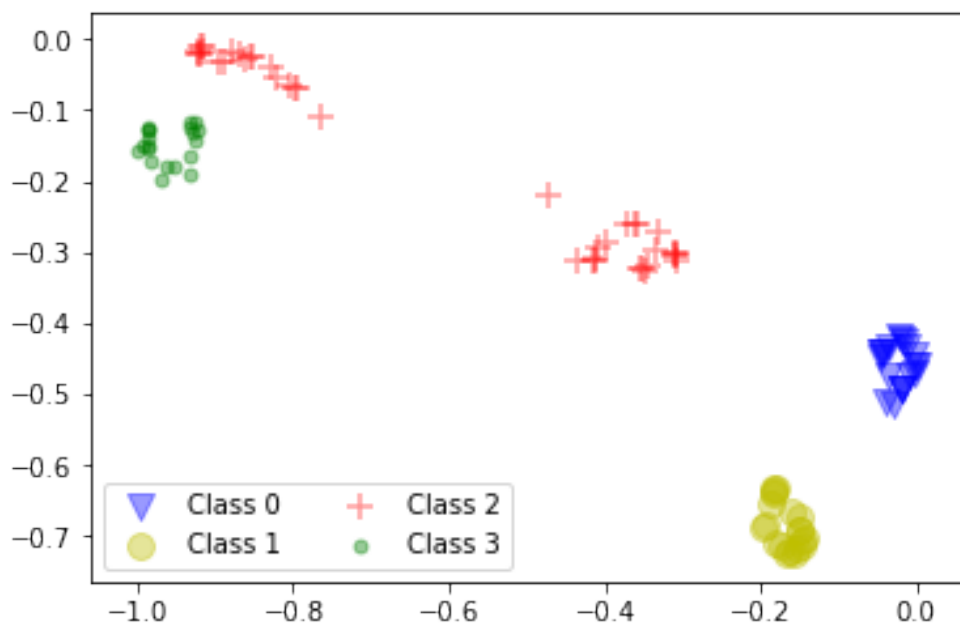


Figure 6.9: TSNE projection of the latent state preceding classification.

Chapter 7

Conclusions and perspectives

In this work, we have sketched a map of modern graph mining and related challenges.

First, we addressed the data representation problem. In particular, we explained how linear operators combined with advanced linear algebra techniques could help make various graph algorithms scale to industrial levels. The next step would be to study these techniques in the context of distributed data and computing power.

Then, we introduced Scikit-network, an open-source Python library for large scale graph mining inspired by Scikit-learn. In the future, we hope to keep integrating new features such as anomaly detection or link prediction and to continue improving the package.

We have provided a simple explanation for the well-known benefits of regularization on spectral embedding. Specifically, regularization forces the embedding to focus on the largest clusters, making the embedding more robust to noise. This result was obtained through the explicit characterization of the embedding for a simple block model and extended to bipartite graphs. An interesting perspective of this work is the extension to *stochastic* block models, using, for instance, the concentration results proved in [Lei et al., 2015, Le et al., 2017]. Another area of interest is the impact of regularization on other downstream tasks such as link prediction. Finally, we would like to further explore the impact of the regularization parameter while exploiting the theoretical results presented in this section.

We have proposed a novel embedding based on the Generalized Singular Value Decomposition, which applies to undirected, directed, and bipartite graphs. We have explained how the distances in the embedding space could be easily interpreted in terms of neighborhood distribution for the best rank- k approximation of the graph. Efficiency of this embedding has been demonstrated on real datasets for both node clustering and node classification. For this new and standard spectral algorithm, we cannot stress enough the importance of normalization. Projecting each node vector onto the unit sphere drastically improves performance for all the downstream tasks we have experimented with.

We showed how the classic modularity function for graph clustering extends to a large family of functions based on graph sampling. We then explained how optimizing such a function corresponds to fitting a simple average model to the data through the Kullback Leibler divergence. We illustrated how the Louvain heuristic can be used to perform greedy maximization for all

these modularity functions. Future work could include the study of soft-clustering membership based on Louvain algorithms as graph embeddings.

We have proposed a novel approach to node classification based on heat diffusion. Specifically, we propose to center the temperatures of each solution to the Dirichlet problem before classification. We have proved the consistency of this algorithm on a simple block model and we have shown that it drastically improves classification performance on real datasets with respect to the vanilla version. In future work, we plan to extend this algorithm to soft classification, using the centered temperatures to get a confidence score for each node of the graph. Another interesting research perspective is to extend our proof of consistency of the algorithm to *stochastic* block models.

We explained the specific challenges of the graph classification problem. Then, we described and evaluated two different approaches to overcome them. However, we believe that further refinement of these algorithms should be more context dependent and include domain knowledge. For example, classifying molecular compounds is not the same as identifying malware programs.

Finally, we wish to study the impact simplex projection for membership matrices. In this work, we always relied on naive scaling to normalize either soft-clustering memberships or classification confidence. However, such scaling does not yield the closest simplex point to the membership, hence this new projection could lead to different results which are worth exploring. In particular, the simplex projection is known to yield sparser vectors than the naive scaling.

Bibliography

- E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic block-models. *Journal of machine learning research*, 9(Sep):1981–2014, 2008.
- L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.
- A. A. Amini, A. Chen, P. J. Bickel, E. Levina, et al. Pseudo-likelihood methods for community detection in large sparse networks. *The Annals of Statistics*, 41(4):2097–2122, 2013.
- W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, 9(1):17–29, 1951.
- M. J. Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76(6):066102, Dec. 2007. doi: 10.1103/PhysRevE.76.066102. URL <https://link.aps.org/doi/10.1103/PhysRevE.76.066102>.
- I. Barnett, N. Malik, M. L. Kuijjer, P. J. Mucha, and J.-P. Onnela. Feature-based classification of networks. *arXiv preprint arXiv:1610.05868*, 2016.
- S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.
- M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002a.
- M. Belkin and P. Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15:1373–1396, 2002b. doi: 10.1162/089976603321780317.
- M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002c.
- N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on cuda. Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.
- R. M. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize. *KorBell Team’s Report to Netflix*, 2007.
- J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- D. Berberidis, A. N. Nikolakopoulos, and G. B. Giannakis. Adadif: Adaptive diffusions for efficient semi-supervised learning over graphs. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 92–99. IEEE, 2018.

- V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, Oct. 2008. ISSN 1742-5468. doi: 10.1088/1742-5468/2008/10/P10008. URL <https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008>.
- T. Bonald and N. De Lara. The forward-backward embedding of directed graphs. *openreview*, 2018.
- T. Bonald, B. Charpentier, A. Galland, and A. Hollocou. Hierarchical graph clustering using node pair sampling. *arXiv preprint arXiv:1806.01664*, 2018a.
- T. Bonald, A. Hollocou, and M. Lelarge. Weighted spectral embedding of graphs. *arXiv preprint arXiv:1809.11115*, 2018b.
- T. Bonald, N. de Lara, Q. Lutz, and B. Charpentier. Scikit-network: Graph analysis in python. *Journal of Machine Learning Research*, 21(185):1–6, 2020.
- U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *arXiv preprint physics/0608255*, 2006.
- M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.
- H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.
- J. Callut, K. Françoisse, M. Saerens, and P. Dupont. Classification in graphs using discriminative random walks, 2008.
- K. Chaudhuri, F. Chung, and A. Tsiatas. Spectral clustering of graphs with general degrees in the extended planted partition model. In *Conference on Learning Theory*, pages 35–1, 2012.
- J. Chauhan, D. Nathani, and M. Kaul. Few-shot learning on graphs via super-classes based on graph spectral measures. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=Bkeeca4Kvr>.
- M. Chen, Q. Yang, X. Tang, et al. Directed graph embedding. In *IJCAI*, pages 2707–2712, 2007.
- L. Childs, Z. Nikoloski, P. May, and D. Walther. Identification and classification of ncra molecules using graph properties. *Nucleic acids research*, 37(9):e66–e66, 2009.
- K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- F. Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.
- F. R. Chung. *Spectral graph theory*. American Mathematical Soc., 1997.
- F. R. Chung and F. C. Graham. *Spectral graph theory*. Number 92 in -. American Mathematical Soc., 1997.
- J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL <http://igraph.org>.
- N. De Lara. The sparse+ low rank trick for matrix factorization-based graph algorithms. In *Proceedings of the 15th International Workshop on Mining and Learning with Graphs (MLG)*, 2019.
- N. De Lara and T. Bonald. A Consistent Diffusion-Based Algorithm for Semi-Supervised Classification on Graphs. working paper or preprint, Aug. 2020. URL <https://hal.archives-ouvertes.fr/hal-02922980>.
- N. de Lara and E. Pineau. A Simple Baseline Algorithm for Graph Classification. *arXiv:1810.09155 [cs, stat]*, Oct. 2018. URL <http://arxiv.org/abs/1810.09155>. arXiv: 1810.09155.
- C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *arXiv preprint arXiv:1910.02370*, 2019.
- I. 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, 2001.
- C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1320–1329. ACM, 2018.
- N. Dugué and A. Perez. *Directed Louvain: maximizing modularity in directed networks*. PhD thesis, Université d’Orléans, 2015.
- A. Dutta and H. Sahbi. High order stochastic graphlet embedding for graph-based pattern recognition. *arXiv preprint arXiv:1702.00156*, 2017.
- L. M. Ewerbring et al. Canonical correlations and generalized svd: applications and new algorithms. In *Advances in Parallel Computing*, volume 1, pages 37–52. Elsevier, 1990.
- M. Fey, J. Eric Lenssen, F. Weichert, and H. Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.
- F. Fous, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering*, 19(3):355–369, 2007.
- E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569, 1983.
- T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.
- T. Funke, T. Guo, A. Lancic, and N. Antulov-Fantulin. Statistical manifold embedding for directed graphs. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.
- H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 45–54, 2013.

- L. G. Gomez, B. Chiem, and J.-C. Delvenne. Dynamics based features for graph classification. *arXiv preprint arXiv:1705.10817*, 2017.
- A. Grover and J. Leskovec. node2vec: Scalable Feature Learning for Networks. *KDD : proceedings. International Conference on Knowledge Discovery & Data Mining*, 2016:855–864, Aug. 2016. ISSN 2154-817X. doi: 10.1145/2939672.2939754. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5108654/>.
- R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004.
- A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *arXiv:0909.4061 [math]*, Sept. 2009. URL <http://arxiv.org/abs/0909.4061>. arXiv: 0909.4061.
- C. Haruechaiyasak and C. Damrongrat. Article recommendation based on a topic model for wikipedia selection for schools. In *International Conference on Asian Digital Libraries*, pages 339–342. Springer, 2008.
- P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.
- A. Hollocoou, T. Bonald, and M. Lelarge. Modularity-based Sparse Soft Graph Clustering. In *AISTATS 2019 - 22nd International Conference on Artificial Intelligence and Statistics*, Naha, Okinawa, Japan, Apr. 2019. URL <https://hal.archives-ouvertes.fr/hal-02005331>.
- Y. Hou, J. Zhang, J. Cheng, K. Ma, R. T. B. Ma, H. Chen, and M.-C. Yang. Measuring and improving the use of graph information in graph neural networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rkeI1kHKvS>.
- X. Hu, T.-c. Chiueh, and K. G. Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 611–620, 2009.
- W. Huang, T. Zhang, Y. Rong, and J. Huang. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems*, pages 4558–4567, 2018.
- L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.
- Y. Jin and J. F. JaJa. Learning graph-level representations with gated recurrent neural networks. *arXiv preprint arXiv:1805.07683*, 2018.
- A. Joseph, B. Yu, et al. Impact of regularization on spectral clustering. *The Annals of Statistics*, 44(4):1765–1791, 2016.
- K. Kamvar, S. Sepandar, K. Klein, D. Dan, M. Manning, and C. Christopher. Spectral learning. In *International Joint Conference of Artificial Intelligence*. Stanford InfoLab, 2003.
- K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels, 2016. <http://graphkernels.cs.tu-dortmund.de>.

- Y. Kim, S.-W. Son, and H. Jeong. Finding communities in directed networks. *Physical Review E*, 81(1):016103, 2010.
- J. Kinable and O. Kostakis. Malware classification based on call graph clustering. *Journal in computer virology*, 7(4):233–245, 2011.
- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.
- A. Kolla, Y. Koutis, V. Madan, and A. K. Sinop. Spectral graph isomorphism. -, 2017.
- R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pages 315–322, 2002.
- T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in neural information processing systems*, pages 729–736, 2005.
- J. Kunegis. KONECT: The Koblenz Network Collection. In *Proceedings of the 22Nd International Conference on World Wide Web, WWW '13 Companion*, pages 1343–1350, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2038-2. doi: 10.1145/2487788.2488173. URL <http://doi.acm.org/10.1145/2487788.2488173>. event-place: Rio de Janeiro, Brazil.
- R. Lambiotte, J.-C. Delvenne, and M. Barahona. Laplacian dynamics and multiscale modular structure in networks. *arXiv preprint arXiv:0812.1770*, 2008.
- C. Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.
- N. D. Lara and T. Bonald. Spectral embedding of regularized block models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1l_0JBYwS.
- S. Latif, R. Rana, J. Qadir, and J. Epps. Variational autoencoders for learning latent representations of speech emotion. *arXiv preprint arXiv:1712.08708*, 2017.
- C. M. Le, E. Levina, and R. Vershynin. Concentration and regularization of random graphs. *Random Structures & Algorithms*, 51(3):538–561, 2017.
- J. Lei, A. Rinaldo, et al. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics*, 43(1):215–237, 2015.
- J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- Q. Li, S. An, L. Li, and W. Liu. Semi-supervised learning on graph with an alternating diffusion process. *CoRR*, abs/1902.06105, 2019. URL <http://arxiv.org/abs/1902.06105>.

- F. Lin and W. W. Cohen. Semi-supervised classification of network data using very few labels. In *2010 International Conference on Advances in Social Networks Analysis and Mining*, pages 192–199. IEEE, 2010.
- X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 273–282, 2013.
- H. Ma, H. Yang, M. R. Lyu, and I. King. Mining social networks using heat diffusion processes for marketing candidates selection. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 233–242. ACM, 2008.
- H. Ma, I. King, and M. R. Lyu. Mining web graphs for recommendations. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):1051–1064, 2011.
- J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1.14, pages 281–297. Oakland, CA, USA, 1967.
- M. Marchiori and V. Latora. Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 285(3-4):539–546, 2000.
- R. T. McGibbon and N. J. Smith. A platform tag for portable linux built distributions. PEP 513, -, 2016. URL <https://www.python.org/dev/peps/pep-0513/>.
- E. Merkurjev, A. L. Bertozzi, and F. Chung. A semi-supervised heat kernel pagerank mbo algorithm for data classification. Technical report, University of California, Los Angeles Los Angeles United States, 2016.
- A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005, 2017. URL <http://arxiv.org/abs/1707.05005>.
- M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245, 2016.
- M. E. Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006a.
- M. E. Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94(5):052315, 2016.
- M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.
- M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, June 2006b. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.0601602103. URL <https://www.pnas.org/content/103/23/8577>.
- A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.
- G. Nikolentzos, P. Meladianos, A. J.-P. Tixier, K. Skianis, and M. Vazirgiannis. Kernel graph convolutional neural networks. *arXiv preprint arXiv:1710.10689*, 2017a.

- G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI*, pages 2429–2435, 2017b.
- G. Nikolentzos, P. Meladianos, S. Limnios, and M. Vazirgiannis. A degeneracy framework for graph similarity. In *IJCAI*, pages 2595–2601, 2018.
- S. M. Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.
- M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.
- L. Page. Method for node ranking in a linked database, Sept. 4 2001. US Patent 6,285,999.
- L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999a. URL <http://ilpubs.stanford.edu:8090/422/>. Previous number = SIDL-WP-1999-0120.
- L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web., Nov. 1999b. URL <http://ilpubs.stanford.edu:8090/422/>.
- A. Paszke, S. Gross, S. Chintala, and G. Chanan. Pytorch, 2017.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011a. ISSN ISSN 1533-7928. URL <http://www.jmlr.org/papers/v12/pedregosa11a.html>.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011b.
- T. P. Peixoto. The graph-tool python library. *figshare*, 2014. doi: 10.6084/m9.figshare.1164194. URL http://figshare.com/articles/graph_tool/1164194.
- B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.
- E. Pineau and N. de Lara. Variational recurrent neural networks for graph classification. In *Representation Learning on Graphs and Manifolds Workshop*, 2019a.
- E. Pineau and N. de Lara. Graph Classification with Recurrent Variational Neural Networks. *arXiv:1902.02721 [cs, stat]*, Feb. 2019b. URL <http://arxiv.org/abs/1902.02721>. arXiv: 1902.02721.
- C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.
- T. Qin and K. Rohe. Regularized spectral clustering under the degree-corrected stochastic block-model. In *Advances in Neural Information Processing Systems*, pages 3120–3128, 2013.

- H. Qiu and E. R. Hancock. Clustering and embedding using commute times. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1873–1890, 2007.
- U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.
- A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.
- S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.
- B. Rozemberczki, O. Kiss, and R. Sarkar. An api oriented open-source python framework for unsupervised learning on graphs, 2020.
- M. Saerens, F. Fouss, L. Yen, and P. Dupont. The Principal Components Analysis of a Graph, and Its Relationships to Spectral Clustering. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Machine Learning: ECML 2004*, Lecture Notes in Computer Science, pages 371–383. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-30115-8.
- V. Sanh, T. Wolf, and S. Ruder. A hierarchical multi-task approach for learning embeddings from semantic tasks. *arXiv preprint arXiv:1811.06031*, 2018.
- S. E. Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.
- N. Segol and Y. Lipman. On universal equivariant set networks. *arXiv preprint arXiv:1910.02421*, 2019.
- D. I. Shuman, B. Ricaud, and P. Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.
- M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.
- P. Snell and P. Doyle. Random walks and electric networks. *Free Software Foundation*, 2000.
- M. Speriosu, N. Sudan, S. Upadhyay, and J. Baldridge. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 53–63. Association for Computational Linguistics, 2011.
- D. A. Spielman. Spectral graph theory and its applications. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 29–38. IEEE, 2007.
- J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015a.
- J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale Information Network Embedding. *Proceedings of the 24th International Conference on World Wide Web - WWW '15*, pages 1067–1077, 2015b. doi: 10.1145/2736277.2741093. URL <http://arxiv.org/abs/1503.03578>. arXiv: 1503.03578.

- J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
- D. Thanou, X. Dong, D. Kressner, and P. Frossard. Learning heat diffusion graphs. *IEEE Transactions on Signal and Information Processing over Networks*, 3(3):484–499, 2017.
- V. A. Traag, P. Van Dooren, and Y. Nesterov. Narrow scope for resolution-limit-free community detection. *Physical Review E*, 84(1):016114, 2011.
- N. Tremblay and P. Borgnat. Graph wavelets for multiscale community mining. *IEEE Transactions on Signal Processing*, 62(20):5227–5239, 2014.
- E. R. Van Dam and W. H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its applications*, 373:241–272, 2003.
- S. Verma and Z.-L. Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pages 88–98, 2017.
- N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.
- P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272, 2020. doi: <https://doi.org/10.1038/s41592-019-0686-2>.
- U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, Dec. 2007. ISSN 1573-1375. doi: 10.1007/s11222-007-9033-z. URL <https://doi.org/10.1007/s11222-007-9033-z>.
- S. J. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.
- M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. URL <https://arxiv.org/abs/1909.01315>.
- J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.
- D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 1998.
- S. White and P. Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 274–285. SIAM, 2005.
- A. Wijesinghe and Q. Wang. Dfnets: Spectral cnns for graphs with feedback-looped filters. In *Advances in Neural Information Processing Systems*, pages 6007–6018, 2019.
- C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.

- S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.
- W. Xu, H. Sun, C. Deng, and Y. Tan. Variational autoencoder for semi-supervised text classification. In *AAAI*, pages 3358–3364, 2017.
- X. Xu, L. Lu, P. He, Z. Pan, and C. Jing. Protein classification using random walk on graph. In *International Conference on Intelligent Computing*, pages 180–184. Springer, 2012.
- S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 29(1):40–51, 2006.
- Y. Yang, R. N. Lichtenwalter, and N. V. Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, 2015.
- J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. Graphrnn: A deep generative model for graphs. *arXiv preprint arXiv:1802.08773*, 2018.
- O. Younis, M. Krunz, and S. Ramasubramanian. Node clustering in wireless sensor networks: Recent developments and deployment challenges. *IEEE network*, 20(3):20–25, 2006.
- W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li. Single-pass pca of large high-dimensional data. *arXiv preprint arXiv:1704.07669*, 2017.
- W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.
- P. Zhang, F. Wang, J. Hu, and R. Sorrentino. Label propagation prediction of drug-drug interactions based on clinical side effects. *Scientific reports*, 5(1):1–10, 2015.
- Y. Zhang and K. Rohe. Understanding regularized spectral clustering via graph conductance. In *Advances in Neural Information Processing Systems*, pages 10631–10640, 2018.
- X. Zhu. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, 2005.
- X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

Titre : Contributions algorithmiques et logicielle à l'apprentissage automatique sur les graphes

Mots clés : graphes, apprentissage automatique, matrices creuses

Résumé : Depuis l'invention du PageRank par Google pour les requêtes Web à la fin des années 1990, les algorithmes de graphe font partie de notre quotidien. Au milieu des années 2000, l'arrivée des réseaux sociaux a amplifié ce phénomène, élargissant toujours plus les cas d'usage de ces algorithmes. Les relations entre entités peuvent être de multiples sortes : relations symétriques *utilisateur-utilisateur* pour Facebook ou LinkedIn, relations asymétriques *follower-followee* pour Twitter, ou encore, relations bipartites *utilisateur-contenu* pour Netflix ou Amazon. Toutes soulèvent des problèmes spécifiques et les applications sont nombreuses : calcul de centralité pour la mesure d'influence, le partitionnement de nœuds pour la fouille de données, la classification de nœuds pour les recommandations ou l'embedding pour la prédiction de liens en sont quelques exemples.

En parallèle, les conditions d'utilisation des algorithmes de graphe sont devenues plus contraignantes. D'une part, les jeux de données toujours plus gros avec des millions d'entités et parfois des milliards de relations limite la complexité asymptotique des algorithmes pour les applications industrielles. D'autre part, dans la mesure où ces algorithmes

influencent nos vies, les exigences d'*explicabilité* et d'*équité* dans le domaine de l'intelligence artificielle augmentent. Les algorithmes de graphe ne font pas exception à la règle. L'Union européenne a par exemple publié un guide de conduite pour une IA fiable. Ceci implique de pousser encore plus loin l'analyse des modèles actuels, voire d'en proposer de nouveaux.

Cette thèse propose des réponses ciblées via l'analyse d'algorithmes classiques, mais aussi de leurs extensions et variantes, voire d'algorithmes originaux. La capacité à passer à l'échelle restant un critère clé. Dans le sillage de ce que le projet Scikit-learn propose pour l'apprentissage automatique sur données vectorielles, nous estimons qu'il est important de rendre ces algorithmes accessibles au plus grand nombre et de démocratiser la manipulation de graphes. Nous avons donc développé un logiciel libre, Scikit-network, qui implémente et documente ces algorithmes de façon simple et efficace. Grâce à cet outil, nous pouvons explorer plusieurs tâches classiques telles que l'embedding de graphe, le partitionnement, ou encore la classification semi-supervisée.

Title : Algorithmic and software contributions to graph mining

Keywords : graphs, machine learning, sparse matrices

Abstract : Since the introduction of Google's PageRank method for Web searches in the late 1990s, graph algorithms have been part of our daily lives. In the mid 2000s, the arrival of social networks has amplified this phenomenon, creating new use-cases for these algorithms. Relationships between entities can be of multiple types: *user-user* symmetric relationships for Facebook or LinkedIn, *follower-followee* asymmetric ones for Twitter or even *user-content* bipartite ones for Netflix or Amazon. They all come with their own challenges and the applications are numerous: centrality calculus for influence measurement, node clustering for knowledge discovery, node classification for recommendation or embedding for link prediction, to name a few.

In the meantime, the context in which graph algorithms are applied has rapidly become more constrained. On the one hand, the increasing size of the datasets with millions of entities, and sometimes billions of relationships, bounds the asymptotic complexity of the algorithms for industrial applications. On the other

hand, as these algorithms affect our daily lives, there is a growing demand for *explainability* and *fairness* in the domain of artificial intelligence in general. Graph mining is no exception. For example, the European Union has published a set of ethics guidelines for trustworthy AI. This calls for further analysis of the current models and even new ones.

This thesis provides specific answers via a novel analysis of not only standard, but also extensions, variants, and original graph algorithms. Scalability is taken into account every step of the way. Following what the Scikit-learn project does for standard machine learning, we deem important to make these algorithms available to as many people as possible and participate in graph mining popularization. Therefore, we have developed an open-source software, Scikit-network, which implements and documents the algorithms in a simple and efficient way. With this tool, we cover several areas of graph mining such as graph embedding, clustering, and semi-supervised node classification.