Hypernode graphs for learning from binary relations between sets of objects

Thomas Ricatte

Ecole Doctorale ED072 "Sciences Pour l'Ingénieur"

Hypernode graphs for learning from binary relations between sets of objects

Thèse préparée par **Thomas Ricatte**
pour l'obtention du grade de :

Docteur de l'Université de Lille

Domaine : **Informatique**

Soutenue le 23/01/2015 (Mention Très Honorable)

Centre de Recherche en Informatique, Signal et Automatique de Lille
(CRIStAL - UMR CNRS 9189)

Composition du Jury :

Yannick **Cras**	Chief Development at SAP SE	Examinateur
François **Denis**	Professeur à l'Université d'Aix-Marseille	Rapporteur
Gemma **Garriga**	Data Scientist at Allianz SE	Examinateur
Rémi **Gilleron**	Professeur à l'Université de Lille	Directeur
Mark **Herbster**	Lecturer at University College London	Rapporteur
Sophie **Tison**	Professeur à l'Université de Lille	Examinateur
Marc **Tommasi**	Professeur à l'Université de Lille	Examinateur

---

Contents

Chapter 1

Introduction

The present thesis has been prepared in partnership with the business software company SAP (CIFRE industrial agreement) in the innovation department dedicated to Business Intelligence (BI). The purpose of BI is to provide decision-makers with the tools and methods that allow for efficient exploitation of corporate data. To achieve this ultimate goal, BI systems have to deal with large amounts of data that come from multiple sources. For this reason, building efficient and robust algorithms often depends on our ability to process complex and heterogeneous relational structures.

In this context, I have carried out in parallel applied research (internal SAP projects) as well as theoretical research. In what follows, I present the outcome of this theoretical research which was focused on the problem of learning with complex inter-connected data.

The objective of Machine Learning techniques is to figure out how to automatically perform tasks by generalizing from examples. Machine Learning algorithms rely on existing data to build models that can be then applied to new situations. Over the past decades, the rise of digital sciences has led to an exponential growth in the volume of data (see for instance Gantz and Reinsel 2012), making it possible to tackle more and more applications.

Data instances are commonly described in terms of *features* and *inter-relations*. The features represent the individual properties of the instances while the inter-relations characterize the dependencies that exist between instances. As an illustration, let us consider the case of recommender systems whose purpose is to propose new movies to users based on historical data. The data contains two main types of instance: the users and the movies.

The movies are characterized by a set of features (title, style, duration, ...). The same goes for the users with a different set (name, age, occupation, ...). In addition of this information come the inter-relations that establish connections between individual instances (e.g.user U_1 is friend with user U_2, user U_3 has watched movie M_1 last week).

Many classical machine learning algorithms such as SVM (Vapnik 1998) or AdaBoost (Freund and Schapire 1995) focus on the specific case where the data is flat, i.e., where all the information lies into the individual features. However, in many recent applications, the role played by the inter-relations has become prominent. This is for example the case of social networks that have known a fast evolution since the early 2000s (Friendster, 2002; MySpace 2003; Facebook, Orkut, 2004). In this context, our knowledge about a given user depends crucially on its relationships with the other users (friendship relations, post interactions, ...). On the contrary, the information brought by the individual features (e.g., the user profile) is often regarded as unreliable and noisy since in many cases, it is not subject to any control.

Binary inter-relations and undirected graph model

The most common case of inter-relations are the binary relations between pairs of individuals. We can find these relations for instance in the social networks presented above (friendship relation, following relation, ...) A popular abstraction for this case lies into the notion of *undirected graph*. An undirected graph consists in a set of nodes N together with a set of edges E, where an edge is defined as an abstract connections between two nodes.

A lot of effort has been put in the past years into the theory of *graph* learning. We can distinguish between two main classes of graph learning problem. The first class focuses on how to improve the graph structure. It includes subproblems such as *Link Prediction* (find the edges that should be added to the graph) and *Entity Disambiguation* (improve the nodeset by, for instance, finding duplicates nodes). The second class of problems is the one on which we will concentrate in the following. It focuses on node and edge valuation and includes problems such as *node clustering* and *node classification* (both are discrete node valuation problems).

Note, that, in all the situations mentioned above, the valuation function has to somehow respect the configuration of the graph. We usually assume that the graph is an *homophilic structure*, i.e., that any two nodes linked by an edge should be regarded as similar and thus, should receive close valuations. This reasonable assumption has been inspired by empirical studies on real

life networks. Indeed, we can observe that binary inter-relations often occur between similar individuals. McPherson et al. (2001) provide an interesting discussion and give additional pointers to former studies on this topic.

In the present work, we will put our focus on the *continuous node valuation* problem where the objective is to find $f : N \to \mathbb{R}$. Note that, since N is finite, f can be regarded as a vector that belongs to $\mathbb{R}^{|N|}$. We consider a specific branch of graph learning called *spectral learning*. Spectral learning allows us to tackle the problem of the continuous node valuation by defining the notion of *graph Laplacian matrix*. The graph Laplacian matrix Δ is a symmetric positive semi-definite matrix in $\mathbb{R}^{|N|\times|N|}$ that encodes the structure of the graph. An interesting property of this matrix is that the term $f^T \Delta f$ is small as soon as the function f tends to respect the homophilic structure of the graph (i.e., as soon as it tends to assign close values the nodes that are connected by edges). Such a function is told to be *smooth* on the graph, and, for this reason, we denote the application $\Omega : f \to f^T \Delta f$ as the *smoothness measure* associated with the graph.

Since Δ is a symmetric real matrix, it is possible to consider an orthogonal basis $\mathbb{R}^{|N|}$ formed with unit eigenvectors of Δ. An interesting property of these vectors is that their smoothness is equal to their associated eigenvalue. For this reason, the analysis of the eigenvectors and the eigenvalues of Δ is of central importance in many situations, which legitimates the term of *spectral* learning (see also Chung 1997). An example of successful algorithm based on this idea is the popular *spectral clustering* algorithm (see Von Luxburg 2007) that tackles the problem of 2-class clustering (discrete node valuation) by considering an adequate relaxation of the objective function, thus leading to a continuous node valuation problem.

Complex relations and higher-order learning

A main limitation of the graph learning framework is that it is inherently limited to the case of binary relations. In the general case, inter-relations between data instances can take various forms and involve an arbitrary number of individuals. We call these general relations *higher-order* relations. As an illustration, let us consider the case of text documents that address different topics such as economics, health or environment. Note that a single document can address multiple topics (for instance health and environment). Belonging to a topic involves potentially more than two objects. Handling this type of relations is beyond the scope of the graph model and require specific tools.

Based on this observation, many people have started to consider generalized graph models with the goal of providing new learning strategies. One of

the most popular framework that allows to encode naturally higher-order relations is the *hypergraph* framework introduced by Berge (1989). Contrary to the graph case, edges in hypergraphs can link more than two nodes and are denoted as *hyperedges*. All the nodes that belong to the same hyperedge share some common property and, therefore, are expected to be similar. Hypergraphs are particularly suited to represent categorical relations such as the topic-based relation presented above.

A considerable amount of work has been conducted on hypergraph learning theory in the past decade (see for instance Agarwal et al. 2005; Tsuda 2005). In particular, several attempts have been made to generalize the notion of Laplacian matrix (Bolla 1993; Rodríguez 2003; Zhou et al. 2006). It can be noted, however, that all these proposals are implicitly based on graph reduction as shown in Agarwal et al. (2006) and, therefore, fail to capture the specificity of the corresponding higher-order relations. From a theoretical point of view, we expect a strict gain of expressiveness when we consider more complex relations, which is not the case here.

Another interesting case of general inter-relations are the *binary relations between sets*. Such relations occur for instance when we consider entities producing and consuming resources. Each type of resource defines a relation between two groups: the producers and the consumers. In this context, the goal can be to find an optimal way to split the entities in several autonomous groups or to identify the most important nodes that require to be monitored carefully. An other situation where binary relations between sets occurs is in multiplayer games: each game can be seen as a relation between several sets of players with an additional semantic associated to the game outcome (winner team, scores, ...) In this context, an interesting task is to estimate the skill level of each player given a set of games. This task, known as *skill rating*, is often associated to the *match-making* task that aims to propose new equilibrated games from historical data. A well-known example of algorithm that tackles these two problems is the Elo ranking system originally designed for chess competitions. The recent explosion of online gaming has again stressed the importance of developing efficient learning techniques that can work with such relations. .

In the present work, we propose a new abstract model called *hypernode graph* that allows to handle binary relations between sets. We define a hyperedge as an abstract connection between two sets of nodes called *hypernodes*. A hypernode graph $\mathbf{h} = (N, H)$ is simply a set of nodes N together with a set of hyperedges H. In a given hyperedge, each node is associated to a positive weight that represent its *contribution* to the relation. We consider only *balanced* relations in the sense where the total contributions of both hypernodes have to be equal. Formally, this leads to a technical constraint on the node weights called *equilibrium condition*. Note that as in the hypergraph

case, our structure is a direct generalization of undirected graphs. Indeed, a classic edge is fully equivalent to a hyperedge where both hypernodes are singletons. In this case, our equilibrium condition reduces to the existence of a single weight associated to the binary relation.

As in the graph case, we regard hypernode graphs as a homophilic structure: two hypernodes that are linked by a hyperedge should be somehow similar. Note that we can extend a real-valued node function $f : N \to \mathbb{R}$ to the set of hypernodes by considering the weighted sums of the values taken by f (the individual contributions are use for the weighting). Similarly to the graph case, we say that f is *smooth* on a hypernode graph $\mathbf{h}$ if it satisfies the homophilic condition, i.e., if it takes similar values on the hypernodes connected by a hyperedge.

In order to build efficient learning algorithms for hypernode graphs, we introduce a discrete analysis framework similar to the one presented in Zhou et al. (2005) for directed graphs. In particular, we define the notion of discrete gradient which allows us to quantify the smoothness of a given function $f : N \to \mathbb{R}$. On this basis, we introduce a general version of the graph Laplacian operator. The hypernode graph Laplacian matrix Δ is again a symmetric positive semi-definite matrix that belongs to $\mathbb{R}^{|N|\times|N|}$. Similarly to the graph case, we can define the smoothness measure $\Omega(f) = f^T \Delta f$. Note that, contrary to the hypergraph case described above, our extension provides a real gain of expressiveness. Indeed, we show that the smoothness measure Ω induced by a given hypernode graph cannot be in general reduced to a graph smoothness measure, even if we allow the usage of auxiliary nodes. This reason is that our model do not try to reduce the higher-order relations to the classic binary case but rather try to capture them as a whole.

As might be expected, many definitions and results coming from the graph theory won't hold in our generalized framework. In many cases, we have to replace them with new concepts and properties. In the present work, we review the most important changes and provide a detailed discussion on all these points. We strongly believe that the hypernode graph extension and its related spectral framework pave the way for new algorithmic applications.

As a proof of concept, we consider the skill rating problem described above and propose a new algorithm based on hypernode graphs. We model multiplayer games using hyperedges and add auxiliary nodes to express the dominance relations (winning team > loosing team) We show that our algorithm is able to outperform several dedicated algorithms for different game configurations.

Outline of the thesis

This dissertation is organized as follows:

- **Chapter 2** reviews essential background work and present the *hypernode graph* model. We start by introducing formally the notions of undirected graph and spectral learning. We define the notion of graph Laplacian matrix and discuss its main properties. Then, we present our model and propose an extension of the spectral learning framework. In particular, we introduce a generalized notion of Laplacian and show that it provides a strict generalization of the graph Laplacian in the sense that one cannot approximate it using a graph with a finite number of nodes. This last property allows us to claim the strict gain of expressiveness induced by our extension.

- **Chapter 3** studies in detail the main properties of hypernode graphs and reviews the main differences with the graph spectral framework. We start by pointing out a strong relation between hypernode graphs and a specific class of signed graphs, i.e., a specific class of graphs where the edges can be weighted with some negative values. We introduce the notion of reduced signed graph and discuss how our definition of the Laplacian matrix can be related to the notions of Laplacian matrix defined on signed graphs. In a second time, we discuss the notion of path and connectivity in hypernode graphs. In particular, we show that the notion of connected component can be replaced by two complementary notions and introduce the concept of strongly connected hypernode graph. Finally, we discuss the notion of distance in hypernode graphs and highlight the role of the kernel matrix which can be computed as the Moore-Penrose pseudo-inverse of the Laplacian matrix.

- **Chapter 4** presents an application of the hypernode graph framework to the problem of skill rating in multiple players games. We start by reviewing important background work for this specific issue. In particular, we will present two dedicated algorithms (Elo and Trueskill) and review their main properties. We then use hypernode graphs to model multiplayer games that involve two teams of distinct players and extend classic graph algorithms in order to learn the skill of each of the individual players. We use these learnt skills in order to predict the outcome of new games, which allows us to compare our methods with dedicated methods such as Elo and TrueSkill. We finally show experimentally that we are able to outperform these methods on various datasets.

- **Chapter 5** presents several research directions and discuss some in-

teresting open problems on hypernode graphs. We start by extending the notion of graph cut to hypernode graphs and review the main differences induced by this generalization. We believe especially that interesting research problems lies in the design of algorithms for hypernode graph MinCut. In a second part, we discuss the question of directivity in hypernode graphs. Indeed, as in the graph case, it makes sense in some applications to consider directed hyperedges. We discuss how this idea can be related to the theory of directed hypergraphs popularized by Gallo et al. (1993). Finally, we present some algebraical problems related to the hypernode graph extension. We describe in detail the different spaces corresponding to the sets of Laplacian matrix, Laplacian kernel, hypernode graph Laplacian and hypernode graph kernel and formulate on this basis a convex hull conjecture. We propose a geodesically complete Riemanian geometry for the set of strongly connected hypernode graphs, which paves the way for building new algorithms.

Notation	Description
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^+$	Set of nonnegative real numbers
$\mathbb{N}$	Set of natural numbers, i.e., $\{0, 1, \dots\}$
$\lvert\mathcal{S}\rvert$	Number of elements in the set $\mathcal{S}$
$\mathbf{x}, \mathbf{y}$	Arbitrary vectors
$\mathrm{Span}(\mathbf{x}_1, \dots, \mathbf{x}_p)$	Vector space spanned by the vectors $\mathbf{x}_1, \dots, \mathbf{x}_p$
$\mathbf{e}_i$	i-th basis vector
$\mathbf{1}$	Uniform vector $\begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix}^T$
$\mathbf{g}$	Arbitrary graph
$\tilde{\mathbf{g}}$	Arbitrary signed graph
$\mathbf{h}$	Arbitrary hypernode graph
N	Set of nodes (usually $\{1, \dots, \lvert N\rvert\}$)
E	Set of edges
H	Set of hyperedges
h	Arbitrary hyperedge
M	Arbitrary matrix (uppercase letter)
$M^\dagger$	Moore-Penrose pseudo-inverse of matrix M
M^T	Adjoint matrix corresponding to matrix M
$\mathrm{Rank}(M)$	Rank of matrix M
$\mathrm{Tr}(M)$	Trace of matrix M
$\mathrm{Null}(M)$	Nullspace of operator M
$M \succeq 0$	Positive semi-definiteness
$M \succ 0$	Positive definiteness
Id	Identity operator
f, g	Arbitrary functions
$f\big\vert_{\mathcal{S}}$	Function f restricted to the set $\mathcal{S}$
$\delta_{(i=j)}$	Kronecker delta (equals 1 if $i = j$, 0 otherwise)
$\delta_{(i\neq j)}$	Negative Kronecker delta (equals 1 if $i \neq j$, 0 otherwise)
$\mathrm{sgn}(\cdot)$	Sign operator ($\mathrm{sgn}(x) = 1$ if $x \geqslant 0$, -1 otherwise)
$\langle \cdot, \cdot \rangle$	Inner product between vectors
$\lVert \cdot \rVert$	Arbitrary norm
$\lVert \cdot \rVert_p$	L_p norm
$\mathbb{P}[\cdot]$	Probability of event
$\mathbb{E}[\cdot]$	Expectation of random variable

Summary of notation

Chapter 2

Hypernode graphs

Chapter abstract In this chapter, we introduce the notion of hypernode graph, which is our main contribution to the modeling of binary relations between sets of entities. Our main motivation is to define a model that goes beyond pairwise relations and allows to encode set based similarities. In the first part of the chapter, we review some important definitions concerning undirected graphs as the main model to represent pairwise relations between individuals. In particular, we recall the notion of Laplacian matrix which is the cornerstone of the spectral theory and discuss its main properties. In the second part, we introduce formally our model and discuss how we can extend the notion of Laplacian matrix to this new class of objects.
Finally, we discuss the expressiveness of our Laplacian framework and compare it with other extensions for higher order learning.

2.1 Undirected Graphs and Spectral Framework

In the following, we recall the commonly accepted definitions of undirected graphs, graph Laplacians and graph kernels.

2.1.1 Graphs and Laplacians

An *undirected graph* $\mathbf{g} = (N, E)$ is a set of nodes N together with a set of undirected edges E. We define $n = |N|$ and $p = |E|$. Each edge is an unordered pair $\{i, j\}$ of distinct nodes (no self-loops) and is associated with a non negative weight $w_{\{i,j\}}$. We define the neighborhood $\mathcal{N}(i)$ of a node i to be the set of all nodes j that are connected to i by an edge:

$$\mathcal{N}(i) = \{j \in N \text{ such that } \{i,j\} \in E\} \ .$$

The *adjacency matrix* W is the $n \times n$ matrix defined by

$$W_{i,j} = \begin{cases} w_{\{i,j\}} \text{ if } \{i,j\} \in E \\ 0 \text{ otherwise} \end{cases}$$

W is a symmetric matrix and has a null diagonal. The *degree* of a node $i \in N$ is defined as

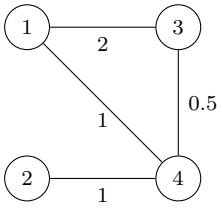
$$d(i) = \sum_{j \in \mathcal{N}(i)} w_{\{i,j\}} = \sum_{j \in N} W_{i,j} \ .$$

Note that the degree of a node is always positive, unless it does not participate in any edge (in this case, the degree is null). The *degree matrix* D is the $n \times n$ diagonal matrix defined by $D_{i,i} = d(i)$. We define the *volume* of the graph to be the sum of the degrees of its nodes. Namely

$$\text{Vol}(\mathbf{g}) = \sum_{i \in N} d(i) \ .$$

We define a *path* on a graph to be a finite sequence of nodes that are connected by a sequence of edges. A *connected component* is a maximal set of nodes in which any two nodes are connected to each other by at least one path.

Example 1. *Let* $\mathbf{g}_1$ *be the undirected graph defined by* $N = \{1,2,3,4\}$ *and* $E = \{\{1,3\},\{1,4\},\{2,4\},\{3,4\}\}$. *The edge weights are set to* $w_{\{1,3\}} = 2$, $w_{\{1,4\}} = 1$, $w_{\{2,4\}} = 1$ *and* $w_{\{3,4\}} = 0.5$. *We represent* $\mathbf{g}_1$ *in Figure 2.1 below, together with its adjacency matrix* W_1 *and its degree matrix* D_1. *The sequence of nodes* $(1,4,3)$ *is a path in the graph* $\mathbf{g}_1$ *but the set* $\{1,3,4\}$ *is not a connected component since we can add the node* 2 *without losing the connectivity property.*



$$W_1 = \begin{pmatrix} 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{pmatrix} \quad D_1 = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2.5 & 0 \\ 0 & 0 & 0 & 2.5 \end{pmatrix}$$

Figure 2.1: Undirected graph $\mathbf{g}_1$ and matrices W_1 and D_1

We now consider the problem of *collective node valuation*: for a given graph $\mathbf{g} = (N, E)$, the objective is to find a node valuation function $f : N \to \mathbb{R}$. As said in the previous chapter, we expect from f that it assigns close values to nodes that are linked by edges in the graph (homophilic assumption). When it happens, we say that f is *smooth* on $\mathbf{g}$. In order to quantify the *smoothness* of a function f over $\mathbf{g}$, let us first assume that each edge is associated with an arbitrary orientation. Then, we define the gradient function grad for f by, for every oriented edge (i, j),

$$\operatorname{grad} f : (i, j) \to \sqrt{w_{\{i,j\}}}(f(j) - f(i)) \ .$$

We can note that $|\operatorname{grad}(f)(i, j)|$ is small whenever $f(i)$ is close to $f(j)$. In the following, we will regard the euclidean space $\mathbb{R}^n$ (resp. $\mathbb{R}^p$) as the space of real-valued node functions (resp. real-valued edge functions). Since the gradient is a linear operator, we can denote by $G \in \mathbb{R}^{p \times n}$ the corresponding matrix. Then, the smoothness of a real-valued node function f over a graph $\mathbf{g}$ is defined by

$$\Omega(f) = \sum_{i,j \in V^2} |\operatorname{grad}(f)(i, j)|^2 = f^T G^T G f \ ,$$

where G is the matrix of the linear mapping grad from $\mathbb{R}^n$ into $\mathbb{R}^p$ where $n = |N|$ and $p = |E|$. In the case of the graph $\mathbf{g}_1$ presented in Figure 2.1, a possible gradient matrix is (for an arbitrary orientation of the edges)

$$G = \begin{pmatrix} -\sqrt{2} & 0 & \sqrt{2} & 0 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -\sqrt{0.5} & \sqrt{0.5} \end{pmatrix} \begin{matrix} \textit{Edge } 1 \to 3 \\ \textit{Edge } 1 \to 4 \\ \textit{Edge } 2 \to 4 \\ \textit{Edge } 3 \to 4 \end{matrix}$$

Note that Ω does not depend on the orientation of the edges. In the following, we will denote Ω as the *smoothness measure* associated with the graph $\mathbf{g}$. Moreover, we can show that

Proposition 1. *$\sqrt{\Omega}$ defines a semi-norm on $\mathbb{R}^n$, i.e., it is positive, homogeneous and sub-additive.*

Proof. $\sqrt{\Omega} = \|Gf\|_2$ is positive by construction. We can write for all $\alpha \in \mathbb{R}$ and $f \in \mathbb{R}^n$, $\sqrt{\Omega(\alpha f)} = \|\alpha G f\|_2 = |\alpha| \sqrt{\Omega(f)}$, which proves the homogeneity of $\sqrt{\Omega}$. Moreover, for all $f, g \in \mathbb{R}^n$,

$$\sqrt{\Omega(f + g)} = \|Gf + Gg\|_2 \leqslant \|Gf\|_2 + \|Gg\|_2 = \sqrt{\Omega(f)} + \sqrt{\Omega(g)} \ ,$$

which proves the sub-additivity. □

The symmetric matrix $\Delta = G^T G$ is called the *(unnormalized) graph Laplacian*, which is also proved to be defined by $\Delta = D - W$ where D and W are the degree matrix and the adjacency matrix of the graph. Note that the Laplacian operator is often introduced directly under the standard form $D-W$, without considering the underlying concept of graph gradient. However, gradients prove to be essential when it comes to extend the notion of Laplacian operator to the hypernode graphs.

Proposition 2. *Let* $\mathbf{g}$ *be an undirected graph and* Δ *its Laplacian matrix.* Δ *is symmetric and positive semi-definite. Moreover, we have*

$$\text{Null}(\Delta) = \text{Null}(G) = \text{Null}(\Omega) = \text{Span}(\mathbf{1}_{C_1}, \dots, \mathbf{1}_{C_p}) \ ,$$

where $\mathbf{1}_{C_1}, \dots, \mathbf{1}_{C_p}$ *are the indicator functions of the* k *connected components of* $\mathbf{g}$ *and* $\text{Null}(\Delta)$ *denotes the nullspace of* Δ*. As a consequence, the dimension of* $\text{Null}(\Delta)$ *is equal to the number of connected components.*

Proof. The symmetricity and the positive semi-definiteness of Δ comes directly from the fact that $\Delta = G^T G$. If f is in $\text{Null}(\Delta)$, we have $f^T \Delta f = 0 = \|Gf\|_2^2$ so $f \in \text{Null}(G)$. Conversely if $Gf = 0$, then $\Delta f = G^T G f = 0$. Moreover, $Gf = 0$ if and only if $\|Gf\|_2^2 = 0 = \Omega(f)$, which concludes the proof of the equality between the three nullspaces.

Let us now consider $f \in \text{Null}(G)$. We have, for every edge (i, j),

$$\text{grad}(f)(i,j) = \sqrt{w_{i,j}}(f(j) - f(i)) = 0 \ .$$

As a consequence, as soon as two nodes i and j are linked by an edge, we must have $f(i) = f(j)$. By transitivity, all the nodes that belong to the same connected component should have the same value. In the following, we will denote by α_t the value taken by f on the connected component C_t. Since by definition the connected components have null intersections, we can write

$$f = \sum_{t=1}^{p} \alpha_t \mathbf{1}_{C_t} \ ,$$

so $f \in \text{Span}(\mathbf{1}_{C_1}, \dots, \mathbf{1}_{C_p})$. The reverse is straightforward: let us choose f in $\text{Span}(\mathbf{1}_{C_1}, \dots, \mathbf{1}_{C_p})$. For any edge $\{i, j\} \in E$, i and j have to belong to the same connected component. Since $f(i) = f(j)$, $\text{grad}(f)(i, j) = 0$ and $\Delta f = \|Gf\|_2^2 = 0$. □

A direct consequence of Proposition 2 is that $\text{Null}(\Delta)$ is never reduced to $\{0\}$ since it always contains the constant vector $\mathbf{1}$. Since Δ is symmetric and positive semi-definite, it is orthogonally diagonalizable and all its eigenvalues are positive or null. In the following, we assume that the $q \leqslant n$ eigenvalues are ordered as follow

$$0 = \lambda_1 < \lambda_2 < \cdots < \lambda_q$$

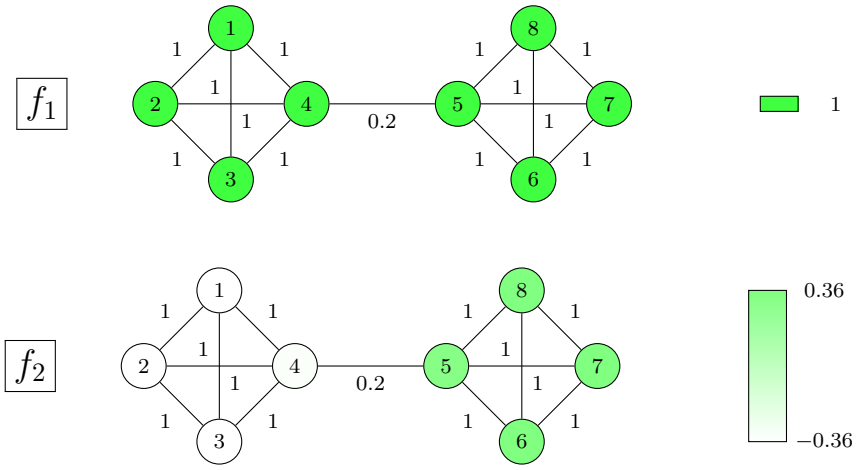


Figure 2.2: Graph $\mathbf{g}_2$ colored using the modes f_1 and f_2

Note that if f is a unit eigenvector associated with the eigenvalue λ, then $\Omega(f) = f^T \lambda f = \lambda$. The unit eigenvectors of Δ can be seen as the *harmonic modes* of the graph and their smoothness depend on their related eigenvalue. A mode associated with the eigenvalue $\lambda_1 = 0$ is a *fundamental mode*. Because of Proposition 2, any fundamental mode is a linear combination of the indicator functions of the connected components.

Example 2. *Let us consider the graph* $\mathbf{g}_2$ *with* $N = \{1, 2, 3, 4, 5, 6, 7, 8\}$ *presented in Figure 2.2 below.*

The Laplacian Δ_2 *of this graph has four distinct eigenvalues* $\lambda_1 = 0$, $\lambda_2 = 0.092$, $\lambda_3 = 4$ *and* $\lambda_4 = 4.3$. λ_1 *is associated to the fundamental mode* $f_1 = 1_N$. λ_2 *is associated to a unique mode* f_2 *(*$\Omega(f_2) = \lambda_2 = 0.092$*). Since* $\lambda_4 > \lambda_3 >> \lambda_2$ *so the modes associated to* λ_3 *and* λ_4 *are far less smooth than* f_2. *We present in Figure 2.2 below the modes* f_1 *and* f_2. *Note that these modes can be used to feed a clustering algorithm on the nodes. This idea leads to the so-called spectral clustering algorithm (see Von Luxburg 2007).*

Note that some papers also consider a different version of the Laplacian matrix called normalized Laplacian that is defined by $\Delta_n = I - D^{-1/2} W D^{-1/2}$. As shown in (Zhou et al. 2005), this Laplacian can be obtained by considering the normalized gradient ngrad defined by

$$\text{ngrad}\, f : (i, j) \to \sqrt{w_{\{i,j\}}} \left(\frac{f(j)}{\sqrt{d(j)}} - \frac{f(i)}{\sqrt{d(i)}} \right) .$$

The only difference with the gradient operator presented above is that the entries of f are divided by the degrees of the nodes. The related gradient

matrix is $G_n = GD^{-1/2}$, which lead to

$$\Delta_n = G_n^T G_n = D^{-1/2}\Delta D^{-1/2} = I - D^{-1/2}WD^{-1/2} \ .$$

Note that the strong link between the number of connected components and the nullspace is preserved in Δ_n as stated in the following Proposition.

Proposition 3. *Let* $\mathbf{g}$ *be an undirected graph and* Δ_n *its normalized Laplacian matrix.* Δ_n *is symmetric and positive semi-definite. We have*

$$\mathrm{Null}(\Delta_n) = \mathrm{Null}(G_n) = \mathrm{Span}(D^{1/2}\mathbf{1}_{C_1}, \ldots, D^{1/2}\mathbf{1}_{C_p})$$

where $\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_p}$ *are the indicator functions of the* p *connected components of* $\mathbf{g}$*. As a consequence, the dimension of* $\mathrm{Null}(\Delta_n)$ *is the number of connected components.*

Proof. The proof is very similar to the proof of Proposition 2. The symmetricity and the positive semi-definiteness of Δ_n comes directly from the fact that $\Delta_n = G_n^T G_n$. For every f in $\mathrm{Null}(\Delta_n)$, we have $f^T\Delta_n f = 0 = \|G_n f\|^2$ so $f \in \mathrm{Null}(G_n)$. Conversely, if $G_n f = 0$, $\Delta f = G_n^T G_n f = 0$, which proves the equality $\mathrm{Null}(\Delta_n) = \mathrm{Null}(G_n)$.

Let us now consider $f \in \mathrm{Null}(G_n)$. We have, for every edge (i,j),

$$\mathrm{ngrad}(f)(i,j) = \sqrt{w_{\{i,j\}}}\left(\frac{f(j)}{\sqrt{d(j)}} - \frac{f(i)}{\sqrt{d(i)}}\right) = 0 \ .$$

As a consequence, as soon as two nodes i and j are linked by an edge, we must have $f(i)/\sqrt{d(i)} = f(j)/\sqrt{d(j)}$. By transitivity, all the nodes i that belong to the same connected component $C \subset N$ should share the same ratio $f(i)/\sqrt{d(i)}$. Said differently,

$$\exists \alpha \in \mathbb{R} \text{ s.t. } \forall i \in C, f(i) = \alpha\sqrt{d(i)} = (\alpha D^{1/2}\mathbf{1}_C)(i) \ ,$$

and, consequently, $f \in \mathrm{Span}(D^{1/2}\mathbf{1}_{C_1}, \ldots, D^{1/2}\mathbf{1}_{C_p})$.

Let us now f in $\mathrm{Span}(D^{1/2}\mathbf{1}_{C_1}, \ldots, D^{1/2}\mathbf{1}_{C_p})$. We can write

$$f = \sum_{t=1}^{p} \alpha_t D^{1/2}\mathbf{1}_{C_t} \ .$$

For each edge (i,j), we can find $t \in [1,p]$ such that i and j belong to C_t. We have $f(i) = \alpha_t\sqrt{d(i)}$ and $f(j) = \alpha_t\sqrt{d(j)}$, which leads to $\mathrm{ngrad}(f)(i,j) = 0$. Consequently, $f \in \mathrm{Null}(G_n)$, which concludes the proof. □

2.1.2 Graph kernels and distances

In this section, we review the notion of graph distance and introduce some important notations for the next chapters. A graph distance is a function $d : N \times N \to \mathbb{R}$ that defines a metric on the nodeset N. Namely, d has to satisfy the following axioms:

1. Positivity: $\forall (i,j) \in N^2, \quad d(i,j) \geqslant 0$
2. Symmetricity: $\forall (i,j) \in N^2, \quad d(i,j) = d(j,i)$
3. Coincidence: $\forall (i,j) \in N^2, \quad d(i,j) = 0 \Leftrightarrow i = j$
4. Triangle Inequality: $\forall (i,j,k) \in N^3, \quad d(i,j) \leqslant d(i,k) + d(k,j)$

A standard graph distance is the *shortest-path distance* and is defined for any pair of nodes (i,j) as the minimum sum of weights along a path joining i and j. For example, let us consider graph $\mathbf{g}_3$ presented in Figure 2.3. The closest nodes to 4 in the sense of the shortest-path distance are 1, 2, 3 and 5 (the distance is equal to 1 for these four nodes). The shortest-path distance can be computed using algorithms such as Djikstra's algorithm and serves as a basis for many applications.

A main limitation of this approach is that it does not take into account the global connectivity of the graph. In the case of $\mathbf{g}_3$, the shortest-path distance does not reflect the existence of the two clusters $\{1,2,3,4\}$ and $\{5,6,7,8\}$. In the following, we will introduce a family of graph distances based on the Laplacian matrix Δ that solves this issue. In particular, we will see that one of these distances is strongly related to the notion of random walk on graphs.

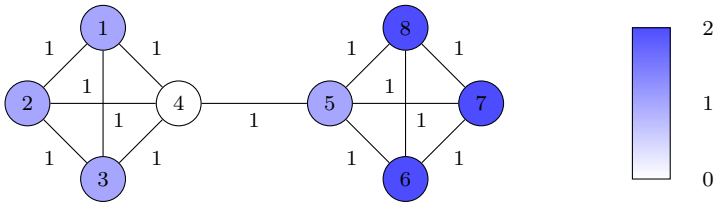


Figure 2.3: Graph $\mathbf{g}_3$ and shortest-path distribution around the node 4

Let us consider in this section a graph $\mathbf{g}$ and its Laplacian Δ. Since Δ is symmetric and positive semi-definite (Proposition 2), its Moore-Penrose pseudo-inverse $\Delta^\dagger$ (see Penrose 1955) is also symmetric and positive semi-definite. This matrix is called the *graph kernel* related to the graph Laplacian Δ. Let us define d^α for $\alpha \geqslant 1$ and for any pair of nodes (i,j)

$$d^\alpha(i,j) = \left(\sqrt{\Delta^\dagger_{i,i} + \Delta^\dagger_{j,j} - 2\Delta^\dagger_{i,j}} \right)^\alpha \quad . \tag{2.1}$$

Proposition 4. *d^1 defines a pseudometric on $\mathbf{g}$, i.e., it is positive, symmetric and satisfies the triangle equality. Moreover, when $\mathbf{g}$ is connected, d^1 becomes a metric (it satisfies in addition the coincidence axiom $d(i,j) = 0 \Leftrightarrow i = j$). When $\mathbf{g}$ is connected, d^2 is also a metric.*

Proof. By definition, we have $d^1(i,j) = \sqrt{(\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_i - \mathbf{e}_j)}$ where $\mathbf{e}_i$ and $\mathbf{e}_j$ are respectively the i-th and j-th vectors of the standard basis. d^1 is positive, symmetric and satisfies the triangle equality since the inner product defined by

$$\langle x, y \rangle_{\Delta^\dagger} = x^T \Delta^\dagger y \ \in \mathbb{R}$$

is positive semi-definite and satisfies the Cauchy-Schwarz inequality for any pair of vectors (x, y). We conclude from these properties that d^1 is a pseudometric on $\mathbf{g}$. d^1 is not a metric in general since we can have $d^1(i,j) = 0$ for $i \neq j$. As a simple example, consider a graph with two disconnected nodes $N = \{1, 2\}$. We have $\Delta = \Delta^\dagger = 0$ and $d^1(1,2) = 0$. Note that the reverse property $\forall i \in N, \quad d^1(i,i) = 0$ always holds.

Let us now show that d^1 is a metric if the graph $\mathbf{g}$ is connected. For this purpose, we consider $i, j \in N$ such that $d^1(i,j) = 0$ and we show that $i = j$. We first write $\Delta^\dagger = V \Lambda^\dagger V^T$ where Λ is a diagonal matrix containing the eigenvalues $\lambda_1, \ldots, \lambda_n$ of Δ and V is an orthogonal matrix. Since the graph is connected, we know that $\dim(\mathrm{Null}(\Delta)) = \dim(\mathrm{Span}(\mathbf{1})) = 1$ (see Proposition 2) and, therefore, we can assume without loss of generality that $\lambda_1 = 0 < \lambda_2 \leqslant \cdots \leqslant \lambda_n$. By definition of d^1, we have

$$\begin{aligned} d^1(i,j) &= \sqrt{\sum_k \lambda_k^\dagger \left(V_{i,k}^2 + V_{j,k}^2 - 2 V_{i,k} V_{j,k} \right)} \\ &= \sqrt{\sum_k \lambda_k^\dagger (V_{i,k} - V_{j,k})^2} \end{aligned}$$

Since $d^1(i,j) = 0$, we have $V_{i,k} = V_{j,k}$ for every $k \geqslant 2$. Moreover, since the graph is connected, the column vector $V_{\cdot,1}$ associated to $\lambda_1 = 0$ is in $\mathrm{Span}(\mathbf{1})$ (see Proposition 2). Consequently, we also have $V_{i,1} = V_{j,1}$ and the line vectors $V_{i,\cdot}$ and $V_{j,\cdot}$ are equals, which is only possible when $i = j$ since V is an orthogonal matrix ($V_{i,\cdot}^T V_{j,\cdot} = \delta_{(i=j)}$). This shows the coincidence axiom for d^1 and show that it defines an actual metric as soon as $\mathbf{g}$ is connected.

Since $d^2(i,j) = \left(d^1(i,j)\right)^2$, d^2 is also positive, symmetric and satisfies the coincidence axiom when $\mathbf{g}$ is connected. It remains to show that it satisfies the triangle inequality, i.e., we have to prove that for all $(i,j,k) \in N^3$

$$d^2(i,j) \leqslant d^2(i,k) + d^2(k,j) \ .$$

Klein and Randić (1993) gives a sketch proof of this property using an electrical equivalence (all edges are replaced by 1-Ohm resistors). In the following, we give a complete algebraic proof of this result, which will be useful in the next chapters when we will consider the situation of hypernode graphs. Let us first remark that for all $(i,j,k) \in N^3$

$$\begin{aligned} d^2(i,j) &= (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_i - \mathbf{e}_j) \\ &= (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_i - \mathbf{e}_k) + (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_k - \mathbf{e}_j) \end{aligned} \tag{2.2}$$

Consequently, it is sufficient to prove that

$$\forall (i,j,k,\ell) \in N^4, \quad (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_k - \mathbf{e}_\ell) \leqslant (\mathbf{e}_k - \mathbf{e}_\ell)^T \Delta^\dagger (\mathbf{e}_k - \mathbf{e}_\ell) \tag{2.3}$$

Indeed, by combining this last inequality with Equation (2.2), we will get the desired result:

$$\begin{aligned} d^2(i,j) &= (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_i - \mathbf{e}_k) + (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_k - \mathbf{e}_j) \\ &\leqslant (\mathbf{e}_i - \mathbf{e}_k)^T \Delta^\dagger (\mathbf{e}_i - \mathbf{e}_k) + (\mathbf{e}_k - \mathbf{e}_j)^T \Delta^\dagger (\mathbf{e}_k - \mathbf{e}_j) \\ &\leqslant d^2(i,k) + d^2(k,j) \end{aligned}$$

In the following, we will assume that $k \neq \ell$ (otherwise, Equation (2.3) reduces to $0 \leqslant 0$). Let us define $y = \Delta^\dagger(\mathbf{e}_k - \mathbf{e}_\ell)$. Our goal is to prove that $y_i - y_j \leqslant y_k - y_\ell$, or equivalently that $\min_s y_s = y_\ell$ and $\max_s y_s = y_k$. Since $\mathbf{g}$ is connected, we have $\mathbf{e}_k - \mathbf{e}_\ell \in \mathrm{Null}(\Delta)^\perp = \mathrm{Span}(\mathbf{1})^\perp$. We know from the general properties of the Moore-Penrose pseudo-inverse (see for example Penrose 1955) that $\Delta\Delta^\dagger$ is the orthogonal projector on $\mathrm{Null}(\Delta)^\perp$. Thus, it follows that

$$\Delta y = \mathbf{e}_k - \mathbf{e}_\ell \tag{2.4}$$

Equation (2.4) can be seen as a diffusion equation where the input function is equal to $\mathbf{e}_k - \mathbf{e}_\ell$ (ℓ corresponds to a *sink* node and k to a *source* node) and where y plays the role of the corresponding potential function. Our goal is thus to prove that the maximal difference of potential should occur between the source node k and the sink node ℓ. We illustrate this idea in Figure 2.4.

We now assume that $y_\ell \neq \min_s y_s$ and show that it leads to an absurdity. First of all, let us notice that y cannot be uniform on N since $\Delta y \neq 0$. Consequently and since the graph is connected, we can find two nodes u and v that are connected by an edge and such that $\min_s y_s = y_u < y_v$. Since $y_\ell \neq \min_s y_s$, we have $\ell \neq u$ and we get from Equation (2.4)

$$(\Delta y)_u = \delta_{(u=k)} - \delta_{(u=\ell)} = \delta_{(u=k)} \geqslant 0 \ .$$

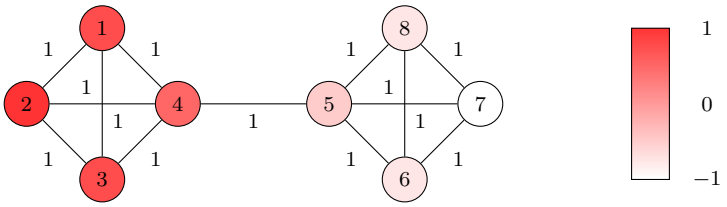


Figure 2.4: Graph $\mathbf{g}_3$ and corresponding potential function y obtained for $k = 2$ and $\ell = 7$. The maximum difference of potential is $y_2 - y_7 = 2$

We now expand the left side of this inequality to get

$$(\Delta y)_u = d(u)y_u - \sum_{s \in \mathcal{N}(u)} w_{\{s,u\}} y_s \geqslant 0 \quad ,$$

which leads to

$$y_u \geqslant \sum_{s \in \mathcal{N}(u)} \frac{w_{\{s,u\}}}{d(u)} y_s \quad . \tag{2.5}$$

We now have two cases to consider:

(∗) Either y is constant on $\mathcal{N}(u)$

(∗∗) Or y takes multiple values on $\mathcal{N}(u)$

Let us first consider (∗). Since $v \in \mathcal{N}(u)$ and $d(u) = \sum_{s \in \mathcal{N}(u)} w_{\{s,u\}}$, Equation (2.5) reduces to

$$y_u \geqslant y_v \quad ,$$

which is inconsistent with our hypothesis $y_v < y_u$. Consequently, the only solution is to have (∗∗), which lead to rewrite Equation (2.5),

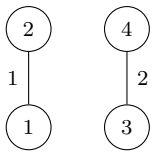
$$y_u \geqslant \sum_{s \in \mathcal{N}(u)} \frac{w_{\{s,u\}}}{d(u)} y_s > \min_{s \in \mathcal{N}(u)} y_s \sum_{s \in \mathcal{N}(u)} \frac{w_{\{s,u\}}}{d(u)} = \min_{s \in \mathcal{N}(u)} y_s \quad .$$

This last inequality is in contradiction with $y_u = \min_s y_s$ so the second case (∗∗) is impossible as well. As a consequence, we have necessarily $y_\ell = \min_s y_s$. We get similarly the second property $\max_s y_s = y_k$, which concludes the proof. □

Note that the fact that d^2 is a metric is a very specific property of connected graphs. Indeed, the square of a metric (resp. a pseudometric) is not a metric (resp. a pseudometric) in general since it does not satisfy the triangle inequality. Let us consider for example the very simple disconnected graph $\mathbf{g}_4$ depicted in Figure 2.5. We present on the same figure the distance matrix

$\mathcal{D}^2$ defined by $(\mathcal{D}^2)_{i,j} = d^2(i,j)$. $\mathbf{g}_4$ has two connected components and we can notice that d^2 does not satisfy the triangle equality since

$$d^2(1,2) = 1 > d^2(1,3) + d^2(3,2) = 0.75 \ .$$



$$\mathcal{D}^2 = \begin{pmatrix} 0 & 1 & 0.375 & 0.375 \\ 1 & 0 & 0.375 & 0.375 \\ 0.375 & 0.375 & 0 & 0.5 \\ 0.375 & 0.375 & 0.5 & 0 \end{pmatrix}$$

Figure 2.5: Graph $\mathbf{g}_4$ and distance matrix $\mathcal{D}^2$ associated with d^2

The distance d^1 serves as a basis for many popular algorithms. As an illustration, we can consider the case of *spectral clustering* (see Von Luxburg 2007) where the main idea is to map the graph nodes to an euclidean latent space where the distance corresponds to d^1. As announced, a desirable property of the family d^α that makes it particularly attractive for learning algorithms is that it allows to take into account the global connectivity of the graph. We illustrate this fact by considering again the graph $\mathbf{g}_3$ presented in Figure 2.3 above. We have $d^2(4,2) = 0.5 < d^2(4,5) = 1$ and the closest nodes to 4 are now 1, 2 and 3, which reflects the existing clusters $\{1,2,3,4\}$ and $\{5,6,7,8\}$. We present in Figure 2.6 the distribution of d^2 on $\mathbf{g}_3$.

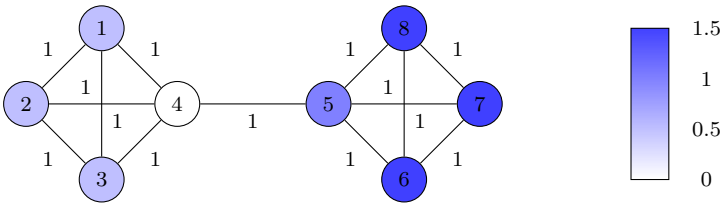


Figure 2.6: Graph $\mathbf{g}_3$ colored using the distribution of d^2 around the node 4

Note that the semantic of d^2 is closely related with the notion of *random walk* on graphs. First, recall that *random walk* on $\mathbf{g}$ can be defined as a Markov chain $(X_n)_{n\geqslant 0}$ whose state space is the set of nodes N and whose probability transition is equal to

$$P_{i\to j} = \frac{W_{i,j}}{d(i)} \ .$$

Intuitively, two nodes should be regarded as distant if it takes time for a random walker to travel from the first one to the second one. For example, in the global webgraph, this corresponds to the average number of (undirected) hyperlinks that a random surfer should follow to travel between two pages.

Let us consider the random walk $(X_n^i)_{n\geqslant 0}$ defined by $X_0^i = i$ for some node $i \in N$. The first passage time to a node j is the random variable $T_j(X^i)$ whose value is the first time n that $X_n^i = j$ (or ∞ if there is no such n). We define the *hitting time* $H : N \times N \to \mathbb{R}$ by

$$H(i,j) = \mathbb{E}\left[T_j(X^i)\right] \quad .$$

Note that $H(i,j)$ is not symmetrical, which is a required property for a metric. In order to solve this issue, we focus on the symmetrized quantity

$$d^{\mathrm{ctd}}(i,j) = H(i,j) + H(j,i) \quad .$$

$d^{\mathrm{ctd}}(i,j)$ is called *commute-time distance* between the nodes i and j and corresponds to the time taken by a random walker to go from i to j and come back. Note that, when $\mathbf{g}$ has several connected components, the hitting-time between two nodes i and j can be infinite. For this reason, we usually do not consider d^{ctd} on disconnected graphs. At the contrary, when the graph is connected, we have the following result:

Proposition 5. *If* $\mathbf{g}$ *is connected,* d^{ctd} *is a metric and* $d^{\mathrm{ctd}} = \mathrm{Vol}(\mathbf{g})d^2$.

Proof. A proof of this property based on a electrical equivalence can be found in Chandra et al. (1996). In the next chapter, we will present a more general proof for the case of hypernode graphs. □

2.2 Hypernode graphs

In this section, we introduce our model for representing binary relations over sets. As discussed in the previous chapter, our goal is to encode pairwise similarity relations between sets of nodes. We start by defining formally the notion of hypernode graph and present some important notations. Then, we introduce our extension of the graph spectral framework (see Section 2.1) and discussed its main properties.

2.2.1 Model definition

Definition 1. *A* hypernode graph $\mathbf{h} = (N, H)$ *is a set of nodes* N *together with a set of hyperedges* H*. Each hyperedge* $h \in H$ *is an unordered pair* $\{s_h, t_h\}$ *of two non empty and disjoint hypernodes (a hypernode is a subset of* N*). Each hyperedge* $h \in H$ *has a weight function* w_h *mapping every node* i *in* $s_h \cup t_h$ *to a positive real number* $w_h(i)$ *(for* $i \notin s_h \cup t_h$*, we define* $w_h(i) = 0$*). Each weight function* w_h *of* $h = \{s_h, t_h\}$ *must satisfy the* Equilibrium Condition *defined by*

$$\sum_{i \in t_h} \sqrt{w_h(i)} = \sum_{i \in s_h} \sqrt{w_h(i)} \ . \qquad \text{(Equilibrium Condition)}$$

We will say that a node i *belongs to* a hyperedge h ($i \in h$) if $w_h(i) \neq 0$. In all that follows, we denote by n the number of edges $|N|$ and by p the number of hyperedges $|H|$. We define the degree of a node i by

$$d(i) = \sum_h w_h(i) \geqslant 0 \ . \tag{2.6}$$

The degree of a node is positive when it participates in at least one hyperedge. We define the diagonal degree matrix by $D = \mathrm{diag}(d(1), \ldots, d(n))$ and the volume of the hypernode graph by $\mathrm{Vol}(\mathbf{h}) = \sum_{i \in N} d(i) = \mathrm{Tr}(D)$.

Example 3. *An example of hypernode graph is given in Figure 2.7 with the hypernode graph* $\mathbf{h}_5$. $\mathbf{h}_5$ *has four nodes denoted by* $1, 2, 3, 4$ *and two hyperedges* $h_1 = \{\{1,3\},\{2,4\}\}$ *and* $h_2 = \{\{1,4\},\{2,3\}\}$. *All the weights are set to* 1, *which satisfy the Equilibrium Condition. The corresponding degree matrix is*

$$D_5 = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

and we have $\mathrm{Vol}(\mathbf{h}_5) = \mathrm{Tr}(D_5) = 8$.

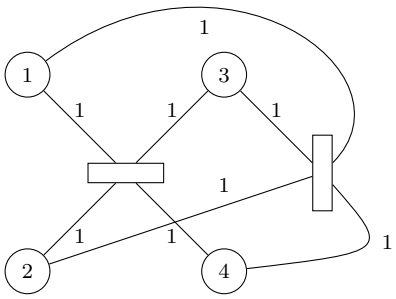


Figure 2.7: Hypernode graph $\mathbf{h}_5$ depicted with the following formalism: each hyperedge $h = \{s_h, t_h\}$ is represented by a rectangle, each of whose long sides is associated to a hypernode (s_h or t_h). All the nodes in h are connected to their corresponding long side.

When a hyperedge h is an unordered pair $\{\{i\},\{j\}\}$ of two nodes i, j, the Equilibrium Condition states that the weights $w_h(i)$ and $w_h(j)$ are equals. Therefore, every hypernode graph such that all hyperedges are unordered

pairs of singleton nodes can be viewed as a graph, and we will say that the hypernode graph is a graph. In this case, the adjacency matrix W of the (equivalent) graph is defined by $W_{i,j} = W_{j,i} = w_h(i) = w_h(j)$ for every hyperedge $\{\{i\}, \{j\}\}$, and 0 otherwise. Note that the hypernode graph model allows for multiple hyperedges with the same hypernodes. In this case we have to sum over all the hyperedges $\{\{i\}, \{j\}\}$ in order to compute the pairwise weight $W_{i,j}$ in the corresponding graph. Conversely, a graph can always be viewed as an hypernode graph by regarding every edge as a hyperedge made up of two singletons.

Example 4. *Let us consider the hypernode graph* $\mathbf{h}_6$ *presented in Figure 2.8. This hypernode graph corresponds to the undirected graph* $\mathbf{g}_1$ *presented in Section 2.1.1. Indeed, every hyperdedge is an ordered pair of two nodes and we have* $W_{1,2} = 2, W_{1,4} = 1, W_{2,4} = 1$ *and* $W_{3,4} = \frac{1}{8} + \frac{3}{8} = \frac{1}{2}$.

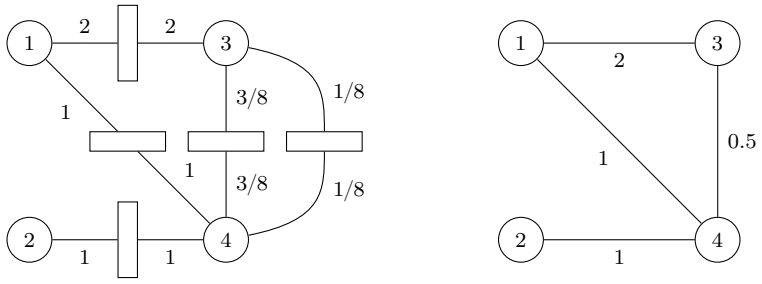


Figure 2.8: (*left*) Hypernode graph $\mathbf{h}_6$ and (*right*) corresponding graph $\mathbf{g}_1$

In some cases, we will need to define an arbitrary orientation of the hyperedges. For this, we define an *orientation function* to be a mapping from $\epsilon : H \times N$ to $\{-1, 0, 1\}$ such that satisfies the following conditions:

1. $\epsilon(h, i) = 0$ if and only if i does not belong to h
2. If i and j are in the same hypernode of h then $\epsilon(h, i) = \epsilon(h, j)$
3. If i and j are in different hypernodes, then $\epsilon(h, i) = -\epsilon(h, j)$

For $h \in H$ and an orientation function ϵ, the hypernode described by $\{i \in N \text{ s.t. } \epsilon(h, i) = -1\}$ is called *tail* of h, while the hypernode described by $\{i \in N \text{ s.t. } \epsilon(h, i) = 1\}$ is called *head* of h.

Example 5. *Let us consider the hypergraph* $\mathbf{h}_7$ *with* $N = \{1, 2, 3, 4\}$ *and a single hyperedge* $h = \{\{1, 2\}, \{3, 4\}\}$ *(all weights are set to 1). A valid orientation function for* $\mathbf{h}_7$ *is given by* $\epsilon(h, 1) = \epsilon(h, 2) = -1$ *and* $\epsilon(h, 3) = \epsilon(h, 4) = 1$. *With this orientation, hypernode* $\{1, 2\}$ *will be the tail of* h *and the hypernode* $\{3, 4\}$ *will be the head. The pair* $(\mathbf{h}_7, \epsilon)$ *defines a directed hypernode graph. We present* $\mathbf{h}_7$ *and* ϵ *on Figure 2.9 below.*

Note that the quantity $\epsilon(h, i)\epsilon(h, j)$ only depends on whether the two nodes

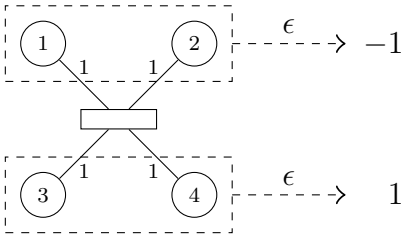


Figure 2.9: Hypergraph $\mathbf{h}_7$ and orientation function ϵ

belong or not to the same hypernode in h. Indeed, if we define the mapping $P : H \times N \times N \to \{-1, 1, 0\}$ by

$$P(h,i,j) = \begin{cases} 1 \text{ if } i \text{ and } j \text{ belong to different hypernodes of } h \ , \\ -1 \text{ if } i \text{ and } j \text{ belong to the same hypernode of } h \ , \\ 0 \text{ if } i \text{ or } j \text{ does not belong to } h \ , \end{cases} \quad (2.7)$$

then for any orientation function ϵ, we have

$$P(h,i,j) = -\epsilon_h(i)\epsilon_h(j) \ .$$

The mapping $P(\cdot,\cdot,\cdot)$ can be interpreted as a type of electrical polarity which depends on whether or not the two nodes belong to the same hypernode in a given hyperedge h.

2.2.2 Hypernode graph Laplacians

The graph Laplacian allows to define a smoothness semi-norm (see Section 2.1) which models the similarity between connected nodes. Indeed, when assigning labels or scores to the nodes of a graph using a real-valued function f, the smoothness operator allows to ensure that $f(i)$ is close to $f(j)$ when i and j are connected. Note that the higher the edge weight connecting i and j, the more important the closeness constraint. We extend this notion of similarity to hypernode graphs by considering the weighted sums of the values of f over all nodes in every end of the hyperedges. Namely, for $f : N \to \mathbb{R}$ and $h = \{s_h, t_h\} \in H$, we define f over the ends of h:

$$f(s_h) = \sum_{i \in s_h} f(i)\sqrt{w_h(i)}; \ f(t_h) = \sum_{i \in t_h} f(i)\sqrt{w_h(i)} \ .$$

We will say that f is smooth on $\mathbf{h}$ if for every hyperedge $h \in H$, $f(s_h)$ is close to $f(t_h)$. With this definition, we model the similarity between

connected node sets by considering the total value of f on these sets. It should be noted that when f is a constant function over N, then $f(t_h)$ is equal to $f(s_h)$ because of the Equilibrium Condition. Let us consider an example.

Example 6. *(Example 3 continued)*

Let us consider the hypernode graph $\mathbf{h}_5$ *together with a real-valued node function* f*. The smoothness of* f *on* $\mathbf{h}_5$ *will express that* $f(1) + f(3)$ *should be close to* $f(2) + f(4)$ *and* $f(1) + f(4)$ *should be close to* $f(2) + f(3)$*.*

Note that if $h = \{\{i\}, \{j\}\}$, then $f(\{i\}) = f(i)\sqrt{w_h(i)}$ and $f(\{j\}) = f(j)\sqrt{w_h(j)}$. Since in this case $w_h(i) = w_h(j)$, the closeness constraint reduces to "$f(i)$ is close to $f(j)$", which is fully consistent with the graph case.

In order to quantify the smoothness of a node function f, we introduce a gradient function similarly to the graph case described in Section 2.1.1.

Definition 2. *Let* $\mathbf{h} = (N, H)$ *be a hypernode graph and* ϵ *an orientation function. The gradient is a linear application that maps any real-valued node function* $f : N \to \mathbb{R}$ *into a real-valued hyperedge function* $(\operatorname{grad} f) : H \to \mathbb{R}$*. It is defined for all* h *by*

$$\operatorname{grad} f : h \to f(t_h) - f(s_h) = \sum_{i \in H} \epsilon(h, i)\sqrt{w_h(i)} f(i) \quad , \tag{2.8}$$

where t_h *(resp.* s_h*) is the* head *of* h *(resp. the* tail *of* h*).*

Note that the quantity $|\operatorname{grad}(f)(h)|^2$ does not depend on the choice of ϵ and, as expected, is small when $f(s_h)$ is close to $f(t_h)$. If $\mathbf{h}$ is an undirected hypernode graph, then the gradient is defined up to an arbitrary orientation of the hyperedges. Similarly to Section 2.1, we regard the euclidean space $\mathbb{R}^n$ (resp. $\mathbb{R}^p$) as the space of real-valued node functions (resp. real-valued hyperedge functions). We denote by $G \in \mathbb{R}^{p \times n}$ the matrix of the gradient mapping. We have for every hyperedge h and every node i,

$$G_{h,i} = \epsilon(h, i)\sqrt{w_h(i)}$$

We present examples of gradient matrices in Example 7 and we discuss the relations between graph gradients and hypernode graph gradients. Recall that a hypernode graph in which every hyperedge is a pair of singleton nodes is a graph and conversely. It is easy to show that, for every graph $\mathbf{g}$, the graph gradient described in Section 2.1 coincides with the hypernode graph gradient for the equivalent hypernode graph, and conversely.

Because of the Equilibrium Condition, the gradient of every constant node function is the zero-valued hyperedge function. This can be written as $\mathbf{1} \in \operatorname{Null}(G)$, where $\operatorname{Null}(G)$ is the set of so-called harmonic functions.

As in the graph case, we define the smoothness of a real-valued node function f over a hypernode graph $\mathbf{h}$ to be $\Omega(f) = \|Gf\|^2 \in \mathbb{R}^+$. We can show that $\sqrt{\Omega}$ still defines a semi-norm on $\mathbb{R}^n$ (the proof of Proposition 1 remains valid). As above, we rewrite Ω under the form

$$\Omega(f) = (Gf)^T(Gf) = f^T G^T G f \ .$$

The square $n \times n$ real valued matrix $G^T G$ does not depend on the orientation function ϵ since

$$\forall i, j, \quad \Delta_{i,j} = \sum_{h \in H} G_{h,i} G_{h,j} = \sum_{h \in H} -P(h,i,j)\sqrt{w_h(i)}\sqrt{w_h(j)} \ . \tag{2.9}$$

This property allow us to define the hypernode graph Laplacian similarly to the graph case:

Definition 3. *Let $\mathbf{h} = (N, H)$ be a hypernode graph. The Laplacian Δ is defined as $G^T G$ where G is the gradient matrix of $\mathbf{h}$ embedded with an arbitrary orientation function ϵ.*

When the hypernode graph is a graph, the hypernode graph Laplacian coincides with the unnormalized Laplacian described in Section 2.1. The hypernode graph Laplacian shares several important properties with the graph Laplacian.

Proposition 6. *Let $\mathbf{h}$ be a hypernode graph and ϵ an orientation function. Let us consider the related gradient G and the Laplacian matrix Δ. Δ is symmetric positive semidefinite and $\mathrm{Null}(\Delta) = \mathrm{Null}(G) = \mathrm{Null}(\Omega)$. As direct consequences, $\mathbf{1} \in \mathrm{Null}(\Delta)$.*

Proof. We have $\Delta^T = (G^T G)^T = G^T G = \Delta$ so Δ is symmetric. For any $f \in \mathbb{R}^n$, we have

$$f^T \Delta f = f^T G^T G f = \|Gf\|^2 = \Omega(f) \geqslant 0 \ .$$

So Δ is positive semidefinite. Moreover, when $\Delta f = 0$, then $f^T \Delta f = \|Gf\|_2^2 = 0$ and $Gf = 0$. Conversely, if $Gf = 0$, then $\Delta f = G^T G f = 0$. Consequently, we have $\mathrm{Null}(\Delta) = \mathrm{Null}(G)$. The last equality $\mathrm{Null}(G) = \mathrm{Null}(\Omega)$ is a direct consequence of $\|Gf\|_2^2 = \Omega(f)$ since $Gf = 0$ if and only if $\|Gf\|_2^2 = 0$. □

Example 7. *We consider three hypernode graphs over $N = \{1, 2, 3\}$ depicted in Figure 2.10. The hypernode graph $\mathbf{h}_9$ shown in the middle of Figure 2.10 has two hyperedges $h_1 = \{\{1\}, \{3\}\}$ and $h_2 = \{\{1, 3\}, \{2\}\}$. The hypernode graph $\mathbf{h}_8$ shown on the left has two hyperedges $h'_1 = \{\{1\}, \{3\}\}$ and $h'_2 = \{\{1\}, \{2\}\}$. The hypernode graph $\mathbf{h}_{10}$ shown on the right has also two hyperedges $h''_1 = \{\{1, 2\}, \{3, 4\}\}$ and $h''_2 = \{\{3\}, \{4\}\}$. The hyperedge weights are detailed in Figure 2.10. For instance in the left hypernode graph $\mathbf{h}_8$,*

we have $w_{h_1}(1) = w_{h_2}(3) = 1$ *and* $w_{h_2}(1) = w_{h_2}(2) = 1$. *For each of the three hypernode graphs, we compute a gradient matrix (based on an arbitrary orientation of the hyperedges) and the Laplacian matrix.*

$$G_8 = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}; \ \Delta_8 = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

$$G_9 = \begin{pmatrix} \frac{-\sqrt{2}}{2} & \sqrt{2} & \frac{-\sqrt{2}}{2} \\ \frac{-\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} \end{pmatrix}; \ \Delta_9 = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

$$G_{10} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}; \ \Delta_{10} = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix}$$

It can be noted that the Laplacian matrix Δ_9 *presented coincide with the graph Laplacian corresponding to the adjacency matrix*

$$W_9 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} .$$

This is not the case for Δ_{10} *who have positive extra-diagonal terms.*

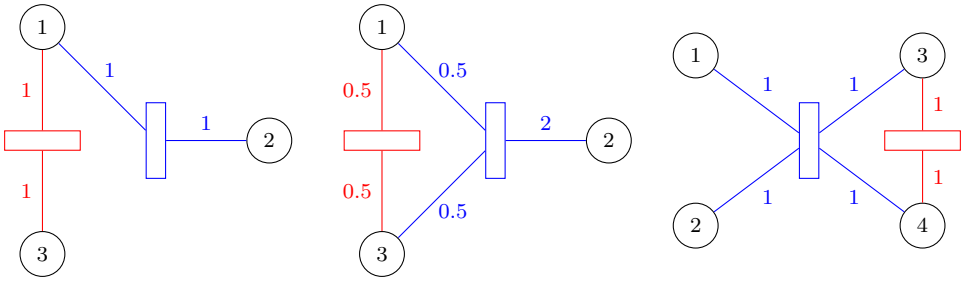


Figure 2.10: Hypernode graphs $\mathbf{h}_8$ (left), $\mathbf{h}_9$ (middle) and $\mathbf{h}_{10}$ (right).

As in the graph case, the Moore-Penrose pseudoinverse of the Laplacian matrix $\Delta^\dagger$ of a hypernode graph $\mathbf{h}$ is symmetric and positive semidefinite. We define the *hypernode graph kernel* of a hypernode graph $\mathbf{h}$ to be the pseudoinverse $\Delta^\dagger$ of its Laplacian. We now give some additional properties of the class of hypernode graph Laplacians.

Proposition 7. *The class of hypernode graph Laplacians with n nodes is*

$$\mathcal{H}(n) = \{M \in \mathbb{R}^{n\times n} \mid M = M^T,\ \mathbf{1} \in \mathrm{Null}(M),\ M \succeq 0\}\ ,$$

where $M \succeq 0$ denotes the positive semi-definiteness property.

Proof. It is an immediate consequence of Proposition 6 that a hypernode graph Laplacian belongs to $\mathcal{H}(n)$. Conversely, we can use Algorithm 1 presented below in order to compute the hypernode graph corresponding to a matrix $M \in \mathcal{H}(n)$.

Algorithm 1 Computing an hypernode graph from $M \in \mathcal{H}(n)$.

Require: $M \in \mathcal{H}(n)$
Define $N = \{1, \ldots, n\}$ and $H = \varnothing$
Compute a square root decomposition $M = G^T G$
for each row $\mathbf{r}$ of G **do**
  Create a new hyperedge $h = \{s_h, t_h\}$ with $s_h = t_h = \varnothing$
  for each node $i \in N$ **do**
    Define $w_h(i) = \mathbf{r}(i)^2$
    if $\mathbf{r}(i) < 0$ **then**
      Add i to s_h
    else if $\mathbf{r}(i) > 0$ **then**
      Add i to t_h
    end if
  end for
  {Since $1 \in \mathrm{Null}(M)$, we have $\sum_i \mathbf{r}(i) = 0$ and the Equilibrium Condition is satisfied for h}
  Add h to H
end for
return The hypernode graph $\mathbf{h} = (N, H)$
{G is a valid gradient matrix of $\mathbf{h}$ and consequently, $M = G^T G$ is the Laplacian of $\mathbf{h}$}

□

2.2.3 Equivalent hypernode graphs

In the previous Section, we have defined the Laplacian matrix Δ of a hypernode graph $\mathbf{h}$ as $G^T G$ where G is a gradient of $\mathbf{h}$ (given an arbitrary orientation of the hyperedges). An interesting question is whether $\mathbf{h}$ is the unique hypernode graph associated to Δ, i.e., whether we can find a hypernode graph $\mathbf{h}' \neq \mathbf{h}$ such that its Laplacian Δ' coincide with Δ. The answer to this question is positive and is a direct consequence of the non-uniqueness

of the square root decomposition of a positive semi-definite matrix. To illustrate this fact, let us consider the Laplacian Δ_{10} corresponding to the hypernode graph $\mathbf{h}_{10}$ (see Example 7). The matrix

$$G_{11} = \begin{pmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & \sqrt{2} & 0 \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & \sqrt{2} \end{pmatrix}$$

is a square root of Δ_{10} since $\Delta_{10} = G_{11}^T G_{11}$. From G_{11}, we can define the hypernode graph $\mathbf{h}_{11} \neq \mathbf{h}_{10}$ presented in Figure 2.11. Based on this observation, we define the notion of hypernode graph equivalence:

Definition 4. *Two hypernode graphs are said to be* equivalent *if they have the same Laplacian matrix.*

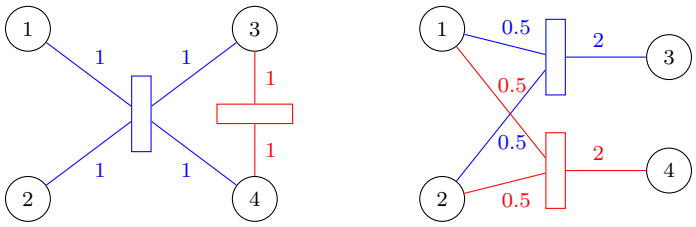


Figure 2.11: Equivalent hypernode graphs $\mathbf{h}_{10}$ (left) and $\mathbf{h}_{11}$ (right)

Equivalent hypernode graphs have the same Laplacian and therefore have the same smoothness measure Ω. We can note that the smoothness constraints expressed by the hyperedges of $\mathbf{h}_{11}$ are consistent with the smoothness constraints expressed by the hyperedges of $\mathbf{h}_{10}$. Indeed, any smooth function f on $\mathbf{h}_{10}$ must satisfy $f(1)+f(2)$ close to $f(3)+f(4)$ and $f(3)$ close to $f(4)$. In $\mathbf{h}_{11}$, a smooth function should have $f(1)+f(2)$ close to $2f(3)$ and to $2f(4)$. Thus, the constraints from $\mathbf{h}_{11}$ are a linear combination of the ones from $\mathbf{h}_{10}$. From an algebraical perspective, we can notice that G_{11} can be written as QG_{10} where

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix} .$$

Q is an isometry ($Q^T Q = I$) that expresses the linear relations between the constraints of both hypernode graphs.

2.3 Expressiveness of the Laplacian framework

We address in this section the important question of the expressiveness of hypernode graphs. We will base our reflexion on Agarwal et al. (2006) who

study the case of hypergraphs, a popular higher-order structure first introduced by Berge (1989). Agarwal et al. (2006) shows notably that the different attempts to build hypergraph Laplacians have actually led to learning problems that can be expressed and analyzed using standard graph Laplacians. The objective of this section is to show that the same does not happen in the case of hypernode graphs, which will allow us to claim that our model is strictly more expressive than the undirected graph model.

Let us first formalize the notion of expressivity. Recall that the main purpose of a Laplacian matrix is to define a notion of smoothness for the real-valued node functions. Consequently, two Laplacian can be regarded as equivalent on a set of nodes S if they define consistent notions of smoothness for the real-valued functions defined on S. We formalize this intuition using the following definition.

Definition 5. *A Laplacian pair $(M, \mathcal{E})$ consists of a positive semi-definite matrix $M \in \mathbb{R}^{n \times n}$ together with a discrete space $\mathcal{E}$ of size n. The associated smoothness measure Ω_M is defined by the mapping*

$$\Omega_M : f \in \mathbb{R}^n \to f^T M f \in \mathbb{R}^+ \quad .$$

($\mathbb{R}^n$ can be regarded as the set of real-valued functions defined on $\mathcal{E}$)

In the case of a graph $\mathbf{g} = (N, E)$ associated with the Laplacian matrix Δ, (Δ, N) is a Laplacian pair and the associated smoothness measure is the smoothness measure described in Section 2.1. We define the notion of generalization using the associated smoothness measures:

Definition 6. *Let $(M_1, \mathcal{E}_1)$ and $(M_2, \mathcal{E}_2)$ be two Laplacian pairs such that $\mathcal{E}_1 \subsetneq \mathcal{E}_2$. We say that M_2* generalizes *M_1 if the smoothness measures associated with M_1 and M_2 are consistent for the real-valued functions defined on $\mathcal{E}_1$, i.e., if the following properties hold:*

$$\forall f^{(1)} \in \mathrm{Null}(\Omega_{M_1}), \quad \exists f^{(2)} \in \mathrm{Null}(\Omega_{M_2}) \text{ such that } f^{(2)}\Big|_{\mathcal{E}_1} = f^{(1)} \qquad (*)$$

$$\forall f^{(2)} \in \mathrm{Null}(\Omega_{M_2}), \quad f^{(2)}\Big|_{\mathcal{E}_1} \in \mathrm{Null}(\Omega_{M_1}) \qquad (**)$$

The first property ($*$) states that any function that is perfectly smooth on the set $\mathcal{E}_1$ can be extended to a perfectly smooth function on the set $\mathcal{E}_2$. Conversely, ($**$) states that any function that is perfectly smooth on $\mathcal{E}_2$ can be restricted to a perfectly smooth function on $\mathcal{E}_1$.

Note that, for any Laplacian pair $(M, \mathcal{E})$, the relation $\mathrm{Null}(\Omega_M) = \mathrm{Null}(M)$ holds true. Indeed, if $Mf = 0$, then $\Omega_M(f) = f^T M f = 0$. The reverse comes from the fact that M is positive semi-definite: we can write $M = G^T G$

and $\Omega_M(f) = \|Gf\|_2^2$ so if $\Omega_M(f) = 0$, $Gf = 0$ and $\Delta f = G^T G f = 0$. As a consequence, another way to look at Definition 6 is to say that the eigenvalue problem $M_1 f^{(1)} = 0 f^{(1)}$ should be somehow equivalent to the eigenvalue problem $M_2 f^{(2)} = 0 f^{(2)}$. Agarwal et al. (2006) follow a similar idea by considering that a hypergraph Laplacian Δ_H can be reduced to a graph Laplacian Δ if the eigenvalue problem $\Delta_H f = \lambda f$ for $\lambda \in \mathbb{R}^+$ is induced by a similar eigenvalue problem on Δ. In both cases, no constraints are imposed on the shapes of the matrices: the generalizing matrix can be strictly bigger than the original matrix. As a result, we will be able to consider the case of graph approximations with additional nodes.

2.3.1 The case of hypernode graphs

Based on Definition 6, we now prove the following

Proposition 8. *The class of hypernode graph Laplacians is* more expressive *than the class of graph Laplacians, i.e., there exist some hypernode graph Laplacians for which we cannot find any generalizing graph Laplacian.*

Proof. Let us consider the hypergraph $\mathbf{h}_7$ from Example 5 and depicted in Figure 2.12). Let us also define $S = \{1, 3\}$. The indicator vector 1_S is in Null(Δ_7) and, thus, define a smooth function on $\mathbf{h}_7$. Let us consider a graph $\mathbf{g} = (N_e, E_e)$ whose nodeset N_e contains N_7 and whose Laplacian Δ_e generalizes Δ_7 as described in Definition 6. Following the first property ($*$), we can find $f_e : N_e \to \mathbb{R}$ such that $f_e\big|_{N_7} = \mathbf{1}_S$ and $f_e \in \text{Null}(\Delta_e)$.

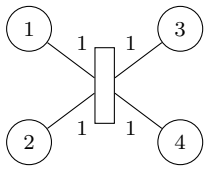
Since $\mathbf{g}_e$ is a graph, we know that Null(Δ_e) is spanned by the indicator vectors of the connected components of $\mathbf{g}_e$ (see Proposition 2 in Section 2.1). Since $f_e(1) = \mathbf{1}_S(1) \neq f_e(2) = \mathbf{1}_S(2)$, 1 and 2 must be in different components in $\mathbf{g}_e$. Note that the same holds with $f_e(1) \neq f_e(4)$, $f_e(3) \neq f_e(2)$ and $f_e(3) \neq f_e(4)$. By following a similar reasoning with $S = \{1, 4\}$, we eventually deduce that the four original nodes 1, 2, 3 and 4 must be in distinct components in $\mathbf{g}_e$.

Let us now consider the distinct components C_1 and C_2 in $\mathbf{g}_e$ that contains respectively the nodes 1 and 2. If f'_e denotes the indicator vector of $C_1 \cup C_2$, we have

$$f'_e \in \text{Null}(\Delta_e) \ .$$

Consequently and because of the second property ($**$) of Definition 6, the restriction of f'_e to the original nodeset N_7 should be in the nullspace of Δ_7. However, we have $f'_e\big|_{N_7} = 1_{S'}$ where $S' = \{1, 2\}$ and

$$\Delta_7 1_{S'} = 4 \neq 0 \ .$$



$$\Delta_7 = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}$$

Figure 2.12: Hypernode graph $\mathbf{h}_7$ and Laplacian Δ_7

Thus, we can't find a graph Laplacian which generalizes Δ_7, which concludes the proof. □

Note that the Laplacian Δ_7 has some positive extradiagonal terms and, therefore, cannot coincide with a graph Laplacian. Proposition 8 goes further and allows to claim as well that it cannot be approximated by a graph with a finite nodeset.

2.3.2 The case of hypergraph Laplacians

In this section, we propose to review the case of hypergraphs and show that the different hypergraph Laplacians that have been proposed in recent years are not more expressive than graph Laplacians. First, let us recall basic notions about hypergraphs.

Hypergraphs have been introduced by Berge (1989) in order to model problems where relationships are no longer binary, that is when they involve more than two individuals. Hypergraphs have been used for instance in bioinformatics (Klamt et al. 2009), computer vision (Zhang et al. 1993) and natural language processing (Cai and Strube 2010). In the hypergraph model, a *hyperedge* is simply a set of nodes. The key idea is to encode the fact that several nodes share a common property. An example is given in Figure 2.13 where nodes are documents described by two features (type of document and language). We create two hyperedges: the first one $\{1, 2, 3\}$ contains all the documents written in French while the second one $\{2, 3, 4\}$ contains all the reports. Note that we do not create any additional hyperedge since there is only one document written in English and one book in the example dataset. A hypergraph is formally defined as a set of nodes N together with a set of hyperedges E. Each hyperedge $e \in E$ is a set of nodes and is embedded with a positive weight $w(e) \in \mathbb{R}^+$.

As said above, several attempts have been made to define Laplacian operators for hypergraphs. The most popular ones are the Laplacians Δ_{B} from Bolla (1993), Δ_{R} from Rodríguez (2003) and Δ_{ZHS} from Zhou et al. (2006). We have the following result:

Node	Language	Type
1	French	Book
2	French	Report
3	French	Report
4	English	Report

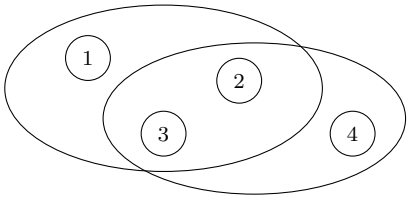


Figure 2.13: Hypergraph with two hyperedges $e_1 = \{1, 2, 3\}$ and $e_2 = \{2, 3, 4\}$ based on the categorical features *Language* and *Type*. $w(e_1)$ and $w(e_2)$ are set to 1.

Proposition 9. *Let us consider a hypergraph on a nodeset N and the Laplacian pairs (Δ_{R}, N), (Δ_{B}, N) and $(\Delta_{\mathrm{ZHS}}, N)$. We can show that Δ_{R} and Δ_{B} are equal to graph Laplacian using an adequate expansion. Moreover, we can find a normalized graph Laplacian Δ_s that generalizes Δ_{ZHS} according to Definition 6.*

Proof. The proof of this proposition is partly based on Agarwal et al. (2006). First note that hypergraphs can be approximated by graphs using two main constructions:

- The *clique expansion*, where each hyperedge e is replaced by an uniformally weighted clique. When the pair is contained in multiple hyperedges, the final weight is either the average or the sum of the weights corresponding to each clique.
- The *star expansion*, where we introduce a new node for each hyperedge $e \in E$. Each of the original nodes of e is linked to the new node through an edge. The star expansion is denoted by $\mathbf{h}_s = (N \cup N_s, E_s)$ where N_s is the set of additional nodes.

Examples of clique expansion and star expansion are given in Figure 2.14. Agarwal et al. (2006) shows that Δ_{B} coincide with the graph Laplacian of a clique expansion built from the original hypergraph. The same result holds for Δ_{R} with a slightly different clique expansion.

In the case of Δ_{ZHS}, Agarwal et al. (2006) builds a star expansion $\mathbf{h}_s = (N \cup N_s, E_s)$ where the edges between the original nodes and the new nodes are weighted by the original hyperedges weights $w(e)$. We denote by n the size of the original nodeset N and by n_s the size of the additional nodeset N_s. Agarwal et al. (2006) shows that the normalized Laplacian of the star expansion can be written as

$$\Delta_s = \begin{pmatrix} I_n & -A \\ -A^T & I_{n_s} \end{pmatrix} ,$$

where A is a rectangular matrix $n \times n_s$ that satisfies $\Delta_{\mathrm{ZHS}} = I - AA^T$.

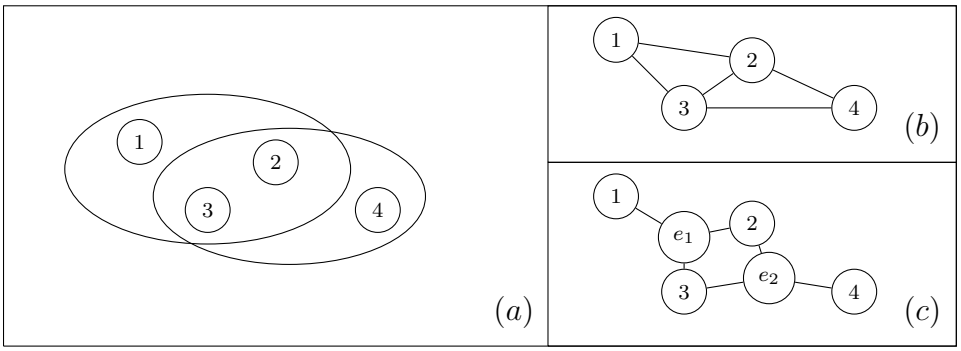


Figure 2.14: Hypergraph (a), clique expansion (b) and star expansion (c)

Based on this property, Agarwal et al. (2006) shows a general relation between the eigenvalue problems on Δ_{ZHS} and Δ_s. Namely for $\lambda \in \mathbb{R}^+$, we have

$$\Delta_s \begin{pmatrix} f \\ f_s \end{pmatrix} = \lambda \begin{pmatrix} f \\ f_s \end{pmatrix} \Rightarrow \begin{cases} f - Af_s = \lambda f \\ -A^T f + \quad f_s = \lambda f_s \end{cases} \Rightarrow (1-\lambda)^2 f = AA^T f \ ,$$

which leads to $\Delta_{\text{ZHS}} f = (1-(1-\lambda)^2) f$. By taking $\lambda = 0$, we get $f \in \text{Null}(\Delta_s)$ which is exactly the property $(**)$. The reverse property $(*)$ does not figure in Agarwal et al. (2006) but is easy to show as well. For any $f \in \text{Null}(\Delta_{\text{ZHS}})$, the objective is to find f_s such that

$$\begin{pmatrix} f \\ f_s \end{pmatrix} \in \text{Null}(\Delta_s) \ .$$

The choice $f_s = A^T f$ satisfies this requirement since

$$\Delta_s \begin{pmatrix} f \\ A^T f \end{pmatrix} = \begin{pmatrix} I_n & -A \\ -A^T & I_{n_s} \end{pmatrix} \begin{pmatrix} f \\ A^T f \end{pmatrix} = \begin{pmatrix} (I - AA^T) f \\ -A^T f + A^T f \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \ .$$

Consequently, Δ_s generalizes Δ_{ZHS} as described in Definition 6, which concludes the proof. □

A recent tentative to fully use the hypergraph structure was proposed by Hein et al. (2013). In this paper, the authors propose to use a dedicated hypergraph cut and introduce the total variation on a hypergraph as the Lovasz extension of this cut. This allows to define a regularization functional on hypergraphs for defining semi-supervised learning algorithms. However, because of its nonlinearity, this term does not define a Laplacian matrix on the hypergraph.

2.4 Conclusion

In this chapter, we have introduced *hypernode graphs* as a new abstract model that generalizes undirected graphs and allows to encode similarity relations between sets of nodes. We have built a consistent extension of the Laplacian framework for these objects and have shown that our new class of hypernode graph Laplacians was more expressive than the class of graph Laplacians, which is not the case for other higher-order models such as hypergraphs. In the next chapter, we will show important properties of hypernode graphs and study the notions of paths, connectivity and distances.

Chapter 3

Properties of hypernode graphs

Chapter abstract In this chapter, we examine the properties of hypernode graphs. We first establish relations between hypernode graphs and a specific class of signed graphs (that contains the undirected graphs). Second, we study whether the notions of path and of connected component can be extended to the case of (undirected) hypernode graphs. Finally, we review the notions of kernels and distances for hypernode graphs and compare them with the graph distances defined in Section 2.1.2.

3.1 Hypernode graphs, graphs and signed graphs

Recall that the Laplacian Δ of a graph $\mathbf{g} = (N, E)$ can be computed from the adjacency matrix W and the corresponding degree matrix D with the equation $\Delta = D - W$ (see Section 2.1). The first objective of this section is to show that the Laplacian of a hypernode graph $\mathbf{h}$ can be computed in a similar way from a symmetrical matrix called *pairwise weight matrix*. Then, we will discuss the relationships between hypernode graphs and a specific class of signed graphs.

3.1.1 Pairwise Weight Matrix and Laplacians

Let us consider in this section a hypernode graph $\mathbf{h} = (N, H)$ with Laplacian matrix Δ. We recall Equation (2.9) from Section 2.2.2:

$$\forall i, j, \quad \Delta_{i,j} = \sum_{h \in H} -P(h, i, j)\sqrt{w_h(i)}\sqrt{w_h(j)} \ . \tag{2.9}$$

In the case of a graph, the extradiagonal elements of Δ are equal to the opposite of the edge weights (see Section 2.1.1). Based on this constatation, we define for every node pair (i, j) the *pairwise weight* $W_{i,j}$ as the opposite of the extradiagonal term $\Delta_{i,j}$. Formally, we get from (2.9)

$$\forall i \neq j, \quad W_{i,j} = \sum_{h \in H} P(h, i, j)\sqrt{w_h(i)}\sqrt{w_h(j)} \ . \tag{3.1}$$

The pairwise weight $W_{i,j}$ is computed as a sum over all the hyperedges and, for this reason, can be understood as an aggregation of all the information that exists between i and j. For every hyperedge h, we define the hyperedge pairwise weight as

$$w_h(i, j) = \delta_{(i \neq j)} P(h, i, j)\sqrt{w_h(i)}\sqrt{w_h(j)} \ ,$$

where δ stands for the negative Kronecker delta: $\delta_{(i \neq j)} = 1$ if $i \neq j$ and 0 otherwise. With this definition, Equation (3.1) becomes

$$W_{i,j} = \sum_{h \in H} w_h(i, j) \ .$$

The quantity $\sqrt{w_h(i)}$ can be viewed as the cost of entering the hyperedge h at node i and $\sqrt{w_h(j)}$ as the cost of exiting the hyperedge h at node j. $w_h(i, j)$ is null as soon as either i or j does not belong to the hyperedge h. When h is a pair of two singletons $\{i\}$ and $\{j\}$, $w_h(i)$ and $w_h(j)$ are equal because of the equilibrium condition, and the hyperedge pairwise edge satisfies $w_h(i, j) = \sqrt{w_h(i)}\sqrt{w_h(j)} = w_h(i) = w_h(j)$.

We finally define the *pairwise (hypernode graph) weight matrix* W by fixing $W_{i,i} = 0$ for all i. It can be noted that the pairwise weight matrix of a graph $\mathbf{g}$ (with no self-loops) considered as a hypernode graph is equal to the adjacency matrix of the graph $\mathbf{g}$. Indeed, for $i \neq j$, the pairwise weight $W_{i,j}$ is by construction the opposite of the term $\Delta_{i,j}$, which is equal to the edge weight in the graph case. Thus, the pairwise weight matrix can be seen as the extension of the graph adjacency matrix. The hypernode graph degrees (see Equation 2.6) can be computed directly from the pairwise weight matrix because

$$\begin{aligned} \forall i \in N, \ \sum_j W_{i,j} &= \sum_h \sqrt{w_h(i)} \sum_{j \neq i} P(h, i, j)\sqrt{w_h(j)} \\ &= \sum_h w_h(i) \quad \text{(Equilibrium Condition)} \\ &= d(i) \ . \end{aligned} \tag{3.2}$$

Consequently, the diagonal degree matrix D is the unique matrix that satisfies $D\mathbf{1} = W\mathbf{1}$. Based on these observations, we can now show that, as in the graph case, the Laplacian matrix can be expressed in terms of the pairwise weight matrix W and of the diagonal degree matrix D.

Proposition 10. *Let* $\mathbf{h} = (N, H)$ *be a hypernode graph, let* W *be the pairwise weight matrix of* $\mathbf{h}$*, and let* D *be the corresponding degree matrix. Then, the unnormalized Laplacian of* $\mathbf{h}$ *is* $\Delta = D - W$*.*

Proof. By construction, the extradiagonal terms of Δ are the opposite of the extra-diagonal terms of W. It remains to show that the diagonal terms of Δ are equal to the degree of the hypernode graph. Because of Equation (2.9), we can write for $i \in N$,

$$\Delta_{i,i} = \sum_h P(h,i,i)\sqrt{w_h(i)}\sqrt{w_h(i)} = \sum_h w_h(i) = d(i) \ ,$$

which concludes the proof. □

Example 8. *(Example 7 continued) For each hypernode graph presented in Example 7, we compute the pairwise weight matrix* W *and the diagonal degree matrix* D*. We present the matrices in Figure 3.1, along with the original hypernode graphs. One can verify that* $D - W$ *is equal to the Laplacian matrices presented in Example 7.*

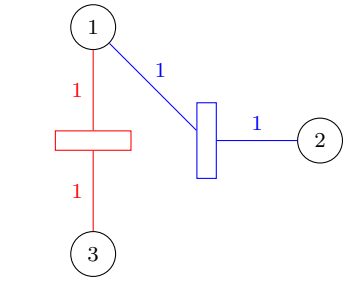
As a consequence of Proposition 10, we can leverage the pairwise weight matrix to characterize equivalent hypernode graphs and give a property of the degree matrices as

Corollary 1. *Two hypernode graphs are equivalent if and only if they have the same pairwise weight matrix. Two equivalent hypernode graphs have the same degree matrix.*

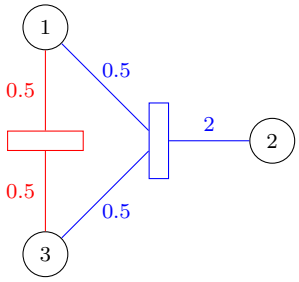
Proof. Proposition 10 states that a hypernode graph Laplacian Δ can be written in a unique way as $D - W$ with D a diagonal matrix and W a matrix with diagonal terms equal to 0. Thus, equivalent hypernode graphs will necessarily share the same pairwise matrix W and the same degree matrix D. Conversely, if two hypernode graphs share the same pairwise weight matrix W, the uniqueness of the form $D - W$ implies that they also share the same Laplacian and are, therefore, equivalent. □

3.1.2 Signed graph reduction

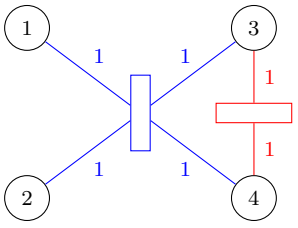
As shown in the previous section, the Laplacian matrix of a hypernode graph can be written under the form $D - W$ where W is the *pairwise weight matrix* and D the corresponding degree matrix. As shown in Example 8, the pairwise weight matrix can contain negative weights and, thus, cannot be interpreted as an adjacency matrix of a graph. However, W can be interpreted as the adjacency matrix of a *signed graph*, i.e., the adjacency matrix of an undirected graph with possibly negative weights. Following this idea, we define the notion of *reduced signed graph*.



$$W_8 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \; ; \; D_8 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



$$W_9 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \; ; \; D_9 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



$$W_{10} = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \; ; \; D_{10} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Figure 3.1: Pairwise weights matrices and degree matrices for the hypernode graphs $\mathbf{h}_8$, $\mathbf{h}_9$ and $\mathbf{h}_{10}$.

Definition 7. *The* reduced signed graph *of a hypernode graph* $\mathbf{h}$ *is the signed graph* $\tilde{\mathbf{g}}$ *with adjacency matrix* W*, where* W *is the pairwise weight matrix of the hypernode graph* $\mathbf{h}$*.*

Since equivalent hypernode graphs share the same pairwise weight matrix (see Corollary 1), they also share the same reduced signed graph. It is easy to see that the reduced signed graph $\tilde{\mathbf{g}}$ of a graph $\mathbf{g}$ is equal to $\mathbf{g}$. Consequently, we can easily characterize the hypernode graphs that are equivalent to a graph.

Proposition 11. *A hypernode graph* $\mathbf{h}$ *is equivalent to a graph* $\mathbf{g}$ *if and only if the reduced signed graph* $\tilde{\mathbf{g}}$ *of* $\mathbf{h}$ *is a graph. And then,* $\mathbf{g} = \tilde{\mathbf{g}}$ *is the unique graph equivalent to* $\mathbf{h}$*.*

Proof. Let us assume that $\mathbf{g}$ is a graph equivalent to $\mathbf{h}$. Necessarily, the pairwise weight matrix W of $\mathbf{h}$ is also the pairwise weight matrix of $\mathbf{g}$ which is equal to the adjacency matrix of $\mathbf{g}$. Then $\tilde{\mathbf{h}}$ is equal to the graph $\mathbf{g}$. Conversely, if $\tilde{\mathbf{h}}$ is a graph, then its pairwise weight matrix is equal to the adjacency matrix of a graph $\mathbf{g}$. Since $\mathbf{g}$ and $\mathbf{h}$ share the same pairwise weight matrix, Corollary 1 allows us to state that $\mathbf{h}$ and $\mathbf{g}$ are equivalent. □

Example 9. *(Example 8 continued) We consider again the hypernode graphs* $\mathbf{h}_8$*,* $\mathbf{h}_9$ *and* $\mathbf{h}_{10}$ *presented in Figure 3.1.* $\mathbf{h}_8$ *is a graph and is consequently equal to its reduced signed graph. The reduced signed graph* $\tilde{\mathbf{g}}_9$ *associated with* $\mathbf{h}_9$ *is depicted on Figure 3.2.* $\tilde{\mathbf{g}}_9$ *is a graph and, therefore, is the unique graph equivalent to* $\mathbf{h}_9$*. In the case of* $\mathbf{h}_{10}$ *the reduced signed graph* $\tilde{\mathbf{g}}_{10}$ *is depicted in Figure 3.3 (we use wavy lines to represent edges with negative weights).* $\tilde{\mathbf{g}}_{10}$ *is not a graph, which shows that* $\mathbf{h}_{10}$ *has no equivalent graph.*

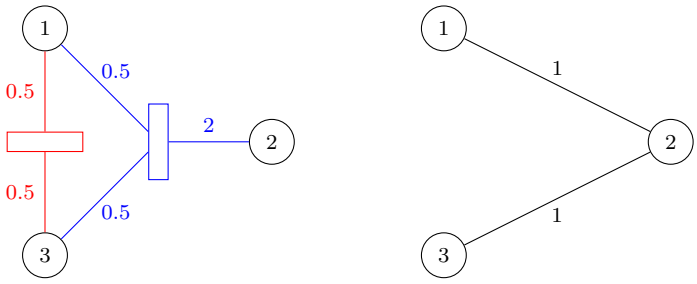


Figure 3.2: Hypernode graph $\mathbf{h}_9$ and its reduced signed graph $\tilde{\mathbf{g}}_9$

So far we have shown that every hypernode graph can be reduced to a signed graph. That is, every hypernode graph can be associated to a signed graph through its pairwise weight matrix. Note that the converse is not necessarily true, as shown in the following proposition.

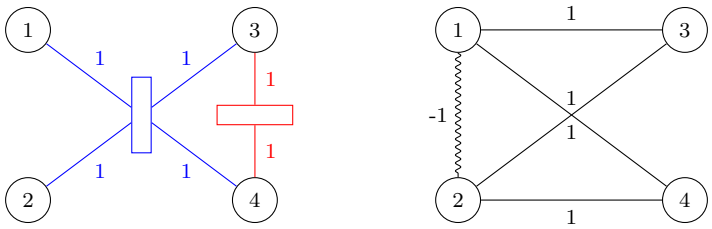
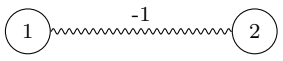


Figure 3.3: Hypernode graph $\mathbf{h}_{10}$ and its reduced signed graph $\tilde{\mathbf{g}}_{10}$

Proposition 12. *Let $\tilde{\mathbf{g}}$ be a signed graph with adjacency matrix W. Similarly to the graph case, we define the degree matrix D to be the unique diagonal matrix such that $W\mathbf{1} = D\mathbf{1}$. $\tilde{\mathbf{g}}$ is the reduced signed graph of a hypernode graph if and only if the matrix $D - W$ is positive semi-definite.*

Proof. Since $W\mathbf{1} = D\mathbf{1}$, we always have $\mathbf{1} \in \text{Null}(D - W)$. Moreover, since W is symmetric, $D - W$ is also symmetric. Let us recall that $\mathcal{H}(n) = \{M \in \mathbb{R}^{n\times n} \mid M = M^T,\ \mathbf{1} \in \text{Null}(M),\ M \succeq 0\}$. If $D - W$ is positive semidefinite, then we have directly $D - W \in \mathcal{H}(n)$. Conversely, if W is the pairwise weight matrix of a hypernode graph, then $D - W$ is a hypernode graph Laplacian and is thus positive semidefinite. □

Example 10. *Let us consider the signed graph $\tilde{\mathbf{g}}_{12}$ with $N = \{1, 2\}$ and $W_{1,2} = -1$ presented on Figure 3.4. The matrix $D - W$ is not positive semidefinite so the signed graph $\tilde{\mathbf{g}}_{12}$ is not the reduced graph of a hypernode graph.*



$$D - W = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

Figure 3.4: Signed graph $\tilde{\mathbf{g}}_{12}$

Thus, the class of pairwise weight matrices corresponds to the case where the matrix $D - W$ is positive semi-definite, and such a matrix has a semantic defined by a hypernode graph. The class of signed graphs introduced in Proposition 12 is similar to the class of PSD-graphs introduced in Koren et al. (2002) (in our case, the set of masses $\mathcal{M}$ is uniform). We now discuss the semantic link between a hypernode graph and its reduced signed graph:

Proposition 13. *For any real-valued function* $f : N \to \mathbb{R}$,

$$\Omega(f) = \underbrace{\sum_{\{i,j\}\in E} |W_{i,j}| \left[f(j) - \mathrm{sgn}(W_{i,j}) f(i)\right]^2}_{\xi_1(f)} + \underbrace{\sum_{i=1}^{n} \left[d(i) - \tilde{d}(i)\right] f(i)^2}_{\xi_2(f)} \quad , \quad (3.3)$$

where $\tilde{d}(i) = \sum_{j\neq i} |W_{i,j}|$ *is the* absolute degree *of the node* i *and where* sgn *denotes the* sign *operator:* $\mathrm{sgn}(x) = 1$ *if* $x \geqslant 0$ *and* -1 *otherwise.*

Proof. This result comes directly from Herbster (2008) who computes a development of $f^T M f$ for any symmetric matrix M. By definition of the reduced signed graph, we can write $\Omega(f) = f^T(D - W)f$, which leads to Equation 3.3. Note that we still have $f^T(D - W)f = \xi_1(f) + \xi_2(f)$ when $D - W$ is not positive semi-definite. □

The first term $\xi_1(f)$ can be interpreted as follows: if a positive edge exists between nodes i and j then $f(i)$ must be close to $f(j)$; in the case of a negative edge, $f(i)$ must be close to $-f(j)$. Note that this is the usual semantic associate with a signed graph according to the *social balance* theory (see for example Chiang et al. 2012). This theory is based on the following principles:

1. A friend of my friend is my friend
2. An enemy of my friend is my enemy
3. An enemy of my enemy is my friend

A positive edge can be seen as a "friendship" relation between two nodes, while a negative edge indicates that the two nodes are "enemies". For instance, let us assume that a node i is the enemy of a second node j, which is itself the enemy of a third node k (chain of two negative edges). In order to minimize $\xi_1(f)$, we should have $f(i) \approx -f(j)$ and $f(j) \approx -f(k)$, which leads to $f(i) \approx f(k)$ (friendship relation between i and k, as claimed by the third principle).

The second term $\xi_2(f)$ involves the differences between the degrees and the absolute degrees of the nodes. The semantic of this term is hard to interpret in the general case. It can be noted that $\xi_2(f) \leqslant 0$ for all f since

$$\forall i, \tilde{d}(i) = \sum_{j\neq i} |W_{i,j}| \geqslant d(i) = \sum_{j\neq i} W_{i,j} \quad .$$

We will see in Section 5.1.4 that, in the discrete case where $f = \mathbf{1}_C$ with $C \subseteq N$, the semantic of $\xi_1 + \xi_2$ is far more easy to apprehend since it reduces to a notion of cut on the reduced signed graph.

In the case of a hypernode graph, $\Omega(f) = \xi_1(f) + \xi_2(f) \geqslant 0$ (positive semi-definiteness of $D - W$). When $D - W$ is not positive semi-definite, we can still write (as stated in the proof of Proposition 13)

$$f^T(D - W)f = \xi_1(f) + \xi_2(f) \ ,$$

but the quantity is not guaranteed to be greater than 0 anymore. When $D-W$ is indefinite (general case of a signed graph), many essential properties are lost. Among others, it becomes impossible to define a valid notion of smoothness measure and a valid notion of distance. A popular approach that allows to circumvent this issue (see Hou 2005; Goldberg et al. 2007; Kunegis et al. 2010) is to forget about $\xi_2(f)$ and focus on the first term $\xi_1(f)$. This idea leads to the definition of an alternate gradient operator called *signed gradient*:

$$\text{sgrad}(f)(i,j) = \sqrt{|W_{i,j}|}\,(f(j) - \text{sgn}(W_{i,j})f(i)) \ .$$

This gradient is fully consistent with the graph case since $\text{sgrad}(f)(i,j) = \text{grad}(f)(i,j)$ as soon as $W_{i,j} \geqslant 0$. The corresponding Laplacian is defined as $\Delta_s = G_s^T G_s$ where G_s is the matrix of sgrad. We can observe that, for any real-valued node function f, we have $f^T \Delta_s f = \xi_1(f)$. The (signed) Laplacian Δ_s is proven to be equal to $\tilde{D} - W$ where $\tilde{D}$ is the diagonal matrix formed with the absolute degrees $\tilde{d}(i)$.

Herbster (2008) proposes a different approach and consider the class $\mathcal{S}(n)$ of symmetric diagonally dominant matrices with nonnegative diagonal entries

$$\mathcal{S}(n) = \{M \in \mathbb{R}^{n \times n} \text{ such that } M = M^T \text{ and } \forall i, M_{i,i} = |M_{i,i}| \geqslant \sum_{j \neq i} |M_{i,j}|\} \ .$$

Note that all the matrices in $\mathcal{S}(n)$ are positive semidefinite by construction. The class $\mathcal{S}(n)$ contains all the signed Laplacians Δ_s defined above since

$$(\Delta_s)_{i,i} = \tilde{d}(i) = \sum_{j \neq i} |W_{i,j}| \ .$$

Any matrix of the form $D-W$ that belongs to $\mathcal{S}(n)$ satisfies $d(i) = \tilde{d}(i)$ and, therefore, is a graph Laplacian. Indeed, the only solution to get $d(i) = \tilde{d}(i)$ for all $i \in N$ is to have $W_{i,j} \geqslant 0$ for every pair (i,j) (in this case, $\xi_2(f) = 0$).

3.2 Paths and components in hypernode graphs

A *path* on a graph $\mathbf{g} = (N, E)$ is usually defined as a sequence of distinct nodes $i_1, \ldots, i_p \in N$ that are connected by a sequence of edges. Formally,

we must have (see Section 2.1.1)

$$\forall k = 1 \dots p-1, \quad W_{i_k, i_{k+1}} \neq 0 \ .$$

In this section, we discuss how to extend this definition in the case of hypernode graphs. We will pay a particular attention to the definitions that satisfies the following properties:

(a) If $\mathbf{h}$ is a hypernode graph which is also a graph, the paths on $\mathbf{h}$ should be equal to the graph paths.

(b) If $\mathbf{h}$ and $\mathbf{h}'$ are two equivalent hypernode graphs (see Section 2.2.3), a path on $\mathbf{h}$ should be also a path on $\mathbf{h}'$.

The first requirement (a) ensures the consistency with the graph case while the second one (b) ensures the consistency with the hypernode graph semantic. Indeed, two equivalent hypernode graphs can be regarded as semantically equivalent in the sense that they share the same smoothness measure.

As an illustration, let us consider the notion of path based on a simple hyperedge walk: a jump from node i to node j is regarded as valid if we can find a hyperedge h such that $P(h, i, j) = 1$ (i and j belong to different hypernodes in h). We give an example in Figure 3.5 using the hypernode graph $\mathbf{h}_5$ from Example 3. This notion of path appears to be intuitive and satisfy the requirement (a). However it does not satisfy the second requirement (b). Indeed, the reduced signed graph of $\mathbf{h}_5$ is a graph with two separated components (also depicted in Figure 3.5) and no valid path between 1 and 3. Note that this limitation is actually not surprising since hyperedges corresponds to set-based interactions while a path should be regarded as a sequence of pairwise interactions. Moreover, different hyperedges can bring non orthogonal information that should be somehow gathered before considering pairwise jumps.

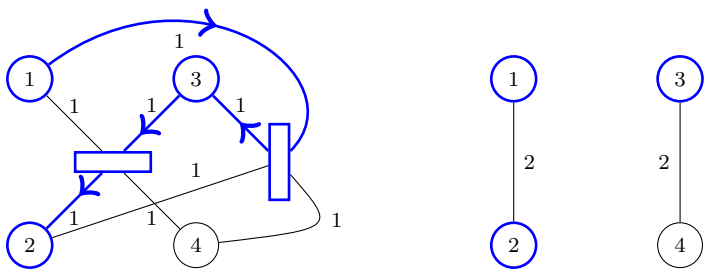


Figure 3.5: (*left*) Hyperedge walk $1 \to 3 \to 2$ on hypernode graph $\mathbf{h}_5$ and (*right*) the reduced signed graph $\tilde{\mathbf{g}}_5$

In the next section, we propose a simple notion of path in hypernode graphs that satisfies the requirements (a) and (b). Based on this definition, we study the related notion of connectivity and components. Note that in the graph case, a strong link exists between connected components and smooth functions. Indeed, we have seen in Proposition 2 that the set of functions that nullify the smoothness measure Ω was spanned by the indicator vectors of the connected components. We discuss whether such a relation exists in hypernode graphs.

3.2.1 Paths and signed components

In order to define a non-ambiguous notion of path, we base our analysis on the pairwise weight matrix defined in Section 3.1.1 . Defined as a sum over all the hyperedges, the pairwise weight allows to take into account all the information between a given pair of nodes. Moreover, it is fully consistent with the equivalence relation since two equivalent hypernode graphs share the same pairwise weight matrix (see Corollary 1).

Definition 8. *Let* $\mathbf{h} = (N, H)$ *be a hypernode graph with pairwise weight matrix* W*. A* path *between two nodes* i *and* j *is a sequence of nodes* $i_1 = i, i_2, \ldots, i_{p-1}, i_p = j$ *such that* $W_{i_k, i_{k+1}} \neq 0$ *for* $1 \leqslant k < p$*.*

When the hypernode graph is a graph, the definition coincides with the definition of path in undirected graphs since W is equal to the adjacency matrix. Moreover, since two equivalent hypernode graphs share the same pairwise weight matrix, this notion of path is consistent with the equivalence relation. Consequently, the notion of path presented in Definition 8 satisfies the two requirements (a) and (b).

We define a *signed component* to be a maximal connected set, i.e., a maximal set of nodes such that there exists a path between any two nodes. By definition, two signed components have necessarily a null intersection. In the graph case, this definition coincides with the classic notion of connected components presented in Section 2.1.1. With these definitions, $\mathbf{h}_5$ has two signed components $C_1 = \{1, 2\}$ and $C_2 = \{3, 4\}$. Indeed, there is no path between 1 and 3, 1 and 4, 2 and 3 or 2 and 4.

As in the graph case, we now show that the indicator function of a signed component nullify the smoothness measure Ω. Consequently, we can define independently a constant label for each signed component, without modifying the global smoothness.

Proposition 14. *Let* $C_1, \ldots, C_\ell$ *be the* ℓ *signed components of a hypernode graph* $\mathbf{h} = (N, H)$*. Let* Ω *be the smoothness measure of* $\mathbf{h}$*. We have*

$$\mathrm{Span}(\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_\ell}) \subseteq \mathrm{Null}(\Omega) \ . \tag{3.4}$$

Proof. Let $C \subset N$ be a signed component. Let i be a node in S. By definition C is a maximal connected set, hence $\forall j \in N \backslash C, \;\; W_{i,j} = 0$ and we have

$$(\Delta \mathbf{1}_C)(i) = d(i) - \sum_{j \in C} W_{i,j} = d(i) - \sum_{j \in N} W_{i,j} = 0 \;\; .$$

For $i \notin C$, we can write

$$(\Delta \mathbf{1}_C)(i) = 0 - \sum_{j \in C} W_{i,j} = 0 \;\; .$$

Consequently, $\Delta \mathbf{1}_C = 0$, which concludes the proof since $\text{Null}(\Delta) = \text{Null}(\Omega)$. □

In the graph case, the inclusion in Equation (3.4) is replaced by an equality (see Proposition 2) and the indicator vector of a set of nodes nullify the smoothness measure Ω if and only if it is a connected component. In the case of hypernode graphs, one can find a set of nodes S that isn't a signed component or an union of signed components and whose indicator vector $\mathbf{1}_S$ nullify Ω. As an example, consider the hypernode graph $\mathbf{h}_7$ first introduced in Example 5 and recalled in Figure 3.6 along with its reduced signed graph $\tilde{\mathbf{g}}_7$. The indicator function $\mathbf{1}_S$ associated with the set $S = \{1, 3\}$ nullify the Laplacian Δ_7 while $\{1, 3\}$ is a strict subset of the unique signed component $\{1, 2, 3, 4\}$. In the next Section, we discuss the case of these "smooth" components.

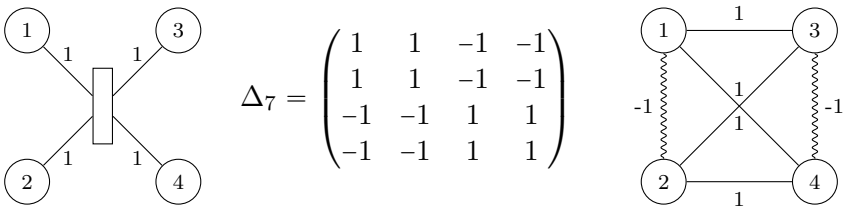


Figure 3.6: (*left*) Hypernode graph $\mathbf{h}_7$ along with (*center*) its Laplacian Δ_7 and (*right*) its reduced signed graph $\tilde{\mathbf{g}}_7$

3.2.2 Independent components and strong connectivity

Let us consider throughout this section a hypernode graph $\mathbf{h} = (N, H)$ and its Laplacian matrix Δ. As announced above, our objective is to study the sets of nodes whose indicator function nullify the smoothness measure Ω. As shown in Proposition 14, this is in particular the case for all the signed components (and all the possible unions of signed components). We show in

the following that many other sets satisfy this property. We call these sets the *independent components* of the hypernode graph.

We start by defining the notion of node *independency*. A node i in N is said to be *independent* of $S \subsetneq N$ if $i \notin S$ and if the contribution of the nodes in S to the degree of i is 0, i.e.if

$$\sum_{j \in S} W_{i,j} = 0 \ .$$

Proposition 15. *Let us consider a set of nodes $S \subsetneq N$. The indicator function $\mathbf{1}_S$ is in* Null(Ω) *if and only if the following conditions are satisfied:*

1. *All the nodes of S are independent of $\overline{S}$*
2. *All the nodes of $\overline{S}$ are independent of S*

Such a set S is called an independent component. *As a consequence, if $C_1, \ldots, C_p$ are the signed components of $\mathbf{h}$ and if $S_1, \ldots, S_m$ are the independent components of $\mathbf{h}$, we have*

$$\text{Span}(\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_p}) \subseteq \text{Span}(\mathbf{1}_{S_1}, \ldots, \mathbf{1}_{S_m}) \subseteq \text{Null}(\Omega) \ .$$

Proof. Let $\mathbf{h} = (N, H)$ be a hypernode graph and let W be its edge weight matrix. For any $i \in N$, we have

$$(\Delta \mathbf{1}_S)(i) = d(i)\mathbf{1}_S(i) - \sum_{j \neq i} W_{i,j}\mathbf{1}_S(j) \ .$$

We have $\Delta \mathbf{1}_S = 0$ if and only if for every node $i \in N$, $(\Delta \mathbf{1}_S)(i) = 0$. If $i \notin S$ then$(\Delta \mathbf{1}_S)(i) = \sum_{j \in S} W_{i,j}$. Otherwise, if $i \in S$ then

$$(\Delta \mathbf{1}_S)(i) = d(i) - \sum_{j \in S \setminus \{i\}} W_{i,j} = \sum_{j \notin S} W_{i,j} \ .$$

Therefore we have $\Delta \mathbf{1}_S = 0$ if and only if for all $i \in S$, $\sum_{j \notin S} W_{i,j} = 0$ and for all $i \notin S$, $\sum_{j \in S} W_{i,j} = 0$, which concludes the proof. □

If S is a signed component, Proposition 14 implies that $\Delta \mathbf{1}_S = 0$ so according to Proposition 15, S is also an independent component. In the graph case, a signed component is strictly equivalent to an independent component and these two notions are replaced by the unique notion of connected component. Indeed, since the pairwise weights of a graph are all positive, the independence conditions from Proposition 15 reduce to the non-existence of edges between S and $\overline{S}$. In the hypernode graph $\mathbf{h}_7$ depicted in Figure 3.6 the independent components are $C_1 = \{1, 2, 3, 4\}$, $C_2 = \{1, 4\}$, $C_3 = \{1, 3\}$, $C_4 = \{2, 3\}$ and $C_5 = \{2, 4\}$.

Note that two distinct independent components can have a non null intersection (for example $\{1, 3\}$ and $\{1, 4\}$ in the graph $\mathbf{g}_5$ shown in Figure 3.6).

This specificity plays a key role in our model and was used implicitely in the proof of Proposition 8 that allows us to claim that hypernode graphs are strictly more expressive than classic graphs.

It should be noted that, contrarily to the graph case, the relation

$$\mathrm{Span}(\mathbf{1}_{C_1}, \ldots, \mathbf{1}_{C_p}) = \mathrm{Null}(\Delta)$$

does not hold in general, even for the independent components. One can even find hypernode graphs where N is the only independent component and where $\mathrm{Span}(\mathbf{1}) \subsetneq \mathrm{Null}(\Delta)$ as shown in the following example.

Example 11. *Let us consider the hypernode graph* $\mathbf{h}_{13}$ *presented in Figure 3.7. This hypernode graph has only one independent component* $N = \{1, 2, 3\}$ *but has two null eigenvalues. Indeed, a corresponding gradient matrix is*

$$G_{13} = \begin{pmatrix} -0.5 & -0.5 & 1 \end{pmatrix} .$$

Consequently, we have $\mathrm{Rank}(\Delta_{13}) = 1$ *and* $\dim(\mathrm{Null}(\Delta_{13})) = 2$.

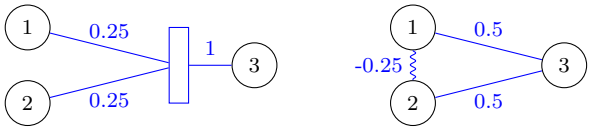


Figure 3.7: Hypernode graph $\mathbf{h}_{13}$ and its reduced signed graph $\tilde{\mathbf{g}}_{13}$

Because of Proposition 15, any hypernode graph such that $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$ will have N as unique independent component. Based on this constatation, we define the notion of *strong connectivity* in hypernode graphs.

Definition 9. *Let* $\mathbf{h} = (N, H)$ *be a hypernode graph with Laplacian matrix* Δ. *Then* $\mathbf{h}$ *is said to be* strongly connected *if* $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$.

If $\mathbf{h}$ is a strongly connected hypernode graph, then N will also be its unique signed component. Indeed, since a signed component is always an independent component, we only have to prove that $\mathbf{h}$ contains at least a connected component. This result is direct since the node singletons are always connected. The reverse does not hold as shown in Example 11.

3.3 Hypernode graph kernels and distances

One reason to use graph kernels for learning in graphs is their relation with meaningful distances on graphs. In particular, we know that the commute-time distance can be computed from the graph kernel (see Property 5 in Section 2.1). We study how these results generalize to hypernode graphs.

3.3.1 Definition and main properties

We consider in this section a hypernode graph $\mathbf{h} = (N, H)$ and its Laplacian matrix Δ. As in the graph case, we define the hypernode graph kernel to be the Moore-Penrose pseudo-inverse $\Delta^\dagger$ of the Laplacian Δ. We consider the family d^α from Section 2.1.2:

$$d^\alpha(i,j) = \left(\sqrt{\Delta^\dagger_{i,i} + \Delta^\dagger_{j,j} - 2\Delta^\dagger_{i,j}}\right)^\alpha \quad . \tag{3.5}$$

This family of functions allows to define metrics on connected graphs. In the case of hypernode graphs, only a part of the properties presented in Proposition 4 is preserved:

Proposition 16. *d^1 defines a pseudometric on $\mathbf{h}$, i.e., it is positive, symmetric and satisfies the triangle equality. Moreover, when $\mathbf{h}$ is strongly connected, d^1 becomes an actual metric (it satisfies in addition the coincidence axiom $d^1(i,j) = 0 \Rightarrow i = j$). In the general case, d^2 is positive and we have $\forall i, \quad d^2(i,i) = 0$. When $\mathbf{h}$ is strongly connected, d^2 also satisfies the coincidence axiom $d^2(i,j) \neq 0 \Rightarrow i \neq j$.*

Proof. Since $\Delta^\dagger$ is symmetric and positive semi-definite, all the properties of d^1 are similar to the graph case. In particular, when $\mathbf{h}$ is strongly connected, we have $\text{Null}(\Delta) = \text{Span}(\mathbf{1})$, which allow us to prove the coincidence axiom for d^1 (and d^2 since $d^2(i,j) = 0 \Leftrightarrow d^1(i,j) = 0$). □

Note that the properties of d^1 presented in Proposition 4 are mostly preserved in the hypernode graph case. The only difference is that the notion of connectivity is replaced by the notion of strong connectivity presented in Section 3.2.2. As a consequence, we can face new types of situations where $d^1(i,j) = 0$ while $i \neq j$. Recall that, in the graph case, this was only possible when i and j were in distinct components. In this case, the null distance between i and j was somehow artificial (no information between the two nodes). In the hypernode graph case, the situation is a bit more complex and we can find interesting configurations where $d^1(i,j) = 0$ with $i \neq j$. Let us for example consider the hypernode graph $\mathbf{h}_7$ from Example 5. We present in Figure 3.8 the distribution of d^2 around node 1 and we can notice that $d^1(1,2) = 0$. A way to interpret this specificity is to say that 1 and 2 are indistinguishable given the information contained in $\mathbf{h}_7$.

Contrary to d^1, many properties of d^2 are lost in the hypernode graph case. In particular, d^2 is not a metric, even in the strongly connected case since we lose the triangle inequality property. Moreover, there is no direct equivalent of the commute-time distance in hypernode graphs. One can prove, however,

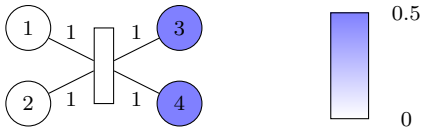


Figure 3.8: Distribution of d^1 around the node 1 in hypernode graph $\mathbf{h}_7$

that d^2 is still linked to the diffusion of information in hypernode graphs, as discussed in the next section.

3.3.2 Diffusion on hypernode graphs and relations with d^2

In this section, we consider a *strongly connected* hypernode graph $\mathbf{h}$ and its Laplacian matrix Δ. Our objective is to express d^2 in terms of a diffusion function in hypernode graphs mimicking the graph case. For this, we first consider the Poisson equation $\Delta f = \mathbf{In}$ that models the diffusion of an input charge $\mathbf{In}$ through a system associated with the hypernode graph Laplacian Δ. Let us consider a node $j \in N$ called *sink* node, we consider the input function $\mathbf{In}_j$ defined by

$$\mathbf{In}_j(i) = \begin{cases} d(i) \text{ if } i \neq j \ , \\ d(j) - \text{Vol}(\mathbf{h}) \text{ if } i = j \ , \end{cases} \tag{3.6}$$

where $d(i)$ denotes the degree of the node i. We define the set $\mathcal{S}_j$ as the set of functions $f \in \mathbb{R}^n$ which are solutions of the equation $\Delta f = \mathbf{In}_j$. For every pair of nodes (k, ℓ) in N and for some $f \in S_j$, we define $\mathrm{V}_j(k, \ell) = f(k) - f(\ell)$, i.e., $\mathrm{V}_j(k, \ell)$ is the difference of potential between k and ℓ. However, to make the definition of V_j consistent, we first have to prove that it does not depend on the choice of a solution f in the set $\mathcal{S}_j$.

Lemma 1. *For every sink node j, the solutions of $\Delta f = \mathbf{In}_j$ are the functions $f = \mu\mathbf{1} + \Delta^\dagger \mathbf{In}_j$ where $\mu \in \mathbb{R}$.*

Proof. Since $\mathbf{g}$ is strongly connected, we have $\text{Null}(\Delta) = \text{Span}(\mathbf{1})$. Therefore, $\mathbb{R}^n$ is the direct sum of the space $\text{Span}(\mathbf{1})$ and of the space $\text{Null}(\Delta)^\perp$. Since $f \in \mathbb{R}^n$, it can be written $f = \mu\mathbf{1} + g$ where $g \in \text{Null}(\Delta)^\perp$. Thus, we have $\Delta f = \Delta(\mu\mathbf{1} + g) = \Delta g$.

Let us suppose that f satisfies $\Delta f = \mathbf{In}_j$, we deduce that $\Delta g = \mathbf{In}_j$. As seen above, we know that the operator $\Delta^\dagger\Delta$ is the orthogonal projector operator on $\text{Null}(\Delta)^\perp$. Since $g \in \text{Null}(\Delta)^\perp$, we have $g = \Delta^\dagger \Delta g = \Delta^\dagger \mathbf{In}_j$. Hence we can write f under the form $\mu\mathbf{1} + \Delta^\dagger \mathbf{In}_j$.

Conversely, let us consider $f = \mu\mathbf{1} + \Delta^\dagger \mathbf{In}_j$ with $\mu \in \mathbb{R}$. We have $\Delta f = \Delta\left(\mu\mathbf{1} + \Delta^\dagger \mathbf{In}_j\right) = \mu\Delta\mathbf{1} + \Delta\Delta^\dagger \mathbf{In}_j$. From the definition of $\mathbf{In}_j$, we deduce

that $\mathbf{In}_j \in \mathrm{Null}(\Delta)^\perp$. Since Δ is symmetric, $\Delta\Delta^\dagger$ is also the orthogonal projector on $\mathrm{Null}(\Delta)^\perp$ and thus $\Delta\Delta^\dagger\mathbf{In}_j = \mathbf{In}_j$. Since $\Delta\mathbf{1} = 0$, we get $\Delta f = \mathbf{In}_j$ which concludes the proof. □

As a consequence, $\mathrm{V}_j(k,\ell)$ does not depend on the choice of f in the set of the solutions of $\Delta f = \mathbf{In}_j$. This allows us to write

$$\mathrm{V}_j(k,\ell) = (\mathbf{e}_k - \mathbf{e}_\ell)^T \Delta^\dagger \mathbf{In}_j \ ,$$

and we can give the main proposition that establish a link between d^2 and the difference of potential V.

Proposition 17. *For every i, j in N, we have* $\mathrm{Vol}(\mathbf{h})d^2(i,j) = \mathrm{V}_j(i,j) + \mathrm{V}_i(j,i)$*, and* $\mathrm{V}_j(i,j)$ *satisfies*

$$\begin{cases} \mathrm{V}_j(i,j) = \displaystyle\sum_{h|i\in h} \frac{w_h(i)}{d(i)} \left[1 + \sum_{k\in h, k\neq i} P(h,i,k) \sqrt{\frac{w_h(k)}{w_h(i)}} \mathrm{V}_j(k,j) \right] & \text{if } i \neq j \ , \\ \mathrm{V}_j(i,i) = 0 \ . \end{cases} \tag{3.7}$$

Proof. First, we show that $\Omega(i,j) = \frac{\mathrm{V}_j(i,j)+\mathrm{V}_i(j,i)}{\mathrm{Vol}(\mathbf{h})}$. For that, let us develop the expression of $\mathrm{V}_j(i,j)$ obtained above

$$\begin{aligned} \mathrm{V}_j(i,j) =& (\mathbf{e}_i - \mathbf{e}_j)^T \Delta^\dagger \mathbf{In}_j \\ =& \sum_{k\neq j} d(k)(\Delta^\dagger_{k,i} - \Delta^\dagger_{k,j}) - (\mathrm{Vol}(\mathbf{h}) - d(j))(\Delta^\dagger_{i,j} - \Delta^\dagger_{j,j}) \\ =& \sum_{k\neq i,j} d(k)(\Delta^\dagger_{k,i} - \Delta^\dagger_{k,j}) + d(i)(\Delta^\dagger_{i,i} - \Delta^\dagger_{i,j}) + (\mathrm{Vol}(\mathbf{h}) - d(j))(\Delta^\dagger_{j,j} - \Delta^\dagger_{i,j}) \\ =& \mathrm{Vol}(\mathbf{h})(\Delta^\dagger_{j,j} - \Delta^\dagger_{i,j}) + R(j,i) \ , \end{aligned}$$

where

$$R_j(j,i) = \sum_{k\neq i,j} d(k)(\Delta^\dagger_{k,i} - \Delta^\dagger_{k,j}) + d(i)(\Delta^\dagger_{i,i} - \Delta^\dagger_{i,j}) - d(j)(\Delta^\dagger_{j,j} - \Delta^\dagger_{i,j}) \ .$$

We can observe that, for every node i and j in N, we have $R_i(i,j)+R_j(j,i) = 0$. Thus, we can write

$$\begin{aligned} \mathrm{V}_j(i,j) + \mathrm{V}_i(j,i) &= \mathrm{Vol}(\mathbf{h})(\Delta^\dagger_{i,i} + \Delta^\dagger_{j,j} - \Delta^\dagger_{i,j} - \Delta^\dagger_{j,i}) \\ &= \mathrm{Vol}(\mathbf{h})(\Delta^\dagger_{i,i} + \Delta^\dagger_{j,j} - 2\Delta^\dagger_{i,j}) \\ &= \mathrm{Vol}(\mathbf{h})d^2(i,j) \ , \end{aligned}$$

which concludes the first part of the proof.

It remains to show that V_j satisfies Equation (3.7). By definition of V_j, we get that, for every node i, $\mathrm{V}_j(i,i) = 0$. Let us now consider $i \neq j$ and let us consider f in $\mathcal{S}_j$, i.e., a solution of $\Delta f = \mathbf{In}_j$. As $i \neq j$, we have $d(i) = \mathbf{e}_i^T \mathbf{In}_j$. Since $f \in \mathcal{S}_j$, we have $\mathbf{In}_j = \Delta f = G^T G f$ so we can rewrite the previous equality as $d(i) = \mathbf{e}_i^T G^T G f = (G\mathbf{e}_i)^T (Gf)$. Now, because of the Equilibrium Condition, we have $G\mathbf{1} = \mathbf{0}$, thus $Gf = G(f - f(j)\mathbf{1})$. This leads to

$$
\begin{aligned}
d(i) &= (G\mathbf{e}_i)^T (G(f - f(j)\mathbf{1}) \\
&= \sum_h (G\mathbf{e}_i)(h) \cdot (G(f - f(j)\mathbf{1})(h) \\
&= \sum_h \left(\sqrt{w_h(i)} \epsilon_h(i) \right) \cdot \left(\sum_{k \in h} \sqrt{w_h(k)} \epsilon_h(k) (f(k) - f(j)) \right) \quad \text{(see Equation (2.8))} \\
&= \sum_{h | i \in h} \sqrt{w_h(i)} \left\{ \sum_{k \in h} (-P(h,k,i)) \sqrt{w_h(k)} (f(k) - f(j)) \right\} \quad \text{(see Equation (2.7))} \\
&= \sum_{h | i \in h} \sqrt{w_h(i)} \left\{ \sum_{k \in h} (-P(h,k,i)) \sqrt{w_h(k)} \mathrm{V}_j(k,j) \right\} \\
&= \mathrm{V}_j(i,j) \sum_{h | i \in h} w_h(i)(-P(h,i,i)) + \sum_{h | i \in h} \sqrt{w_h(i)} \left\{ \sum_{k \in h, k \neq i} (-P(h,k,i)) \sqrt{w_h(k)} \mathrm{V}(k,j) \right\} .
\end{aligned}
$$

We have for all i, $P(h,i,i) = -1$ and $\sum_h w_h(i) = d(i)$ so the previous equality can be rewritten under the form

$$
d(i) = \mathrm{V}_j(i,j) d(i) + \sum_{h | i \in h} \sqrt{w_h(i)} \left\{ \sum_{k \in h, k \neq i} (-P(h,k,i)) \sqrt{w_h(k)} \mathrm{V}_j(k,j) \right\} .
$$

Hence, we get the linear system

$$
\begin{aligned}
\mathrm{V}_j(i,j) &= 1 + \sum_{h | i \in h} \frac{\sqrt{w_h(i)}}{d(i)} \left\{ \sum_{k \in h, k \neq i} P(h,k,i) \sqrt{w_h(k)} \mathrm{V}_j(k,j) \right\} \\
&= \sum_{h | i \in h} \frac{w_h(i)}{d(i)} \left\{ 1 + \sum_{k \in h, k \neq i} P(h,k,i) \sqrt{\frac{w_h(k)}{w_h(i)}} \mathrm{V}_j(k,j) \right\} ,
\end{aligned}
$$

which concludes the proof. □

It should be noted that the above proof generalizes the classic proof for graphs based on electrical equivalence. Indeed, the proof from Chandra et al. (1996) considers an electrical network where each edge of the original graph is replaced by a one Ohm resistor. He shows that, when we inject $d(r)$ unit of current in each node r and remove an equivalent quantity from a specific sink node j, the difference of potential between a random node i and the sink node j is proportional to the hitting-time distance from i to j. Please note that this definition of the input current is equivalent to our input function **In**. Such a system can be seen as a density of charge that diffuses through an electrical network. Chandra et al. (1996) leverages the classic laws of electrostatic to solve this problem (Ohm's law and Kirchoff's law) but from a more general perspective, the diffusion of a charge in a continuous system can be described by Poisson's equation for electrostatics

$$\Delta V = \frac{-\rho}{\epsilon} \ ,$$

where V is the electric potential, ρ describes the charges brought from outside (free charge density) and ϵ is a constant depending on the material. This equation is similar to our diffusion equation $\Delta f = \mathbf{In}$ that links an input function **In** with a function f which can be seen as the potential function of the system. Thus, our definition of $\mathrm{V}_j(i,j) = f(i) - f(j)$ is compliant with the one of Chandra et al. (1996) since it denotes the difference of potential between a node i and the system sink node j.

It should be noted that, in Proposition 17, the quantity $\mathrm{V}_j(i,j) + \mathrm{V}_i(j,i)$ is equal to $(\mathrm{V}_j - \mathrm{V}_i)(i,j)$. The function $\mathrm{V}_i^j = \mathrm{V}_j - \mathrm{V}_i$ can be seen as the potential function associated with the input function $\mathbf{In}_j - \mathbf{In}_i = \mathrm{Vol}(\mathbf{h})(e_i - e_j)$ (current entering by a source node i and exiting by a sink node j). By a proof similar to Lemma 1, we can show that

$$\mathrm{V}_i^j(k,\ell) = (e_k - e_\ell)\Delta^\dagger(\mathbf{In}_j - \mathbf{In}_i) = \mathrm{Vol}(\mathbf{h})(e_k - e_\ell)\Delta^\dagger(e_i - e_j) \ ,$$

which is consistent with the expression of $\Omega(i,j)$.

When the hypernode graph is a graph, all the hyperedges h which contain i are simple edges $\{\{i\},\{k\}\}$ with $k \in N$. In Equation (3.7), $w_h(i)$ and $w_h(k)$ reduces to $W_{i,k}$ and the linear system reduces to

$$\begin{cases} \mathrm{V}_j(i,j) = \displaystyle\sum_{k \in N} \frac{W_{i,k}}{d(i)}(1 + \mathrm{V}_j(k,j)) & \text{if } i \neq j \ , \\ \mathrm{V}_j(i,i) = 0 \ . \end{cases}$$

Consequently, $\mathrm{V}_j(i,j)$ can be interpreted as the hitting-time distance from i to j (average number of steps needed by a random walker to travel from i to

j). Therefore, the metric $d^2(i,j)$ coincides with the commute-time distance divided by the overall volume in the case of graphs (see also Klein and Randić 1993; Chandra et al. 1996; Fouss et al. 2007).

In the general hypernode graph case, the situation is more elaborated. Indeed, let us define $p(h|i) = \frac{w_h(i)}{d(i)}$ and $p(k|h,i) = P(h,i,k)\sqrt{\frac{w_h(k)}{w_h(i)}}$. Then, we can rewrite the Equation (3.7) as

$$\begin{cases} \mathrm{V}_j(i,j) = \sum_{h|i\in h} p(h|i)\left[1 + \sum_{k\in h, k\neq i} p(k|h,i)\mathrm{V}_j(k,j)\right] & \text{if } i \neq j \ , \\ \mathrm{V}_j(i,i) = 0 \ . \end{cases}$$

Notice that $p(h|i)$ is non-negative and that $\sum_h p(h|i) = 1$. Thus, $p(h|i)$ can be interpreted as a jumping probability from i to the hyperedge h. We also have $\sum_n p(k|h,i) = 1$ but $p(k|h,i)$ is negative as soon as i and k belong to the same end of h. This prevents us from interpreting this quantity as a jumping probability from i to k by h. Therefore, in the general case, we do not have a random walk interpretation of the d^2. Notice however that several theoretical approaches coming from the world of quantum physics have been built to take into consideration quantities like $p(k|h,i)$. (See for example Burgin 2010).

3.3.3 The transition matrix $P = D^{-1}W$

Another way to apprehend the semimetric d^2 in strongly connected hypernode graphs is to rewrite Equation (3.7) as

$$\begin{aligned} \mathrm{V}_j(i,j) &= 1 + \sum_{h|i\in h} \sum_{\substack{k\in h \\ k\neq i}} \frac{W_{i,k}}{d(i)} \mathrm{V}_j(k,j) \\ &= 1 + \sum_{k\in N} P_{i,k}\mathrm{V}_j(k,j) \ , \end{aligned}$$

where W is the pairwise weight matrix of $\mathbf{h}$ and P is the *transition matrix* defined by $P = D^{-1}W$. The rows of P sums to 1 but P is not a stochastic matrix since it can contain negative values. However, P can still be seen as a transition matrix associated with the reduced signed graph of $\mathbf{h}$.

Let us now briefly discuss the existence of a stationary distribution based on the transition matrix P. In the graph case, the transition matrix P is a stochastic matrix. For connected graphs we can apply Perron-Frobenius Theorem to show the uniqueness of a stationary distribution of nodes. In the hypernode graph case, this is not possible in general but we can still exhibit a stationary distribution based on the degrees.

Proposition 18. *Let* $\mathbf{h} = (N, H)$ *be a hypernode graph with transition matrix* P*. The vector* $\pi = \frac{1}{\mathrm{Vol}(\mathbf{h})}(d(0), \ldots, d(n))$ *is a probability vector and satisfies*

$$\pi P = \pi \quad .$$

Proof. It is easy to see that $\sum_{i \in N} \pi_i = 1$ since $\mathrm{Vol}(\mathbf{h}) = \sum_{i \in N} d(i)$. Let us consider a node $i \in N$. From the definition of degrees, $d(i) > 0$, thus $\pi_i > 0$. So π is a probability vector and:

$$(\pi P)_i = \sum_{k \in N} \pi_k P_{k,i} = \frac{1}{\mathrm{Vol}(\mathbf{h})} \sum_{k \in N} d(k) \frac{W_{k,i}}{d(k)} = \frac{1}{\mathrm{Vol}(\mathbf{h})} d(i) = \pi_i \quad .$$

□

3.4 Conclusion

In this chapter, we have highlighted important properties of hypernode graphs and shown how the concepts defined on graphs can be extended to our new framework. We have defined the notion of reduced signed graph that allows us to build bridges between the theory of hypernode graphs and the theory of signed graphs. We have also addressed the question of connectivity in hypernode graphs and defined the notion of signed components and independent components. Following this, we have discussed the notion of metrics on hypernode graphs and studied the properties of the family d^α first introduced in Section 2.1.2. We have especially pointed out a very general relation between the family d^α and the notion of diffusion defined by the matrix $P = D^{-1}W$.

In the next section, we put our framework into practice and propose a new algorithm based on hypernode graphs that solves the skill rating problem in multiplayer games.

Chapter 4

Skill rating with hypernode graphs

Chapter abstract In this chapter, we apply the theory of hypernode graphs to the problem of skill rating in multiplayer games. We use hypernode graphs to represent games between teams of players and learn the individual skills from the games outcomes. Then, we use the learnt skill values in order to predict the outcomes of new games. We show that our method outperform specialized algorithms such as Elo or Trueskill for different datasets in the batch setting.

In the first part of the chapter, we introduce definitions and notations concerning the skill rating problem and describe specialized algorithms that can be used to solve this problem. Then, in a second part, we explain how we can encode games using hypernode graphs and present our algorithm. Finally, we present some experimental results and discuss the performance of the different algorithms.

4.1 Skill rating in multiplayer games

Throughout this chapter, we consider a set of individual players $P = \{1, \dots, n\}$ and a set of games $\Gamma = \{\gamma_1, \dots, \gamma_p\}$. Every game is played with the same rules by two teams of players. The result of a game may be a win for one of the two teams playing or a draw. Skill rating methods aim to estimate the skill ratings of the different players using the outcomes of the games. Skill rating techniques are extensively used in competitive online games such as Starcraft II or League Of Legend in order to propose equilibrated games

and, thus, improve player experience. In some cases, skill rating can also be used to define ranks and classes of players. In chess games for example, the title *Grandmaster* is awarded to players that have attained a specific rating.

In the following sections, we introduce the notion of team additive model that was notably used in Herbrich et al. (2006). Then, we present two popular algorithms that solve the problem for different settings: Elo (and its "duelling" extension) (Elo 1978) and TrueSkill (Herbrich et al. 2006).

4.1.1 Notations and team additive model

Let us consider a game $\gamma_j \in \Gamma$ and a player $i \in P$ involved in this game. We assume that the contribution of player i to its team during the game can be summarized into a non negative weight $c_j(i)$. We denote by $s(i)$ the *skill rating* vector of player i. The *performance* of player i in the game γ_j is a real value denoted by $x_j(i)$ and is directly related to $s(i)$ using one of the following model:

- The *deterministic model* where $s(i) \in \mathbb{R}$ and $x_j(i) = s(i)$
- The *probabilistic model* where $x_j(i)$ is drawn from a probability distribution parametrized by the skill vector $s(i)$.

We assume that the outcome of γ_j can be predicted by comparing the weighted sum of the performances of the players of each of the two teams. Given two teams of players $A = \{a_1, a_2, \ldots, a_\ell\}$ and $B = \{b_1, b_2, \ldots, b_k\}$ playing game γ_j, then A is predicted to be the winner if

$$\sum_{i=1}^{\ell} c_j(a_i)x_j(a_i) > \sum_{i=1}^{k} c_j(b_i)x_j(b_i) \ . \qquad (\mathcal{C}_j)$$

Said differently, we assume that the performance of a team is a weighted sum of the individual performances of the players. This simple linear model is called *team additive model* and was notably used in the TrueSkilll algorithm (Herbrich et al. 2006). The contribution $c_j(i)$ can depend on the general rules of the games (*a priori* knowledge such as asymmetricity of roles in a team) or on the features of the specific game γ_j (asymmetricity brought by the game conditions for example). This model is limited when it comes to capture the complexity of the interactions between players (such as complementarity) but is a fair approximation for the skill rating problem. Let us give a few more details about the *deterministic* and *probabilistic* model.

The deterministic skill model

In this model, we assume that $s(i) \in \mathbb{R}$ and $x_j(i) = s(i)$. Thus, the performance of a player is independent on the game and we can rewrite inequality ($\mathcal{C}_j$) as

$$\sum_{i=1}^{\ell} c_j(a_i)s(a_i) > \sum_{i=1}^{k} c_j(b_i)s(b_i) \ .$$

We can turn this equation into an equality by introducing a non negative real number o_j on the right hand that quantifies the game outcome:

$$\sum_{i=1}^{\ell} c_j(a_i)s(a_i) = o_j + \sum_{i=1}^{k} c_j(b_i)s(b_i) \ . \qquad (\mathcal{C}'_j)$$

In the case of a draw, the game outcome o_j is set to 0. In order to be consistent with the game γ_j, the skill s must satisfy the equality ($\mathcal{C}'_j$). However, for a given a set of games $\Gamma = \{\gamma_1, \ldots, \gamma_p\}$, it may be impossible to satisfy simultaneously all the constraints $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_p\}$. Our goal is then to learn a function s that respects the game constraints as much as possible. More formally, we define the error vector $\varepsilon(s) \in \mathbb{R}^p$ by

$$\varepsilon(s)_j = \left| \sum_{i=1}^{\ell} c_j(a_i)s(a_i) - o_j - \sum_{i=1}^{k} c_j(b_i)s(b_i) \right| \ .$$

$\varepsilon(s)_j$ quantifies the consistency of s with the j-th game γ_j. Our goal is to find a skill rating function s^* that minimizes the norm of ε, i.e. search for

$$s^* = \arg\min_s \|\varepsilon(s)\| \qquad (4.1)$$

The choice of the norm $\|\cdot\|$ will directly impact on the properties of our optimal function s^*. For instance, using the L1-norm will result in a skill rating function s that neglects a minority of games in order to better satisfy the majority, leading to a sparse residual error $\varepsilon(s^*)$. On the contrary, the L2-norm will tend to smooth the residual error $\varepsilon(s^*)$.

The probabilistic skill model

In this model, we define the performances to be drawn from $\mathcal{D}_{s(i)}$ where $(\mathcal{D}_\theta)_{\theta \in \mathbb{R}}$ is a given parametric family. Again, the performance $x_j(i)$ does not depend on the potential features of the game γ_j but should rather be seen

as a random variable X_i. This model is mainly used in the Bayesian rating methods such as Elo or TrueSkill.

In what follows, we introduce a new skill rating method that is based on the deterministic model and that leverages our hypernode graph framework. We compare the performance of our algorithm with the abovementioned methods Elo and TrueSkill. For this reason, we first review briefly these two algorithms.

4.1.2 The Elo rating system

The Elo rating system is an *online* method proposed by Elo (1978) in order to estimate the skills of chess players in international competitions. Even today, it remains one of the most popular ranking system because of its efficiency and simplicity. The original system was designed for single player games but some ad-hoc extensions have been proposed for the case of multiplayer games.

In Elo, the skill rating of a player $i \in P$ is represented by a real value $s(i) \in \mathbb{R}$. The performance of a player in a game is assumed to be a random variable X_i drawn from a symmetrical distribution centered in $s(i)$ with fixed width. The original Elo system is based on normal distributions but some more recent versions of the algorithm use logistic distributions[1]. For any pair of players (i, j), the random variable $X_j - X_i$ represents the difference of performance between player i and player j in a game. The probability p_i for i to win a game against j is given by

$$p_i = \mathbb{P}\left[X_j - X_i < 0\right] \quad .$$

We illustrate the computation of p_i and p_j in Example 12 below.

Example 12. *Let us consider the case of normal distributions and let i and j be two players such that*

$$X_i \sim \mathcal{N}(1, 1) \quad \textit{and} \quad X_j \sim \mathcal{N}(2, 1),$$

($s(i) = 1$ and $s(j) = 2$) as presented in Figure 4.1 (left). If X_i and X_j are independant, $X_j - X_i \sim \mathcal{N}(1, 2)$ so the probability for i to win a game against j is $p_i = 0.16$. Conversely, we have $p_j = 1 - p_i = 0.84$.

Let us now assume that we observe a game between i and j where i defeats j. Elo proposes to update the skills $s(i)$ and $s(j)$ by considering the posterior

[1] This is for instance the case of the implementation used by the United States Chess Federation.

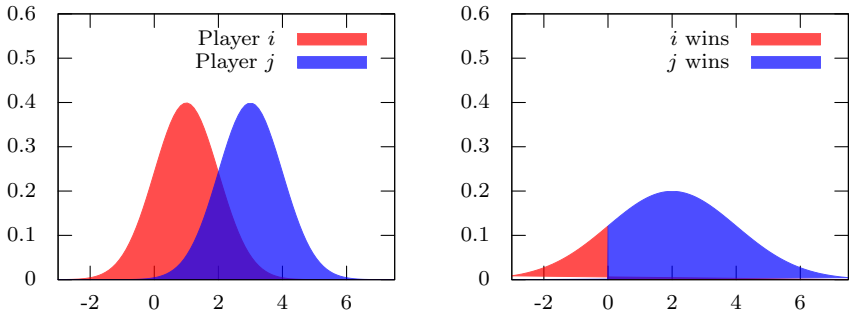


Figure 4.1: *[Left]* Individual performances X_i and X_j *[Right]* Difference of performances $X_j - X_i$

probabilities $p_i^{post} = 1$ and $p_j^{post} = 0$. The update formula is defined as the simple descent

$$s(i) \leftarrow s(i) + K_i(p_i^{post} - p_i)$$
$$s(j) \leftarrow s(j) + K_j(p_j^{post} - p_j) \ ,$$

where K_i and K_j are in $\mathbb{R}^+$. K_i is the *K-factor associated with player* i and allows to fix the convergence rate for the skill $s(i)$. If we observe a draw game between players i and j, the posterior probabilities p_i^{post} and p_j^{post} to be equal to 0.5.

When the system is not confident about the value $s(i)$, a good option is to use a big K-factor K_i, which allows important updates of the skill. This is for instance the case of the newcomers that are assigned a default value. Conversely, a smaller K-factor allows a finer tuning of the skill $s(i)$ and is a good choice when the confidence in the current value is high. Note that the estimation of the confidence is not part of the Elo algorithm and, therefore, remains a tricky point of this approach. A simple heuristic used in official chess rankings is to consider that the higher the skill, the more confident we are about it. Indeed, good ratings are usually obtained after a high number of played games, which reduces the uncertainty. For instance, the world chess federation (FIDE) uses a three-stage strategy: $K = 40$ for the newcomers (first 30 games), $K = 20$ for the "standard" players ($s < 2400$) and $K = 10$ for the "master" players ($s > 2400$).

The duelling extension for multiplayer games

The Elo rating system is not designed to handle directly multiplayer games. Indeed, the skill update rule is limited to the one vs one case. The main difficulty of the multiplayer case is to formulate an individual update rule

based on team information. We briefly present an extension based on the *duelling* paradigm that allow us to circumvent this limitation. The main idea is to see a game between two teams of respectively ℓ and k players as a combination of $\ell * k$ *duels* between two players.

To do so, we consider all the possible pairs of players (i, j), where i belongs to the first team and j to the second team. Each of these pairs is a duel and we define the outcome of all duels to be the outcome of the original game (if the first team wins then i should win his duel against j). For each duel, we can apply the original Elo algorithm and compute the variation of skills for the two players. Finally, we upgrade the skill of each player by considering the average of the duel variations.

Example 13. *As an example, let us consider the simple game presented in Figure 4.2. We assume that team* $\{1, 2\}$ *defeats team* $\{3, 4\}$*. We denote by* $\delta_{1>3}$ *(resp.* $\delta_{1>4}$*) the variation of skill of* 1 *in the duel* $(1, 3)$ *(resp.* $(1, 4)$*). The final skill update for* 1 *is* $s(1) \leftarrow s(1) + \frac{(\delta_{1>3} + \delta_{1>4})}{2}$.

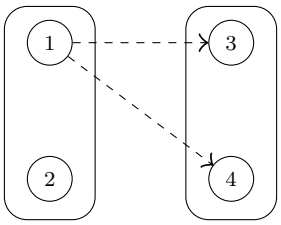


Figure 4.2: Game between $\{1, 2\}$ and $\{3, 4\}$. The *duels* of player 1 are $(1, 3)$ and $(1, 4)$.

4.1.3 The TrueSkill rating system

TrueSkill algorithm is an online algorithm introduced by Herbrich et al. (2006). It allows to solve two main limitations of Elo:

1. the estimation of the confidence, which is now part of the player rating
2. the handling of multiplayer games through the usage of the additive team model presented in Section 4.1.1 (note that it also allows to handle more than two teams in a game but we won't focus on this case in our work)

As in the Elo framework, the idea is to express the outcome of a game as a random variable depending on the skill rating s. The rating of a player $i \in P$ is encoded through a pair of values $s(i) = (\mu_i, \sigma_i) \in \mathbb{R}^2$, which defines a gaussian distribution. The instant skill S_i of player i in a game is then defined as a random variable drawn from this distribution. The variable

width σ_i allows to keep trace of the uncertainty of the current skill value μ_i, as shown in Figure 4.3.

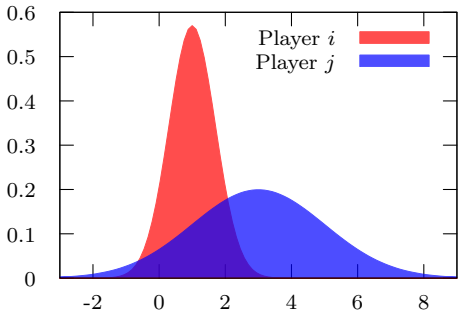


Figure 4.3: Distributions of S_i and S_j for $i, j \in P$

The performance of player i is then given by the random variable $X_i \sim \mathcal{N}(S_i, \beta^2)$ where $\beta \in \mathbb{R}^+$. The total performance T_I of a team $I \subset P$ is then defined using an additive model with unitary contributions:

$$T_I = \sum_{i \in I} X_i \ .$$

Finally, the difference of performance $T_J - T_I$ between two teams I and J is then directly related to the outcome of the game (a team is better than another if its total performance exceeds the performance of the other one). All this construction leads to a complex belief network that can be used to infer the ratings $(s(i))_{i \in P}$ based on the observed outcomes.

Example 14. *Let us consider a set of three players $\mathcal{P} = \{1, 2, 3\}$ and a single game between the teams $\{1\}$ and $\{2, 3\}$. The factor graph built by Trueskill is depicted in Figure 4.4.*

4.2 Learning skill ratings with hypernode graphs

In this section, we describe our contribution to the problem of skill rating in multiplayer games. We consider the deterministic model described in Section 4.1.1 and show that the minimization problem (4.1) instantiated with the Euclidean norm (L2-norm) can be turned into a semi-supervised learning problem on a hypernode graph.

4.2.1 Modeling Games with Hypernode Graphs

We model games using hypernode graphs and infer the skill ratings. We use the deterministic skill model as presented in Section 4.1.1. Each game

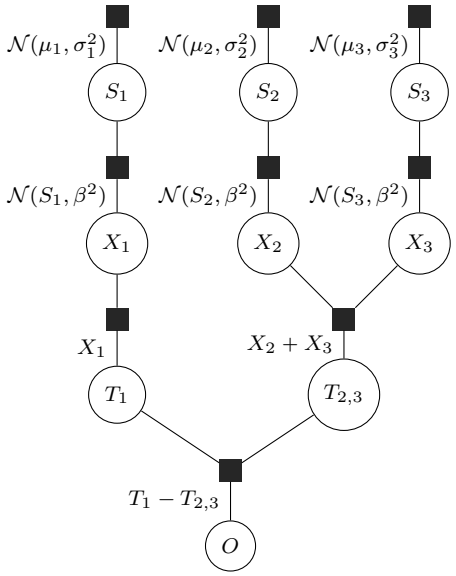


Figure 4.4: Trueskill factor graph for a single game between the player 1 and a team formed with the players 2 and 3

will be represented by a hyperedge with specific nodes related to the game outcomes. We introduce the general construction by considering an example. Let us consider a game γ between two teams $A = \{1, 2\}$ and $B = \{3, 4\}$. Let us also assume that all the players contribute to the game with the same weight $c(1) = c(2) = c(3) = c(4) = 1$. Note that using uniform weights implies that the roles of the players inside a team are interchangeable and that uniform skills lead to draw games. Such a game can be modeled by a hyperedge between sets of nodes $\{1, 2\}$ and $\{3, 4\}$ with weights equal to 1.

Now, let us suppose that A wins the game, then the skill rating function s must satisfy Equation ($\mathcal{C}'_j$), which reduces in this simple case to

$$s(1) + s(2) = o + s(3) + s(4) \ , \qquad (\mathcal{C}')$$

where $o > 0$ represents the outcome of the game γ. In order to model the game outcome in the hyperedge, we introduce a new node H called *outcome node*. H is added to the hypernode $\{3, 4\}$ and has a weight equal to 1. It can be regarded as a virtual player that plays along with team B and whose skill is fixed to $s(H) = o > 0$. Last, for the hyperedge to satisfy the equilibrium condition, we add a node Z called *lazy node* to the hypernode $\{1, 2\}$ and set its weight to 1. The skill $s(Z)$ of the lazy node Z is fixed to be 0 such as the equation ($\mathcal{C}'$) can be rewritten as

$$s(1) + s(2) + s(Z) = s(3) + s(4) + s(H) \ ,$$

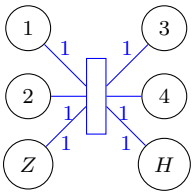


Figure 4.5: Hyperedge h for a game γ between team $A = \{1, 2\}$ and $B = \{3, 4\}$. We assume that team A wins and add an outcome node H and a lazy node Z. The node contributions are set to 1.

where s satisfies $s(H) = o$ and $s(Z) = 0$. Similarly to H, Z can be seen as a virtual player that plays along with team A (the only difference is that Z is useless and does not help his team). Note that this equation is the definition of the *smoothness* of a node real valued function s over the hyperedge h with $s_h = \{1, 2, Z\}$ and $t_h = \{3, 4, H\}$ as depicted in Figure 4.5.

For each game $\gamma_j \in \Gamma$, let us denote by A_j and B_j the two teams involved. The winning team is known as well as the game outcome o_j. We can define, for every game γ_j a hyperedge h_j as follows

1. The players of the first team A_j define one of the two hypernodes of h_j. The weight of a player node i is defined to be the square of the player contribution $c_j(i)^2$,
2. do the same construction for the second team B_j,
3. add a *outcome node* H_j to the set of player nodes corresponding to the losing team. Its weight is set to 1,
4. add a *lazy node* Z_j to the set of player nodes corresponding to the winning team. Its weight is chosen in order to ensure the Equilibrium condition for the hyperedge h.

We follow the above procedure for all the games contained in Γ and build a hypernode graph $h = (N, H)$. Now, the skill rating functions correspond to the real-valued nodes functions over the hypernode graph. In order to model the game outcomes in the computation of the player skills, we fix the skill rating function values over the additional nodes for the virtual players. A skill rating function s over $\mathbf{h}$ must thus satisfy, for every lazy node Z_j, $s(Z_j) = 0$, and, for every outcome node H_j, $s(H_j) = o_j$ (outcome value of the corresponding game γ_j).

Formally, we assume that the first n nodes in N are the player nodes followed by the t_ℓ lazy nodes, then followed by the t_o outcome nodes. Namely,

$$N = \{1, \ldots, n\} \cup \{n + 1, \ldots, n + t_\ell\} \cup \{n + t_\ell + 1, \ldots, n + t_\ell + t_o\} \quad (4.2)$$

In the following, we denote by $m = n + t_\ell + t_o$ the total number of nodes. Let Δ be the unnormalized Laplacian of $\mathbf{h}$, and let s be a real-valued node

function on h, s can be seen as a real vector in $\mathbb{R}^m$ where the first n entries represent the skills of the n players. Then, it is easy to show that the *skill rating problem* (4.1) is equivalent to find the optimal vector s solving the optimization problem

$$\begin{aligned} \underset{s\in\mathbb{R}^N}{\text{minimize}} \quad & s^T\Delta s \\ \text{subject to} \quad & \forall n+1 \leqslant j \leqslant n+t_\ell,\ s(j)=0 \text{ (for lazy nodes)} \\ & \forall n+t_\ell+1 \leqslant j \leqslant n+t_\ell+t_o,\ s(j)=o_j \text{ (for outcome nodes)} \end{aligned} \tag{4.3}$$

4.2.2 Regularizing the hypernode graph

When the number of games is small, many players participate at most in one game. Thus, in this case, the number of signed components can be quite large. The player skills in each signed component can be defined independently while satisfying the constraints. Thus, it will be irrelevant to compare player skills in different signed components. In order to solve this issue, we introduce in Equation (4.3) a regularization term based on the standard deviation of the players skills $\sigma(s_p)$, where $s_p = (s(1),\dots,s(n))$. This leads to the new formulation

$$\begin{aligned} \underset{s\in\mathbb{R}^N}{\text{minimize}} \quad & s^T\Delta s + \mu\sigma(s_p)^2 \\ \text{subject to} \quad & \forall n+1 \leqslant j \leqslant n+t_\ell,\ s(j)=0 \text{ (lazy nodes)} \\ & \forall n+t_\ell+1 \leqslant j \leqslant n+t_\ell+t_o,\ s(j)=o_j \text{ (outcome nodes)}, \end{aligned} \tag{4.4}$$

where μ is a regularization parameter. Thus, we control the spread of s_p, avoiding to have extreme values for players participating in a small number of games.

In order to apply graph-based semi-supervised learning algorithms using hypernode graph Laplacians, we now show that the regularized optimization problem can be rewritten as an optimization problem for some hypernode graph Laplacian. For this, we show that it suffices to add a regularizer node in the hypernode graph $\mathbf{h}$. First, let us recall that if $\overline{s}$ is the mean of the player skills vector $s_p = (s(0),\dots,s(n))$, then, for all $q \in \mathbb{R}$, we have

$$\sigma(s_p)^2 = \frac{1}{n}\sum_{i=1}^{n}(s(i)-\overline{s})^2 \leqslant \frac{1}{n}\sum_{i=1}^{n}(s(i)-q)^2 \ .$$

Thus, in the problem 4.4, we can instead minimize $s^T\Delta s + \frac{\mu}{n}\sum_{i=1}^{n}(s(i)-q)^2$ over s and q. We now show that this is also equivalent to minimize $r^T\Delta_\mu r$ for

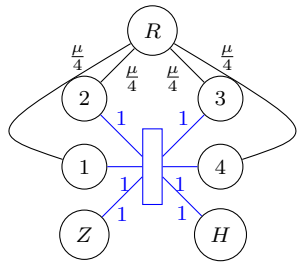


Figure 4.6: Adding a regularizer node R to the hypergraph presented in Figure 4.5

some vector r and well chosen hypernode graph Laplacian Δ_μ. For this, let us consider the $p \times m$ gradient matrix G of the hypernode graph $\mathbf{h}$ associated with the set of games Γ, and let us define the matrix G_μ by

$$G_\mu = \begin{pmatrix} G & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \sqrt{\frac{\mu}{n}} B & \end{pmatrix},$$

where B is the $n \times (m+1)$ matrix defined by

$$\begin{cases} \forall i = 1, \ldots, n \quad B_{i,i} = -1 \text{ and } B_{i,m+1} = 1, \\ B_{i,j} = 0 \text{ in all other cases.} \end{cases}$$

The matrix G_μ is the gradient of the hypernode graph $\mathbf{h}_\mu$ obtained from the hypernode graph $\mathbf{h}$ by adding a regularizer node R, an hyperedge between every player node and the regularizer node R with node weights μ/n (such a hyperedge can be viewed as an edge with edge weight μ/n). The construction is illustrated in Figure 4.6 with the hypernode graph reduced to the single hyperedge depicted in Figure 4.5.

Let us denote by r the vector $(s(0), \ldots, s(m), q)$ and by Δ_μ the Laplacian matrix of $\mathbf{h}_\mu$. Since the original Laplacian Δ is equal to $G^T G$, we can write

$$r^T \Delta_\mu r = r^T G_\mu^T G_\mu r = s^T \Delta s + \frac{\mu}{n} r B^T B r \ .$$

Since $r B^T B r = \sum_i (s_i - q)^2$, we can rewrite the regularized problem (4.4) under the final form

$$\begin{array}{ll} \underset{r\in\mathbb{R}^{N+1}}{\text{minimize}} & r^T\Delta_\mu r \\ \text{subject to} & \forall n+1 \leqslant j \leqslant n+t_\ell,\ r(j)=0 \text{ (for lazy nodes)} \\ & \forall n+t_\ell+1 \leqslant j \leqslant n+t_\ell+t_o,\ r(j)=o_j \text{ (for outcome nodes)} \end{array} \tag{4.5}$$

In order to provide some meaningful guidelines for the choice of μ, we have to evaluate the relative magnitude of $s^T\Delta s$ and $\mu\sigma(s_p)^2$.

Proposition 19. *Let us consider a pool of players whose skills are drawn from a Gaussian law $\mathcal{N}(0,\sigma^2)$ and let us denote by G the random variable that gives the difference of skills between two independent teams composed of T independent players each. Then, if we consider p random variables $G_1,\ldots,G_p$ following the same law as G, we have*

$$\mathbb{E}\left[\sum_{i=1}^{p} G_i^2\right] = 2Tp\sigma^2$$

Proof. Let us consider a team of T independent players. The total skill X of such a team follows again a centered Gaussian law $\mathcal{N}(0,T\sigma^2)$. If X_a and X_b are the total skills of two independent teams, we can assume that $cov(X_a,X_b)=0$ and we can write

$$G = X_a - X_b \sim \mathcal{N}(0,2T\sigma^2) \tag{4.6}$$

Thus, it follows that $\frac{G}{\sqrt{2T}\sigma}$ follows a standard normal law. We know that if $Z_1,\ldots,Z_p$ are independent, standard normal random variables, then the sum of their squares is distributed according to a chi-squared distribution with p degrees of freedom. Thus, we have:

$$\sum_{i=1}^{p}\frac{G_i^2}{2T\sigma^2} \sim \chi^2(p)\ .$$

We conclude the proof by noticing that the mean of a chi-squared distribution with p degree of freedom is p. □

Let us now remark that the term $s^T\Delta s$ is the sum of the squared differences of skills between teams, for all the games available in Γ. If we assume that the player skills actually follow a Gaussian law, Proposition 19 allows us to state that the expected value of $s^T\Delta s$ should be close to $2Tp\sigma^2$ where T is the average number of players in a team. Note that the Gaussian hypothesis

has been validated experimentally by studying the distribution of the learnt skills for several values of μ.

As stated above, the regularization term is $\mu\sigma^2(s_p) \approx \mu\sigma^2$. In order to have a consistent tradeoff in our optimization process, the quantity μ/n should have the same order of magnitude as the expected value $2Tp/n$. Note that $2Tp/n$ is the average number of games played by a player.

4.2.3 Inferring Skill Ratings and Predicting Game Outcomes

We have shown that predicting skill ratings can be rewritten as the optimization problem (4.5). It should be noted that it can also be viewed as a semi-supervised learning problem on the hypernode graph $\mathbf{h}_\mu$ because the question is to predict node scores (skill ratings) for player nodes when node scores for lazy nodes and outcome nodes are given. As shown for instance in Proposition 6, Δ_μ is a positive semidefinite real-valued matrix as a hypernode graph Laplacian. We propose two algorithms based on this results that allows to estimate the skill ratings and present our general protocol to predict the game outcomes.

Algorithm 1: H-ZGL

This algorithm is a direct adaptation of the (graph) semi-supervised learning algorithm presented in Zhu et al. (2003). The main idea is to focus on the first order condition $\Delta_\mu r = 0$ (also called harmonic condition). In order to solve this condition, we consider the following block decomposition based on the node ordering defined in (4.2):

$$r = \begin{pmatrix} r_P \\ r_L \\ r_O \end{pmatrix} ,$$

where $r_P \in \mathbb{R}^n$ (player nodes), $r_L \in \mathbb{R}^{t_\ell}$ (lazy nodes) and $r_O \in \mathbb{R}^{t_o}$ (outcome nodes). We rewrite the first-order condition using a corresponding decomposition of the regularized Laplacian Δ_μ:

$$\begin{pmatrix} \Delta_\mu^{P,P} & \Delta_\mu^{P,L} & \Delta_\mu^{P,O} \\ \Delta_\mu^{L,P} & \Delta_\mu^{L,L} & \Delta_\mu^{L,O} \\ \Delta_\mu^{O,P} & \Delta_\mu^{O,L} & \Delta_\mu^{O,O} \end{pmatrix} \begin{pmatrix} r_P \\ r_L \\ r_O \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} .$$

If we consider the first line of this block system, we get the following equation:

$$\Delta_\mu^{P,P} r_P = -\left(\Delta_\mu^{P,L} r_L + \Delta_\mu^{P,O} r_O\right) \tag{4.7}$$

From the general properties of the Moore-Penrose pseudo-inverse, the least-square solution to the last equation (4.7) is given by:

$$r_P = -(\Delta_\mu^{P,P})^\dagger \left(\Delta_\mu^{P,L} r_L + \Delta_\mu^{P,O} r_O\right) \tag{4.8}$$

We can now inject our optimization constraints on r_L and r_O in (4.8): we know that $r_L = \mathbf{0}$ (skills of the lazy nodes) and that $r_O = \mathbf{o}$ where $\mathbf{o}$ is the vector formed with the outcomes $(o_j)_{j=1,\dots,t_o}$ of the different games. Finally, we get the final closed formula for r_P:

$$r_P = -(\Delta_\mu^{P,P})^\dagger \Delta_\mu^{P,O} \mathbf{o} \tag{4.9}$$

It should be noted that, in general, the learnt vector r_P won't provide a fully harmonic solution to $\Delta_\mu r = 0$. We can define the error vector e_P by

$$e_P = \Delta_\mu^{P,P} r_P + \Delta_\mu^{P,O} \mathbf{o} \in \mathbb{R}^n \ .$$

For a given player i, the quantity $e_P(i)$ can be interpreted as the uncertainty of the skill $r_P(i)$. Indeed, if $e_P(i)$ is high, we can deduce that the learnt skill $r_P(i)$ remains inconsistent with many games played by i. Conversely, if $e_P(i)$ is small, $r_P(i)$ should satisfy most of the game constraints, which give us more confidence about the result. An interesting question is whether we can leverage this information in order to improve the hyperedge weights. A solution could be to use a two-step algorithm (EM-like):

- Step 1 (Expectation): Estimate the best skill vector r_P and the corresponding error vector e_P
- Step 2 (Minimization): Update the weights of the hyperedges in order to reduce the norm of the error vector e_P

Note that the second step should learn a weighting model and not directly the weights of the game hyperedges. Indeed, we have to keep in mind that the prediction of unknown games requires a node weighting model that should remain consistent with the weighting model used in the training set.

Algorithm 2: H-SVR

In order to predict skill ratings, another approach is to infer player nodes scores from lazy nodes scores and outcome nodes scores using a regression algorithm. For this, we consider the hypernode graph kernel $\Delta_\mu^\dagger$ (defined as the Moore-Penrose pseudoinverse of the Laplacian Δ_μ) and train a regression support vector machine (see Drucker et al. 1996) on the last $t_\ell + t_o$ nodes (lazy nodes and outcome nodes). Finally, we estimate the player skills r_P using the learnt SVM model. Note that this approach does not directly solve

the original optimization problem (4.5) but rather relies on the geometry induced by the regularized hypernode graph $\mathbf{h}_\mu$ (pseudometric d^1). Indeed, the feature space associated to the kernel $\Delta_\mu^\dagger$ is an Euclidean space where the distance between two nodes i and j is equal to $d^1(i,j)$.

Predicting game outcomes

Using the two previous methods, we can infer skill ratings for players from a given set of games together with their outcomes. The inferred skill ratings can be used to predict game outcomes for new games. For this, we suppose that we are given a training set of games Γ_l with known outcomes together with a set of testing games Γ_u for which game outcomes are hidden. The goal is to predict game outcomes for the testing set Γ_u. Note that other works have considered similar questions in the online setting as in Herbrich et al. (2006), Elo (1978) while we consider the batch setting. For the prediction of game outcomes, we first apply a skill rating prediction algorithm using the training set Γ_l and output a skill rating function s^*. Then, for each game in Γ_u, we evaluate the inequality $(\mathcal{C}_j)$ with the skills defined by s^* and decide the winner. Note that the evaluation of $(\mathcal{C}_j)$ require the definition of the contributions $c_j(i)$ for the new game. This definition should be done in a manner consistent with the weighting model used in the training set. For every player who do not appear in the training set, the skill value is fixed a priori to the mean of the known player skills.

Algorithm 2 Predicting game outcomes

Require: Training set of games Γ_l, set of testing games Γ_u

1: Build the regularized hypernode graph $\mathbf{h}_\mu$ as described in Sections 4.2.1 and 4.2.2
2: Compute an optimal skill rating s^* using H-ZGL or H-SVR.
3: Compute the mean skill $\tilde{s}$ among players in Γ_l
4: **for** each game in Γ_u **do**
5: Assign skill given by s^* for players involved in Γ_l, and $\tilde{s}$ otherwise
6: Evaluate the inequality $(\mathcal{C}_j)$ and predict the winner
7: **end for**

4.3 Experiments

In this section, we report our experimental results for the inference of the player skills and the prediction of the game outcomes for different datasets.

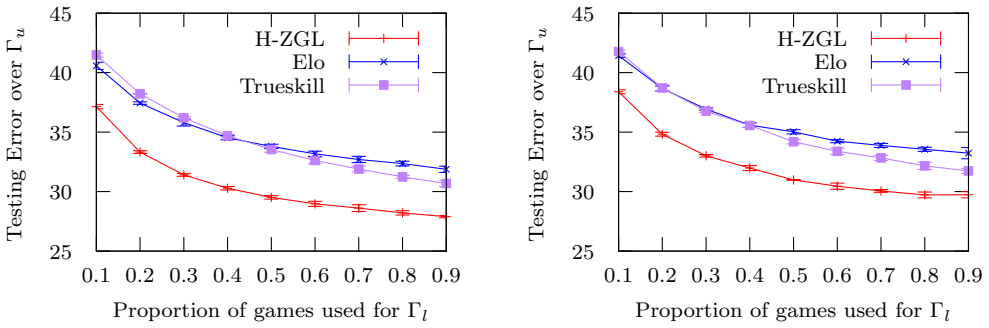


Figure 4.7: Predictive error for ATP games (*left*) and WTA games (*right*)

4.3.1 Tennis Singles

We consider two datasets of tennis singles collected between January 2011 and June 2014. Tennis singles are played by two players. Each game has a winner (no draw is allowed). A game is played in two or three winning sets. The final score corresponds to the number of sets won by each player during the game. The first dataset comes from ATP tournaments (World Tour, Challengers and Futures) and consists in 182665 games with 15385 players. The second dataset comes from WTA tournaments (International Events, Premier Events and ITF tournaments) and consists in 112499 games with 8292 players.

In each experiment, we randomly select a training subset Γ_l of games and all the remaining games define a testing subset Γ_u. We will consider different sizes for the training set Γ_l and will compute the outcome prediction error on the corresponding set Γ_u. More precisely, for a given proportion ρ varying from 10% to 90% , we build a training set Γ_l using ρ% of the games chosen randomly among the full game set, the remaining games form the test set Γ_u. Given a training set of games Γ_l and a test set Γ_u, we follow the experimental process described in Algorithm 2.

All the player contributions are set to 1 both for the learning part and the prediction part. We construct the hypernode graph $\mathbf{h}_\mu$ using the game set Γ_l, as shown in Section 4.2. In the optimization problem 4.5, the game outcomes o_j are defined to be the net scores of the winner teams (that is, the number of sets won minus the number of sets lost). This allows to take account of the score gaps when computing player skills. In order to reduce the number of nodes, all the lazy nodes are merged in a single one that is shared by all the hyperedges. We do the same for the outcome nodes because score differences can be 1, 2 or 3. In order to complete the definition of $\mathbf{h}_\mu$, we need also to fix the value of the regularization parameter μ/n. In

Section 4.2.2, we have shown that μ/n should have of magnitude similar to the average number of games played by a player. For this reason, we use $\mu/n = 16$ for these two datasets.

Given $\mathbf{h}_\mu$, following Algorithm 2, we apply the skill rating prediction algorithms H-ZGL and H-SVR. In order to compare our method, we also infer skill ratings using Elo and Trueskill[2] Then, we predict game outcomes from the inferred skill ratings. The results are given in Figure 4.7 (for each value of ρ, we repeat the experiment 10 times). We can notice that H-ZGL outperforms strictly Trueskill and Elo for all values of ρ. It can be noted that Elo and Trueskill are relatively close on these two datasets (1vs1 case). In what follows, we focus on general multiplayer games where the sizes of the teams are strictly greater than 1.

4.3.2 Tennis Doubles

We now consider a dataset of tennis doubles collected between January 2009 and September 2011 from ATP tournaments. Tennis doubles are played by two teams of two players and, as in the case of tennis singles, no draw is allowed. The dataset consists in 10028 games with 1834 players. We present in Figure 4.8 and 4.9 several statistics related to the double dataset. It is worth noticing that many players have only played once. Therefore, the skill rating problem and the game outcome prediction problem become far more difficult to solve when few games are used for learning. Indeed, when the number of games in Γ_l, the number of players in the test set who are involved in a game of the training set is small as well. In this case many players will have a skill estimated to be the average skill.

We follow the same strategy as in Section 4.3.1 to build the hypernode graph $\mathbf{h}_\mu$. At the end, we have at most 1839 nodes: at most 1834 player nodes (depending on the selected games), 1 lazy node, 3 outcome nodes, and 1 regularizer node. In this experiment, we fix μ/n to 16 as in the previous case. Moreover, since the Elo algorithm is unable to handle multiplayer games, we replace it by Elo Duelling described in Section 4.1.2. The results are given in Figure 4.10 (for each value of ρ, we repeat the experiment 10 times). It should be noted that Elo duelling performs poorly, which is not surprising since this method is mainly designed for the $1vs1$ case. Also, it can be noted that H-ZGL is significantly better than Trueskill whatever is the chosen proportion.

[2]TrueSkill and Elo implementations are from Hamilton (2012). Results were double-checked using Lee (2013b) and Lee (2013a). Parameters of Elo and TrueSkill are the default parameters of Hamilton (2012) ($K = 32$ for Elo, $\mu_0 = 25, \beta = 12.5, \sigma = 8.33$ and $\tau = 0.25$ for TrueSkill).

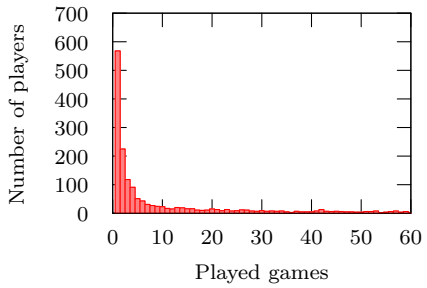


Figure 4.8: Distribution of the number of player against the number of played games

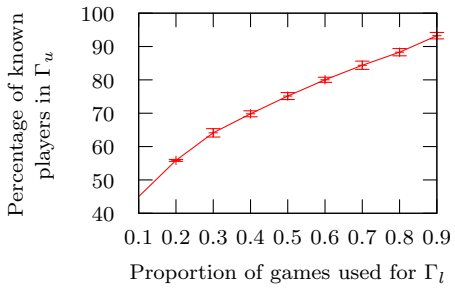


Figure 4.9: Average percentage of players in Γ_u which are involved in some game in Γ_l

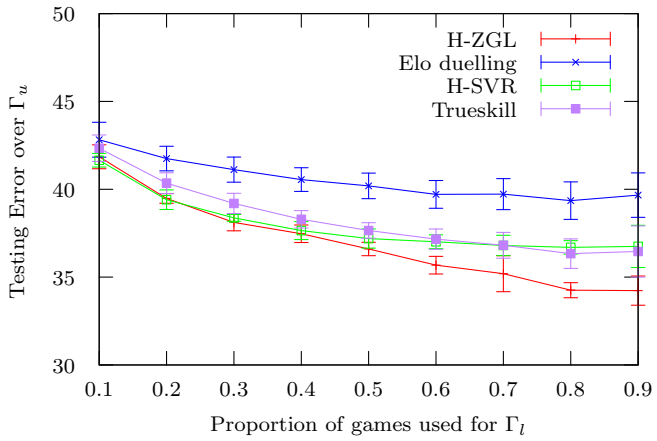


Figure 4.10: Predictive error depending on the proportion of games used to build Γ_l

4.3.3 Xbox Title Halo2

The Halo2 dataset was generated by Bungie Studio during the beta testing of the XBox title Halo2. It has been notably used in Herbrich et al. (2006) to evaluate the performance of the Trueskill algorithm. We consider the *Small Teams* dataset with 4992 players and 27536 games opposing up to 12 players in two teams which can have a different size. Each game can result in a draw or a win of one of the two teams. The proportion of draws is 22.8%. As reported in Herbrich et al. (2006), the prediction of draws is challenging and it should be noted that Trueskill and our algorithm fail to outperform significantly a random guess for the prediction of draw.

We again consider the experimental process described in Algorithm 2. As

for the Tennis datasets, we fix all the players contributions in games to 1. In the optimization problem 4.5, the game outcomes o_j are defined to be equal to 1 when the game has a winner and 0 otherwise because the game scores vary depending on the type of game. As above, we merge the lazy nodes into a single one and do the same for the outcome nodes. The value of μ/n is again set to 16. We represent in Figure 4.11 the value of the ratio $\frac{2pT}{n}$ as described in Proposition 19.

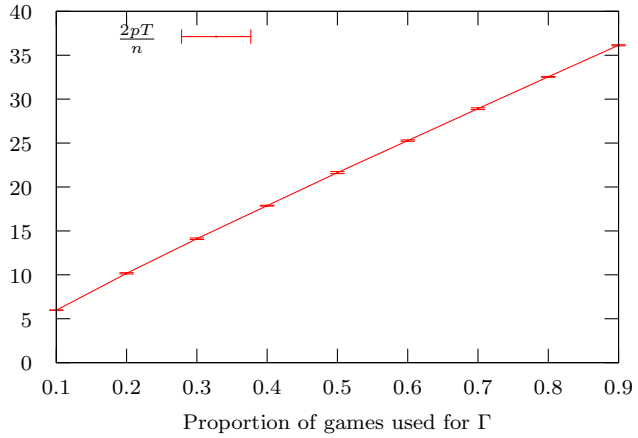


Figure 4.11: Evolution of $\frac{2pT}{n}$ for the dataset Halo SmallTeams

As for the Tennis dataset, we compare the skill rating algorithms H-ZGL, H-SVR, Elo Duelling and Trueskill. The number of prediction errors for game outcomes is computed assuming that a draw can be regarded as half a win, half a loss Lasek et al. (2013). We present the experimental results in Figure 4.12. For a proportion of 10% of games in the training set, H-ZGL, H-SVR and Trueskill give similar results while with larger training sets, our hypernode graph learning algorithms outperform Trueskill. Contrary to the previous experiment, H-SVR performs better than H-ZGL. This result has however to be qualified given the fact that H-SVR depends on the soft margin parameter C whereas H-ZGL is strictly non-parametric.

4.4 Conclusion

In this chapter, we have used the hypernode graph framework to address the problem of skill rating. We have proposed a simple model for multiplayer games and used dedicated nodes to represent the outcome of the games. With a simple regularization term, we have shown that we can obtain very competitive results compared to specialized algorithms such as Elo duelling or Trueskill. It should be noted that we have used the kernel $\Delta^{\dagger}$ in a support

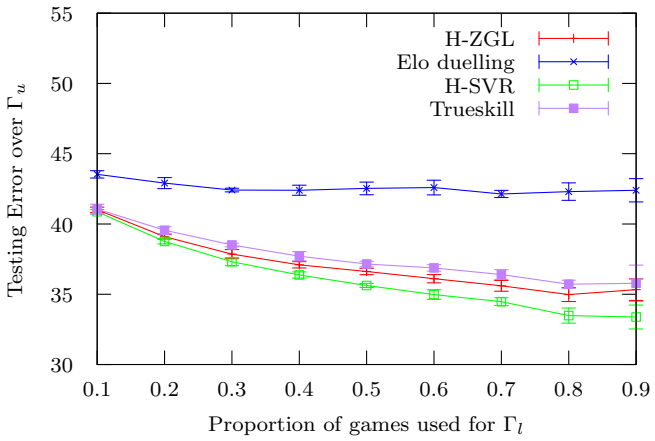


Figure 4.12: Predictive error depending on the proportion of games used to build Γ_l

vector machine, which is equivalent to use the metric d^1 on the hypernode graph. We strongly believe that interesting work remains to be achieved in order to build better models and algorithms. In particular, the question of individual weights in hyperedge should be addressed to allow more flexibility in skill rating. Note that there are two main ways to tackle this problem: the first one is to leverage our apriori knowledge of the game to tune the weights; the second one is to use our aposteriori knowledge (for instance, the learning error as indicated in Section 4.2.3) in order to update a weighting model. In the next chapter, we discuss how to define the notion of cut on hypernode graphs and propose new applications.

Chapter 5

Perspectives and open problems

Chapter abstract In this chapter, we present several open research problems in hypernode graphs. We first study the notion of cut in hypernode graphs and its main properties. We discuss the potential applications and review some promising lines of research. We then tackle the issue of directivity in hypernode graphs and give some pointers to other related theories. We finally discuss hypernode graphs from an algebraic perspective and present a convex hull conjecture.

5.1 Cuts in hypernode graphs

In this section, we put our focus on the problems of graph and hypernode graph partitioning. A *p-way graph partition* on a graph $\mathbf{g} = (N, E)$ is defined as a collection of sets $\mathcal{P} = (C_1, \ldots, C_p)$ such that

$$\bigcup_{i=1}^{p} C_i = N \text{ and } \forall i \neq j,\ C_i \cap C_j = \varnothing \ .$$

Note that, contrary to the classic definition of a p-way partition, we allow $\mathcal{P}$ to contain the empty set.

When $p = 2$, the graph partition is called *cut* and reduces to a couple $(C, \overline{C})$. Finding a "good" cut according to some criteria is a fundamental problem in graph theory, with applications in various fields such as computer vision (Greig et al. 1989), communication networks (Karger 1993, section 7) and operational research (Huang 2011).

The objective of this section is to discuss how we can define cuts for the case of hypernode graphs. We start by recalling important definitions and results from the graph case. Then we propose an extension for the hypernode graphs and discuss the related research problems.

5.1.1 Cuts in undirected graphs

Let us consider a cut $(C, \overline{C})$ on a graph $\mathbf{g} = (N, E)$. We define the corresponding *cut-set* as the set of edges that have one endpoint in each subset C and $\overline{C}$. We define the *cut-size* $\mathrm{Cut}\,(C) = \mathrm{Cut}\left(\overline{C}\right)$ to be the sum of the weights of all edges in the cut-set. Namely:

$$\mathrm{Cut}\,(C) = \sum_{\substack{i \in C \\ j \in \overline{C}}} W_{i,j} \quad . \tag{5.1}$$

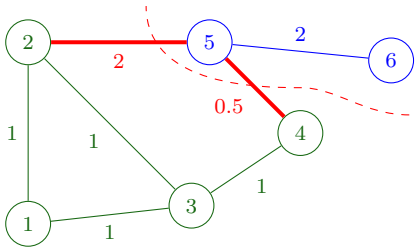


Figure 5.1: Graph $\mathbf{g}_{14}$ and simple cut with $C = \{5, 6\}$ and $\mathrm{Cut}\,(C) = 2.5$.

An example of cut is depicted in Figure 5.1. We can observe that the cut-size is always positive as a sum of positive weights. Moreover, we have $\mathrm{Cut}\,(\varnothing) = \mathrm{Cut}\,(N) = 0$ by definition. It should be also noted that the cut-size can be expressed using the gradient operator. Indeed, we have for all $C \subseteq N$,

$$\mathrm{Cut}\,(C) = \sum_{\{i,j\} \in E} \left[\mathrm{grad}(\mathbf{1}_C)(i,j)\right]^2 = \Omega(\mathbf{1}_C) \quad . \tag{5.2}$$

For a given edge $\{i, j\} \in E$, the quantity $\mathrm{grad}(\mathbf{1}_C)(i, j)$ can be understood as the disequilibrium induced by the partitioning $(C, \overline{C})$. As a consequence of Equation (5.2), the smoothness measure Ω can be regarded as a (convex) continuous extension of the graph cut-size.

An other interesting property of the cut-size is given in the following proposition.

Proposition 20. *The cut-size is* submodular, *i.e.,* $\forall C \subset C' \subset N$ *and* $i \in N \backslash C'$, *it satisfies*

$$\mathrm{Cut}\,(C \cup \{i\}) - \mathrm{Cut}\,(C) \geqslant \mathrm{Cut}\left(C' \cup \{i\}\right) - \mathrm{Cut}\left(C'\right) \quad . \tag{5.3}$$

Proof. For any set $C \subset N$ and any node $i \in \overline{C}$, we define

$$\mathcal{G}(C,i) = \mathrm{Cut}\left(C \cup \{i\}\right) - \mathrm{Cut}\left(C\right) \quad .$$

We have to show that, for all $C \subset C' \subset N$ and for all $i \in N \backslash C'$, $\mathcal{G}(C,i) \geqslant \mathcal{G}(C',i)$. Since $\mathbf{1}_{\{i\}} = \mathbf{e}_i$, we can write using Equation (5.2)

$$\begin{aligned} \mathcal{G}(C,i) &= \Omega(\mathbf{1}_C + \mathbf{e}_i) - \Omega(\mathbf{1}_C) \\ &= (\mathbf{1}_C + \mathbf{e}_i)^T \Delta (\mathbf{1}_C + \mathbf{e}_i) - \mathbf{1}_C^T \Delta \mathbf{1}_C \\ &= \mathbf{e}_i^T \Delta \mathbf{e}_i + 2\mathbf{1}_C^T \Delta \mathbf{e}_i \quad . \end{aligned}$$

We define D to be the (possibly empty) set $C' \backslash C$. We have similarly

$$\mathcal{G}(C',i) = \mathbf{e}_i^T \Delta \mathbf{e}_i + 2\left(\mathbf{1}_C + \mathbf{1}_D\right)^T \Delta \mathbf{e}_i \quad .$$

Hence we get finally

$$\begin{aligned} \mathcal{G}(C,i) - \mathcal{G}(C',i) &= -2\mathbf{1}_D \Delta \mathbf{e}_i \\ &= \sum_{j \in D} W_{i,j} \\ &\geqslant 0 \ , \end{aligned}$$

which concludes the proof. □

Submodular functions are real-valued set functions which satisfy the *diminishing return* property defined in Equation 5.3. The class of functions has been extensively studied from an optimization perspective, which has led to many theoretical results. We can cite Nemhauser et al. (1978) that provides fundamental results on the approximate maximization problem and Lovász et al. (1993, Chapter 10) that study, among others, the problem of minimization. In particular, it shows that there exists an algorithm that computes the minimum of any submodular function in polynomial time. More recently, Iwata and Orlin (2009) have proposed a strongly polynomial algorithm for exact minimization of submodular functions. In our context, one may want to maximize or minimize the cut-size on a given graph $\mathbf{g}$. We investigate both cases in the following sections.

The Max-Cut problem

We first begin with the maximization problem presented below.

PROBLEM 1: *Maximum Cut*

INPUT: Graph $\mathbf{g} = (N, E)$

SOLVE:

$$\underset{C \subset N}{\text{Maximize}}\, \text{Cut}\,(C)$$

We illustrate the Max-Cut problem with Example 15 below.

Example 15. *Let us consider the simple communication network represented by the undirected graph* $\mathbf{g}_{15}$ *(depicted in Figure 5.2). Each node represents a server that can communicate with its peers using different channels (represented by the graph edges). For a given nodeset* C*, the associated cut size* $\text{Cut}\,(C)$ *can be seen as the number of broken connections implied by the simultaneous failure of the servers that belong to* C*. For this reason, we can define the* criticality *of a set* C *as its cut size* $\text{Cut}\,(C)$*. If our goal is to find the set of nodes whose failure will impact the most strongly on the network, we have to solve a Max-Cut problem.*

In the case of $\mathbf{g}_{15}$*, we can observe that the network is centered around the "bottleneck" node 3 and we can verify that*

$$\text{Cut}\,(\{3\}) = 4 = \max_{C} \text{Cut}\,(C) \quad .$$

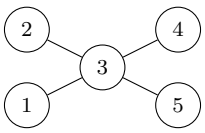


Figure 5.2: Simple graph $\mathbf{g}_{15}$ with bottleneck

Problem 1 is APX-hard (see Papadimitriou and Yannakakis 1991) and the current best polynomial-time approximate solution is given by Goemans and Williamson (1995), which achieves a guaranteed ratio of 0.87856. Solving exactly the maximization is computationally challenging and leads to complex algorithms (see for example Krislock et al. 2014, which use a branch-and-bound paradigm). Some recent works propose to tackle the problem by leveraging the submodularity property (see Buchbinder et al. 2014, with some additional cardinality constraints).

The Min-Cut Problem

The Min-Cut problem can be seen as a 2-class partitioning problem: the objective is to separate the nodeset N into two parts C and $\overline{C}$ such that we have few links between C and $\overline{C}$ (small cut-size $\mathrm{Cut}\,(C)$). Formally, we write

PROBLEM 2: *Minimum Cut*

INPUT: Graph $\mathbf{g} = (N, E)$

SOLVE:

$$\underset{C \subset N}{\text{Minimize}}\ \mathrm{Cut}\,(C)$$

Note that, contrary to Problem 1, it is very easy to find an optimal set C for Problem 2. Indeed, as recalled in Proposition 20, we have

$$\mathrm{Cut}\,(\varnothing) = \mathrm{Cut}\,(N) = 0 = \min_{C \subset N} \mathrm{Cut}\,(C) \quad .$$

Actually, we can even carry the reasoning one step further by noticing that

Proposition 21. *A set $C \subseteq N$ satisfies $\mathrm{Cut}\,(C) = 0$ if and only if C is a (possibly empty) union of connected components.*

Proof. As stated in (5.2), we have for all C, $\mathrm{Cut}\,(C) = \Omega(\mathbf{1}_C)$. If C is the union of m connected components $C_1, \ldots, C_m$, then $\mathbf{1}_C = \sum_{i=1}^{m} \mathbf{1}_{C_i}$ and

$$\mathrm{Cut}\,(C) = \Omega(\mathbf{1}_C) = \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{1}_{C_i}^T \Delta \mathbf{1}_{C_j} = 0 \quad ,$$

since $\mathrm{Null}(\Delta)$ is spanned by the indicator vectors of the connected components (see Proposition 2). Conversely, if $\mathrm{Cut}\,(C) = 0$, then $\mathbf{1}_C$ is in $\mathrm{Null}(\Omega) = \mathrm{Null}(\Delta)$, and $\mathbf{1}_C$ can be expressed as a linear combination of the indicator vectors of m' connected components $C_1, \ldots, C_{m'}$. Since $\mathbf{1}_C$ is an indicator vector, the only solution is to have $\mathbf{1}_C = \sum_{i=1}^{m'} \mathbf{1}_{C_i}$ and

$$C = \bigcup_{i=1}^{m'} C_i \quad .$$

□

When the graph is connected, the solution of the unconstrained Min-Cut reduces to N and $\varnothing$. For this reason, we usually replace the plain Min-Cut problem with a constrained Min-Cut problem. We distinguish between

two main type of constraints: the *balancing contraints* and the *node constraints*.

The objective of the *balancing constraints* is to prevent the trivial cases $C = \varnothing$ and $C = N$ from happening by adding size constraints to the partition $(C, \overline{C})$. A common example is the case of the Min-Ratio-Cut problem presented below.

PROBLEM 3: *Minimum Ratio-Cut*

INPUT: Graph $\mathbf{g} = (N, E)$

SOLVE:

$$\underset{C \subset N}{\text{Minimize}} \, \text{RatioCut}\,(C) \quad ,$$

where $\text{RatioCut}\,(C) = \left(\frac{1}{|C|} + \frac{1}{|N|-|C|}\right) \text{Cut}\,(C)$.

Note that it has been shown (see for instance Kunegis et al. 2010) that the following relation holds for any C:

$$\text{RatioCut}\,(C) = \Omega(r_C) \quad , \tag{5.4}$$

where r_C is defined by

$$r_C(i) = \begin{cases} \sqrt{\dfrac{|N| - |C|}{|N||C|}} & \text{if } i \in C \\ -\sqrt{\dfrac{|C|}{|N|(|N| - |C|)}} & \text{otherwise} \end{cases} .$$

We can observe that $\|r_C\|_2 = 1$ and $r_C \in \text{Span}(\mathbf{1})^\perp$. Consequently, a possible continuous relaxation of Problem 3 is

$$\text{Minimize } \Omega(f) \text{ subject to } \|f\|_2 = 1 \text{ and } f \in \text{Span}(\mathbf{1})^\perp \tag{5.5}$$

The additional constraints $f \in \text{Span}(\mathbf{1})$ and $\|f\|_2 = 1$ allow to eliminate the undesirable cases $C = N$ and $C = \varnothing$. We know from Proposition 2 that $\text{Span}(\mathbf{1}) \subset \text{Null}(\Delta)$ so $\text{Null}(\Delta)^\perp \subset \text{Span}(\mathbf{1})^\perp$, which leads to consider the alternate problem

$$\text{Minimize } \Omega(f) \text{ subject to } \|f\|_2 = 1 \text{ and } f \in \text{Null}(\Delta)^\perp \tag{5.6}$$

When the graph is connected, (5.5) and (5.6) are equivalent since $\text{Span}(\mathbf{1}) = \text{Null}(\Delta)$. In the other cases, considering (5.6) leads to disregard the cuts $(C, \overline{C})$ where C is an union of connected components. The problem (5.6)

can be solved directly by conducting a spectral analysis of the Laplacian matrix Δ. Indeed, since Δ is symmetric, $\text{Null}(\Delta)^{\perp}$ is a direct orthogonal sum of the eigenspaces $E_{\lambda_0}, E_{\lambda_1}, \ldots, E_{\lambda_k}$ associated to the eigenvalues of the Laplacian matrix $\lambda_0 = 0 < \lambda_1 < \cdots < \lambda_k$. Moreover, we know that the smoothness of an unit vector in E_{λ_i} is equal to λ_i. Consequently, the solutions of (5.6) are the unit vectors of the eigenspace E_{λ_1} associated to the smallest but non-null eigenvalue λ_1. When $\dim(E_{\lambda_1}) = 1$, there is an unique solution called *unit Fiedler vector*.

Given a solution f of one of the relaxed problem (5.5) and (5.6), we can define the set $C(t)$ by

$$C(t) = \{i \text{ such that } f(i) \geqslant t\} \ .$$

Several strategies exists to choose the threshold t but the most common ones are the *sign* strategy (choose $t = 0$) and the *ratio* strategy (choose t that minimize the ratio $\frac{1}{|C(t)|} + \frac{1}{|N|-|C(t)|}$). This algorithm can be extended in order to solve a m-class clustering problem instead of a cut, which lead to the popular *spectral clustering* algorithm (see Von Luxburg 2007).

A major limitation of this approach is that we cannot guarantee in the general case that the relaxed solutions of (5.5) or (5.6) are close to the optimal solutions of Problem 3. Partial results rely on *Cheeger's inequality* (see for instance Chung 2005) that makes the connection between the eigenvalue λ_1 and the discrete optimum.

Note that we can find other formulations of the minimum cut with *balancing constraints* (see for instance Bresson and Szlam 2010). Similarly, other continuous relaxations of the cut-size $\text{Cut}(C)$ can be envisaged such as the popular Lovasz extension described in Lovász (1983). An interesting property of the Lovasz extension is that it is convex as soon as the function is submodular, which makes it particularly suitable for use with the cut-size (see Proposition 20). However, we do not consider it in the following, since the submodularity property do not hold for hypernode graphs, as shown in the next section.

We can observe that the *balancing constraints* do not require any particular expertise on the specific problem. On the contrary, the second type of constraints called *node constraints* are based on a priori knowledge.

PROBLEM 4: *Minimum Cut With Node Constraints*

INPUT: Graph $\mathbf{g} = (N, E)$ and nodesets $N_1, N_2 \subset N$

SOLVE:

$$\underset{C \subset N}{\text{Minimize}}\, \text{Cut}\,(C) \quad ,$$

such that $N_1 \subseteq C$ and $N_2 \subseteq \overline{C}$.

Example 16. *Let us consider the binary image depicted in Figure 5.3 (left). Our objective is to denoise the image and find the vertical edge between the "black" left part and the "white" right part.*

To do so, we first compute a prior distribution $p(x)$ (probability of a pixel to be black) based on the pixel values. Then, we build a graph $\mathbf{g}$ *as shown in Figure 5.3 (right) where each pixel is associated with a node. We add two auxiliary nodes b and w (**b**lack and **w**hite) and create edges depending on the prior $p(x)$ (if $p(x) > 0.5$, the node is linked to b; otherwise the node is linked to w). Our denoising problem can be solved by considering Problem 4 instantiated with the pixel graph* $\mathbf{g}$*, the nodeset $N_1 = \{b\}$ and the nodeset $N_2 = \{w\}$.*

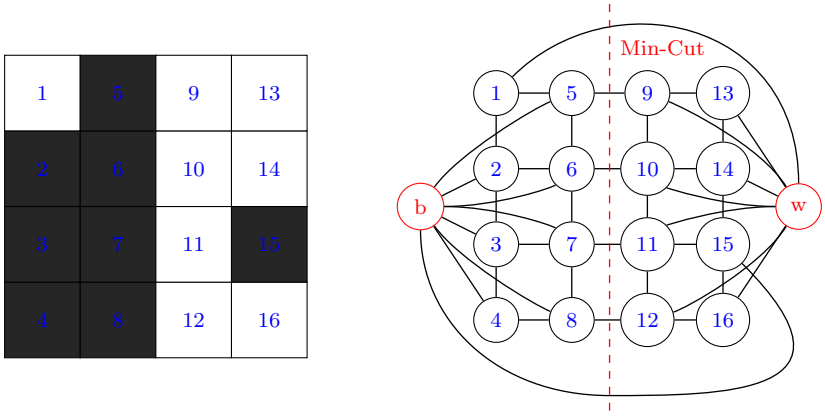


Figure 5.3: Binary picture with noise and corresponding graph with Min-cut

When the input sets N_1 and N_2 are singletons, Problem 4 is proved to be equivalent to an other problem called Max-Flow that can be solved in polynomial time (see Edmonds and Karp 1972). The singleton node constraints have many applications in networks as shown for instance in Picard and

Queyranne (1980) or in Greig et al. (1989). In the following, we will see that the Min-Cut problem also leads to natural applications in the hypernode graph case.

5.1.2 Cuts and hypernode graphs

In this section, we introduce a definition of cuts for hypernode graphs. We first propose a motivating example.

Example 17. *Let us consider four players* 1,2,3 *and* 4 *playing 1vs1 games. We are given the following game log:*

- 1 *lost to* 3
- 1 *lost to* 4
- 2 *lost to* 3
- 2 *lost to* 4

We can use an undirected hypernode graph to represent these games, as shown in Figure 5.4. For the sake of simplicity, all the weights are set to 1. *We use the additional nodes Z (lazy helper) and H (active helper) in order to express the known outcomes of the games.*

Our objective is to cluster the players in two balanced groups: the skilled players and the unskilled players. To do this, we have to find a partition (C, C') of the nodeset N such that $H \in C$ and $Z \in C'$. Moreover, we expect the players in C to have a different profile from the players in C', which requires us to define an adequate objective function. We can notice that this problem looks similar to Problem 4 presented above (partitioning problem with node constraints).

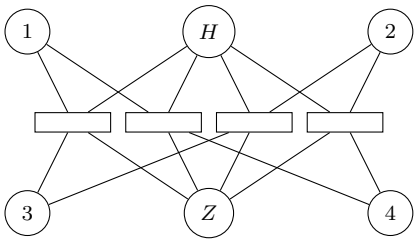


Figure 5.4: Hypernode graph $\mathbf{h}_{16}$

In the following, we will consider a hypernode graph $\mathbf{h} = (N, H)$ and an arbitrary orientation function ϵ (see Section 2.2). The Equilibrium Condition can be expressed as follows:

$$\forall h \in \mathbf{h}, \quad \sum_{i \in h} \sqrt{w_h(i)}\epsilon(h,i) = 0 \qquad \text{(Equilibrium Condition)}$$

In the graph case, a strong link exists between the cut-size and the gradient operator, as stated in Equation 5.2. Based on this observation, we define the cut-size for hypernode graphs as follows:

Definition 10. *Let* $\mathbf{h} = (N, H)$ *be a hypernode graph and let* C *be a subset of* N*. We define the cut-size of* C *by*

$$\mathrm{Cut}\,(C) = \sum_{h \in \mathbf{h}} [\mathrm{grad}(\mathbf{1}_C)(h)]^2 = \sum_{h \in \mathbf{h}} \left[\sum_{i \in h \cap C} \sqrt{w_h(i)} \epsilon_h(i) \right]^2 \tag{5.7}$$

When all the nodes of h belong either to C or to $\overline{C}$, $\mathrm{grad}(\mathbf{1}_C)(h)$ is null because of the Equilibrium Condition. We illustrate Definition 10 with the simple hypernode graph presented in Figure 5.5.

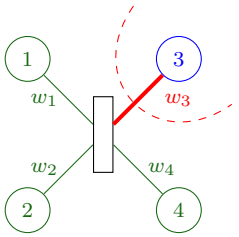


Figure 5.5: Hypernode graph cut with $C = \{3\}$ and $\mathrm{Cut}\,(C) = [\sqrt{w_3}]^2 = [\sqrt{w_1} + \sqrt{w_2} - \sqrt{w_4}]^2$

If the hypernode graph is a graph (i.e., if all the hypernodes are singletons), then the hypernode graph cut-size coincide with the graph cut-size. Indeed, for any set of nodes C, we have

$$\mathrm{Cut}\,(C) = \sum_{h \in \mathbf{h}} [\mathrm{grad}(\mathbf{1}_C)(h)]^2 = \mathbf{1}_C^T \Delta \mathbf{1}_C = \Omega(\mathbf{1}_C) \ , \tag{5.8}$$

which is similar to Equation 5.2.

Example 18. *(Example 17 continued) As a consequence of Equation (5.8), finding a Min-Cut on the hypernode graph* $\mathbf{h}_{16}$ *with* $H \in C$ *and* $Z \in C'$ *is equivalent to find a smooth skill rating* s *restricted to the values* 0 *and* 1 *and subject to the constraints* $s(H) = 1$ *and* $s(Z) = 0$*. Thus, the 2-class clustering problem originally described in Example 17 can be seen as a hypernode graph Min-Cut problem with singleton node constraints on* $\mathbf{h}_{16}$*.*

In the following proposition, we present the main properties of the hypernode graph cut-size.

Proposition 22. *The cut-size satisfies the following properties:*

1. $\forall C, \quad \mathrm{Cut}\,(C) \geqslant 0$

2. $\forall C, \quad \mathrm{Cut}\,(C) = \mathrm{Cut}\,\left(\overline{C}\right)$
3. $\mathrm{Cut}\,(N) = \mathrm{Cut}\,(\varnothing) = 0$

Proof. For all $C \subseteq N$, $\mathrm{Cut}\,(C) \geqslant 0$ as a sum of squared values. To prove the second property, we rewrite the Equilibrium Condition as:

$$\sum_{i \in C \cap h} \sqrt{w_h(i)}\epsilon_h(i) + \sum_{i \in \overline{C} \cap h} \sqrt{w_h(i)}\epsilon_h(i) = 0 \ .$$

Hence, we get

$$\begin{aligned} \mathrm{grad}(\mathbf{1}_C)(h) = & \left[\textstyle\sum_{i \in C \cap h} \sqrt{w_h(i)}\epsilon(h,i)\right]^2 \\ = & \left[-\textstyle\sum_{i \in \overline{C} \cap h} \sqrt{w_h(i)}\epsilon(h,i)\right]^2 \\ = & \quad \mathrm{grad}(\mathbf{1}_{\overline{C}})(h) \ . \end{aligned}$$

Consequently, we have for all C, $\mathrm{Cut}\,(C) = \mathrm{Cut}\,\left(\overline{C}\right)$. Finally, we have $\mathrm{Cut}\,(\varnothing) = 0$ since $\mathbf{1}_\varnothing = \mathbf{0}$. Using the second property, we have $\mathrm{Cut}\,(N) = \mathrm{Cut}\,\left(\overline{N}\right) = \mathrm{Cut}\,(\varnothing) = 0$. □

Note that, contrary to the graph case, the hypernode graph cut is not submodular. This difference is a direct consequence of the negativity of the pairwise weights. Indeed, let us consider a hypernode graph $\mathbf{h}$ that does not reduce to a graph. By definition, we can find a pair of distinct nodes (i, j) such that the pairwise weight $W_{i,j}$ is negative. We define, similarly to the proof of Proposition 20,

$$\mathcal{G}(C, i) = \mathrm{Cut}\,(C \cup \{i\}) - \mathrm{Cut}\,(C) \ .$$

In order to be submodular, we must have for any $C \subsetneq C'$ and any $i \in N \backslash C'$, $\mathcal{G}(C, i) > \mathcal{G}(C', i)$. However, we can show (direct from the graph case) that $\mathcal{G}(\varnothing, i) - \mathcal{G}(\{j\}, i) = W_{i,j} < 0$, which violates the submodularity condition.

5.1.3 The Min-Cut problem on hypernode graphs

Similarly to the graph case, we now consider the minimization of the cut-size as stated in Problem 5 below.

PROBLEM 5: *Minimum Cut on Hypernode graph*

INPUT: Hypernode graph $\mathbf{h} = (N, H)$

SOLVE:

$$\underset{C \subset N}{\text{Minimize}} \, \text{Cut}\,(C) \quad ,$$

In the graph case, the optimal sets were given by the union of connected components. A important difference with the graph case is that there exists many non trivial cuts with a null cut-size.

Proposition 23. *Let $\mathbf{h} = (N, H)$ be a hypernode graph. A subset C induces a cut with null cut-size if and only if C is an independent component.*

Proof. Let us consider $C \subset N$. Recall that we have

$$\text{Cut}\,(C) = \mathbf{1}_C^T \Delta \mathbf{1}_C \quad ,$$

where Δ is the Laplacian matrix of $\mathbf{h}$. Consequently, $\text{Cut}\,(C) = 0$ if and only if $\mathbf{1}_C \in \text{Null}(\Delta)$, which is the exact characterization of an independent component as stated in Proposition 15. □

Proposition 23 is a generalization of Proposition 21 that allows to characterize the sets with null cut-size in the graph case. We can observe that the notion of independent component replaces the graph notion of connected component and that the unconstrained Min-Cut problem reduces to the search of the independent components of the hypernode graph. In Example 17, the ideal cut given by $C = \{3, 4, H\}$ (unskilled players and lazy helper) and $\overline{C} = \{1, 2, Z\}$ (active helper and skilled players) has a null cut-size ($\text{Cut}\,(C) = 0$) and it is easy to verify that C and $\overline{C}$ are two independent components of the graph.

Similarly to the graph case (see Section 5.1.1), we can also consider the constrained Min-cut problem as in Example 17 where the objective is to split the players into two groups: the "good" players that should be similar to H, and the "bad" players that should be similar to Z. For this reason, we consider the singleton node constraints $H \in C$ and $Z \in \overline{C}$. Note that the issue of whether we can extend the Max-Flow Min-Cut theorem for hypernode graphs remains open. We strongly believe that addressing this question will allow a better understanding of the hypernode graph structure and will open the way for the creation of new algorithms. Note that the notion of flow is directly related to the notion of directivity in hypernode graphs that we will address in the next section.

The case of the *balancing constraints* is also interesting since we do not always have specific knowledge on the nodes. As in the graph case, we can define the operator RatioCut as

$$\text{RatioCut}\,(C) = \left(\frac{1}{|C|} + \frac{1}{|N| - |C|}\right) \text{Cut}\,(C) \quad ,$$

which leads to the *Min Ratio-Cut problem for hypernode graphs* (its formulation is similar to Problem 3).

Note that Equation (5.4) from Section 5.1.1 remains true in the hypernode graph case, which allows to make the connection between this problem and the search of a smooth function in $\text{Span}(\mathbf{1})^\perp$:

$$\text{RatioCut}\,(c) = \Omega(r_C) \quad .$$

Indeed, we can write for all $C \subseteq N$,

$$\begin{aligned}
\Omega(r_C) &= \sum_{h\in\mathbf{h}} \left\{ \sum_{i\in C} \left[\epsilon_h(i)\sqrt{w_h(i)} \sqrt{\frac{|N|-|C|}{|N||C|}} \right] - \sum_{i\in\overline{C}} \left[\epsilon_h(i)\sqrt{w_h(i)} \sqrt{\frac{|C|}{|N|(|N|-|C|)}} \right] \right\}^2 \\
&= \sum_{h\in\mathbf{h}} \left\{ \sum_{i\in C} \left[\epsilon_h(i)\sqrt{w_h(i)} \sqrt{\frac{|N|-|C|}{|N||C|}} \right] + \sum_{i\in C} \left[\epsilon_h(i)\sqrt{w_h(i)} \sqrt{\frac{|C|}{|N|(|N|-|C|)}} \right] \right\}^2 \\
&= \sum_{h\in\mathbf{h}} \left[\sum_{i\in C} \epsilon_h(i)\sqrt{w_h(i)} \left(\sqrt{\frac{|N|-|C|}{|N||C|}} + \sqrt{\frac{|C|}{|N|(|N|-|C|)}} \right) \right]^2 \\
&= \left(\frac{1}{|C|} + \frac{1}{|N|-|C|}\right) \sum_{h\in\mathbf{h}} \left[\sum_{i\in C} \epsilon_h(i)\sqrt{w_h(i)} \right]^2 \\
&= \text{RatioCut}\,(C)
\end{aligned}$$

As in the graph case, this result leads us to consider the following relaxed problems (5.9) and (5.10):

$$\text{Minimize } \Omega(f) \text{ subject to } \|f\|_2 = 1 \text{ and } f \in \text{Span}(\mathbf{1})^\perp \tag{5.9}$$

$$\text{Minimize } \Omega(f) \text{ subject to } \|f\|_2 = 1 \text{ and } f \in \text{Null}(\Delta)^\perp \tag{5.10}$$

As above (5.9) corresponds to the relaxation of the Min-RatioCut problem while (5.10) corresponds to a simpler but more restrictive problem that can be solved by considering the eigenspace of Δ which is associated with the smallest but non-null eigenvalue. The main difference with the graph case is a consequence of Proposition 23: when we replace $\text{Span}(\mathbf{1})^\perp$ by $\text{Null}(\Delta)^\perp$, we disregard the solutions corresponding to the independent components. Note that both problems are equivalent when the hypernode graph is strongly connected.

Based on this observation, an interesting question is whether we can consider the spectral clustering algorithm in the case of hypernode graphs and whether we can use it to solve real partitioning problems. This question is beyond the scope of the present work but should be definitely addressed in the future. Other promising research problems include whether we can generalize theoretical results such as the Cheeger's inequality for the case of hypernode graphs. In the next section, we discuss how we can connect the hypernode graph cuts to the notion of reduced signed graph presented in Section 3.1.2.

5.1.4 Relation with the signed graph cuts

Recall that a hypernode graph $\mathbf{h} = (V, E)$ is equivalent to a signed graph $\tilde{\mathbf{g}}$ where the weight between two arbitrary nodes u and v is given by:

$$W_{i,j} = \sum_{h \in \mathbf{h}} -\epsilon_h(i)\epsilon_h(j)\sqrt{w_h(i)}\sqrt{w_h(j)} \ .$$

The pairwise matrix W defines a notion of reduced signed graph as described in section 3.1.2. Moreover, we have shown in Proposition 13 that the smoothness measure $\Omega(f)$ of a function $f : N \to \mathbb{R}$ could be expressed directly on the reduced signed graph:

$$\Omega(f) = \underbrace{\sum_{\{i,j\} \in E} |W_{i,j}| \left[f(j) - \operatorname{sgn}(W_{i,j}) f(i)\right]^2}_{\xi_1(f)} + \underbrace{\sum_{i=1}^{n} \left[d(i) - \tilde{d}(i)\right] f(i)^2}_{\xi_2(f)} \ . \tag{3.3}$$

As discussed above, the first term $\xi_1(f)$ carries the common semantic of a signed graph according to the social balance theory. The second term $\xi_2(f)$ is harder to interpret in the general case, which limit our ability to reason directly on the signed graph.

However, when $f = \mathbf{1}_C$ with $C \subseteq N$, we have $\Omega(f) = \operatorname{Cut}(C)$ (see Equation 5.8) and Equation 3.3 becomes much easier to interpret since

Proposition 24. *For any set $C \subseteq V$, we have*

$$\operatorname{Cut}(C) = \sum_{\substack{i \in C \\ j \notin C}} W_{i,j} \tag{5.11}$$

Proof. This result can be obtained by taking $f = \mathbf{1}_C$ into Equation 3.3. However, for the sake of readability, we present here a shorter proof based

on the expression of the cut. Thus, we write for $C \subseteq V$,

$$\begin{aligned}
\text{Cut}\,(C) &= \sum_{h\in\mathbf{h}} \left[\sum_{i\in h\cap C} \sqrt{w_h(i)}\epsilon_h(i) \right]^2 \\
&= \sum_{h\in\mathbf{h}} \left[\sum_{i\in h\cap C} \sqrt{w_h(i)}\epsilon_h(i) \right] \left[\sum_{i\in h\cap C} \sqrt{w_h(i)}\epsilon_h(i) \right] \\
&= \sum_{h\in\mathbf{h}} \left[\sum_{i\in h\cap C} \sqrt{w_h(i)}\epsilon_h(i) \right] \left[-\sum_{j\in h\cap \overline{C}} \sqrt{w_h(j)}\epsilon_h(j) \right] \\
&\quad \text{(because of the Equilibrium Condition)} \\
&= \sum_{h\in\mathbf{h}} \sum_{\substack{i\in h\cap C \\ j\in h\cap\overline{C}}} -\epsilon_h(i)\epsilon_h(j)\sqrt{w_h(i)}\sqrt{w_h(j)} \\
&= \sum_{\substack{i\in C \\ j\in\overline{C}}} \sum_{h\in\mathbf{h}} -\epsilon_h(i)\epsilon_h(j)\sqrt{w_h(i)}\sqrt{w_h(j)} \\
&= \sum_{i\in C, j\notin C} W_{i,j}
\end{aligned}$$

□

We can rewrite Equation (5.11) as

$$\text{Cut}\,(C) = \underset{\tilde{\mathbf{g}}_+}{\text{Cut}}(C) - \underset{\tilde{\mathbf{g}}_-}{\text{Cut}}(C), \tag{5.12}$$

where $\text{Cut}_{\tilde{\mathbf{g}}_+}$ (resp. $\text{Cut}_{\tilde{\mathbf{g}}_-}$) is the classic graph cut defined on the positive part (resp. negative part) of $\tilde{\mathbf{g}}$. As a consequence, a cut on a hypernode graph can be expressed as a cut on its reduced signed graph $\tilde{\mathbf{g}}$.

Example 19. *(Example 18 continued)*
As a consequence of Proposition 24, the Min-Cut problem on hypernode graph $\mathbf{h}_{16}$ presented in Example 17 is fully equivalent to a signed Min-Cut problem on the reduced signed graph $\tilde{\mathbf{g}}_{16}$ depicted in Figure 5.6. For the sake of readability, we use the thickness of the edge lines to figure the magnitude of the weights. Note that a strong link exists in the signed graph between Z and H. The reason is that the structure does not encode our additional knowledge on these two nodes. In the previous chapter on skill rating, this knowledge was injected as a supervision term. Here, we use a similar strategy by adding the node constraint $Z \in C$ and $H \in C'$. As expected, the best cut on this graph leads to the clusters $C = \{3, 4, H\}$ (unskilled players and lazy helper) and $C' = \{1, 2, Z\}$ (skilled players and active helper).

Note that Proposition 24 allows incidentally to get more information about the signed graphs that corresponds to hypernode graphs. Indeed, let us recall

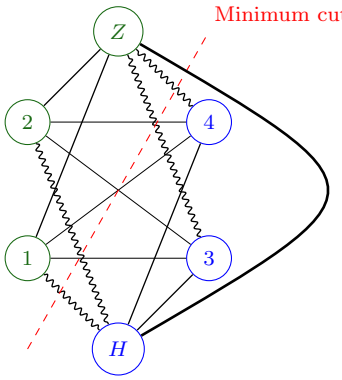


Figure 5.6: Reduced signed graph $\tilde{\mathbf{g}}_{16}$

that a signed graph is said to be *balanced* as soon as any cycle contains an even number of negative edge. We can show that:

Corollary 2. *The reduced signed graph of a hypernode graph is either a graph or an unbalanced signed graph.*

Proof. Let $\tilde{\mathbf{g}} = (N, E)$ be a reduced signed graph associated with hypernode graph $\mathbf{h}$. We assume that $\tilde{\mathbf{g}}$ has a least one negative edge and show that it is necessarily unbalanced. If $\tilde{\mathbf{g}}$ is balanced, a classic theorem (see Harary 1953) claims that we can split N in two non-empty sets N_1 and N_2 such that:

- All the edges between N_1 and N_2 are negative
- All the edges inside N_1 (resp. N_2) are positive

The intuition behind this result is easy to apprehend: we can assign a label in $\{-1, 1\}$ to each of the nodes in N by starting from an arbitrary node with label 1 and reversing the sign each time we traverse a negative edge. This simple procedure is guaranteed to work without conflicts because every cycle contains an even number of negative edge.

Consequently, and since we have at least one negative edge, we can write

$$\sum_{\substack{i \in N_1 \\ j \in N_2}} W_{i,j} < 0 \ . \qquad (*)$$

However, we know from Proposition 24 that

$$\sum_{\substack{i \in N_1 \\ j \in N_2}} W_{i,j} = \mathrm{Cut}\,(N_1) \geqslant 0 \ . \qquad (**)$$

(the cut-size is always positive or null as stated in Proposition 22)
Equations $(*)$ and $(**)$ are incompatible so $\tilde{\mathbf{g}}$ is necessarily unbalanced. □

Note that the question of whether a signed graph is balanced or not is of central importance in the signed graph theory. Indeed, a *balanced* graph is considered as stable from the perspective of the social balance theory since it does not contain any pathological patterns such as "A and B are ennemies but share C as a common friend". Corollary 2 allows us to claim that all the reduced signed graphs that are balanced are necessarily classic graphs. An interesting research problem would be to establish connections between a hypernode graph and the frustration index (measure of imbalance) of its reduced signed graph.

5.1.5 Algorithmical perspectives and partial results

In the past sections, we have defined the notion of hypernode graph cut and discussed its main properties. We have put our focus on the Min-Cut problem and highlighted several interesting problems that should be tackled in the future. In particular, we have reviewed the strong connection that exists between the hypernode graph cut and the cut on the reduced signed graph. In this section, we consider the Min-Cut problem from a purely algorithmical perspective and derive some results that will be helpful for future developments.

We first introduce the notion of *positive set* and *positive sequence*. We will use the following notation for $C_1, C_2 \subseteq N$,

$$d(C_1, C_2) = \sum_{i \in C_1, j \in C_2} W_{i,j}$$

Definition 11. *A* positive sequence *is a sequence of nodes* $(u_1, u_2, \dots)$ *such that:*

$$\forall i, \quad d(\{u_i\}, \{u_1, \dots, u_{i-1}\}) \geqslant 0$$

A positive set *is a set of nodes that can be ordered into a positive sequence.*

As an example, let us consider the hypernode graph h_7 recalled in Figure 5.7. $\{1, 3\}$ is a positive set since $d(\{1\}, \{3\}) = 1$. Conversely, $\{1, 2\}$ is not a positive set since $d(\{1\}, \{2\}) = -1$.

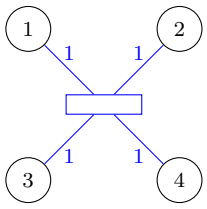


Figure 5.7: Hypernode graph h_7

We now show the following result

Proposition 25. *Let $C \subseteq N$ be a set of nodes. Then if C is not a positive set, we can build C' with $|C'| < |C|$ such that* $\mathrm{Cut}\,(C') < \mathrm{Cut}\,(C)$

Proof. Let us consider $C \subseteq N$ that is not a positive set and let $\ell = |C|$. We can find an ordering of C, $(i_1, \ldots, i_p, i_{p+1}, \ldots, i_\ell)$ such that for all $k \in [p{+}1, \ell]$

$$d(\{i_k\}, \{i_1, \ldots, i_p\}) < 0 \ .$$

As a consequence, we have

$$d(\{i_1, \ldots, i_p\}, \{i_{p+1}, \ldots, i_\ell\}) < 0 \ .$$

In the following, we define $C_1 = \{i_1, \ldots, i_p\}$ and $C_2 = \{i_{p+1}, \ldots, i_\ell\}$. We can write

$$\begin{aligned}
\mathrm{Cut}\,(C_1) &= d(C_1, \overline{C}) + d(C_1, C_2) \\
&= d(C_1, \overline{C}) + d(C_2, \overline{C}) - d(C_2, \overline{C}) + d(C_1, C_2) \\
&< d(C_1, \overline{C}) + d(C_2, \overline{C}) - d(C_2, \overline{C}) - d(C_1, C_2) \\
&< \mathrm{Cut}\,(C) - \mathrm{Cut}\,(C_2) \\
&< \mathrm{Cut}\,(C)\,, \quad \text{since a cut is always positive or null}
\end{aligned}$$

We have $|C_1| < |C|$, which concludes the proof. □

As a direct consequence of Proposition 25, we can state that the set of all nodes N is a *positive set*. Indeed, if it was not the case, then we could find a set C' such that $\mathrm{Cut}\,(C') < \mathrm{Cut}\,(N) = 0$. Due to a lack of time, we did not go further into this direction. However, we hope that these first results will help to design efficient Min-Cut algorithms.

5.2 Directed hypernode graphs

In Section 2.2, we introduced the notion of orientation function ϵ. Based on this definition, we can introduce formally the notion of *directed hypernode graph*:

Definition 12. *A* directed hypernode graph $(\mathbf{h}, \epsilon)$ *is an hypernode graph* $\mathbf{h}$ *together with an arbitrary orientation function* ϵ.

As in the graph case, the orientation allows to encode relations such as set causality and precedence relationships. However, as noted in Section 2.2.2, the Laplacian matrix is unable to keep track of the orientation. Indeed, the direction information contained in the gradient matrix G is lost as soon as we consider the symmetrized quantity G^TG.

Note that the exact same problem occurs in the graph case. We can even carry the reasoning one step further and notice that a symmetric matrix with a fully constrained diagonal lies in a space of dimension $\frac{n(n-1)}{2}$, which is necessarily limited when it comes to capture the $n(n-1)$ distinct ordered pairs of nodes of a directed graph. An interesting example is the case of the Laplacian matrix proposed by Zhou et al. (2005) and Chung (2005) for directed graphs. The reasoning is based on a notion of random walk formalized through the transition matrix P and its associated stationary law Π. The Laplacian introduced by Zhou et al. (2005) and Chung (2005) is symmetric and can be written as

$$\Delta = I - \frac{\Pi^{\frac{1}{2}} P \Pi^{-\frac{1}{2}} + \Pi^{-\frac{1}{2}} P^T \Pi^{\frac{1}{2}}}{2}$$

If $P = D^{-1}W$ and $\Pi = D$ (classic random walk), the Laplacian reduced to

$$\Delta = I - D^{-1/2} \left(\frac{W + W^T}{2} \right) D^{-1/2} \ ,$$

which is the classic normalized graph Laplacian defined on the undirected graph with adjacency matrix $\frac{W+W^T}{2}$. In order to circumvent this issue, recent works such as Boley et al. (2011) focus on the case of asymmetric Laplacians. This approach allows to capture more deeply the specificities of the directed graphs but also prevents us from working with positive semi-definite matrices. It remains an open question to extend these results to the case of directed hypernode graphs.

An other interesting line of research is to bring the theory of directed hypernode graphs closer to the theory of *directed hypergraphs*, a model notably popularized by Gallo et al. (1993). *Directed hypergraphs* (also called *AND/OR hypergraphs*) are implicative structures that generalize over directed graphs and were first introduced in order to formalize the notion of functional dependency between objects.

The main idea is to represent the AND implication $x_1 \wedge x_2 \rightarrow x_3$ by a directed link between the sets $\{x_1, x_2\}$ and $\{x_3\}$. Similarly, a directed link between $\{x_3\}$ and $\{x_1, x_2\}$ can be used to represent an OR implication $x_3 \rightarrow x_1 \vee x_2$.

A directed hyperedge is defined as a directed link between two sets of nodes respectively denoted as the *tail* and the *head* of the hyperedge. When the tail

(resp. the head) consists in a single node, the directed hyperedge is called OR hyperedge (resp. AND hyperedge). A *directed hypergraph* is defined as a set of nodes $N = (1, \ldots, n)$ together with a set of directed hyperedges $H = (h_1, \ldots, h_p)$. A simple example is presented in Figure 5.8.

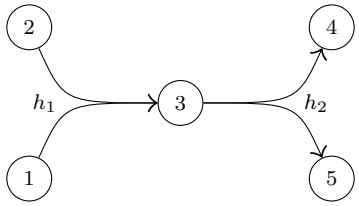


Figure 5.8: AND hyperedge h_1 followed by OR hyperedge h_2. We have $\text{tail}\,(h_1) = \{1, 2\}$ and $\text{head}\,(h_1) = \{3\}$.

A particular case arises when the directed hypergraph only contains AND hyperedges. This restriction makes sense in multiple contexts where the goal is to represent only many-to-one implications. Among many other examples, may be cited the case of propositional logic (see notably Gallo et al. 1998). The reader can refer to Ausiello et al. (2001) for a detailed list of applications. Note that a weighted version of these objects has also been defined in Cambini et al. (1997) by attaching individual weights $\mu_h(v)$ for each node v participating in a hyperedge h. By convention, the quantity $\mu_h(v)$ will be negative (resp. positive) if v belong to $\text{tail}\,(h)$ (resp. $\text{head}\,(h)$).

In terms of structure, directed hypernode graphs are very close to the directed hypergraphs. The individual weights $\mu_h(i)$ can be regarded as equivalent to the terms $\epsilon_h(i)\sqrt{w_h(i)}$. In our framework, we choose to relax some constraints (arbitrary number of nodes in the tails and the heads) and add a new one on the weights (the equilibrium condition):

$$\forall h, \quad \sum_{i \in \text{tail}(h)} \mu_i(h) + \sum_{i \in \text{head}(h)} \mu_i(h) = 0 \quad .$$

Note that our notion of set similarity can be brought closer to the notion of two way implication between sets (or set equivalence). Thus, we can consider our (undirected) hyperedges as the superposition of two directed hyperedges. As far as we know, the class of directed hypergraphs has not been studied from the machine learning point of view and no attempt was made to define a spectral framework for these objects.

It remains an open question to know whether our discrete analysis framework designed for hypernode graphs can integrate with the theory of directed hypergraphs and whether it can help to define new tools for the directed and undirected cases. In this context, a promising line of research is to investigate the notion of directed hypergraph flows as defined by Cambini

et al. (1997). Similarly to the graph case, a directed hypergraph flow is defined as a real-valued function $g : H \to \mathbb{R}$ that should satisfy two main properties. The first one called *feasibility* expresses the fact that the value of g on a directed hyperedge h should be bounded by a specific constant called *upper capacity of the hyperedge*. The second one is the *conservativity* and is associated with a specific demand function $b : N \to \mathbb{R}$. It states that for any node $i \in N$, the flow g must satisfy

$$\sum_{h \backslash i \in \text{head}(h)} \mu_i(h) g(h) - \sum_{h \backslash i \in \text{tail}(h)} \mu_i(h) g(h) = b(i) \ . \tag{5.13}$$

In the graph case, a strong link exists between cuts and flows through the Min-Cut Max-Flow theorem. It remains an open question to extend this result to the case of hypernode graphs.

Note that if we denote by G the gradient matrix as defined in Section 2.2.2, we can rewrite Equation (5.13) as $G^T g = b$. In (non-discrete) vector analysis, the *divergence* operator is defined as minus gradient operator. Analogously, we can define the discrete divergence operator div as the operator described by the matrix $-G^T$, which allows us to rewrite again Equation (5.13) as $\text{div}(g) + b = 0$, which is the shape of a standard local conservation law. Thus, our discrete analysis framework allow us to express very naturally the conservativity constraints from Cambini et al. (1997). We did not investigate further this line of research since it was out of the concerns of the present work but we strongly believe that future research will allow the fields of directed hypergraphs and directed hypernode graphs to cross-fertilize each other. In particular, it remains an open question to extend the Min-Cut Max-Flow theorem to the hypernode graph case.

5.3 An algebraical interpretation of hypernode graphs

We now discuss some interesting properties concerning the classes of graph Laplacians and graph kernels. In particular, we show that the class of hypernode graph Laplacians can be seen as a convex relaxation of the class of graph kernels. We also show that the class of hypernode graph Laplacians is equal to the graph hypernode graph kernels.

5.3.1 The classes of Graph Kernels and Graph Laplacians

Let us consider the class

$$\mathcal{L}(n) = \{M \in \mathbb{R}^{n \times n} \mid M = M^T, \ \mathbf{1} \in \text{Null}(M), \ \text{extradiag}\,(M) \leqslant 0\} \ , \tag{5.14}$$

where $\text{extradiag}\,(M)$ is the matrix M with the diagonal removed.

Proposition 26. *The class $\mathcal{L}(n)$ is the set of unnormalized graph Laplacians.*

Proof. For $M \in \mathcal{L}(n)$, let us consider $W = -\text{extradiag}\,(M)$. By definition of $\mathcal{L}(n)$, $W \geqslant 0$, thus W is a graph adjacency matrix. Let us denote by D the degree matrix of W. For every $i \in N$, $W_{i,i} = 0$ and we have

$$D_{i,i} = \sum_{\substack{1 \leqslant j \leqslant n \\ j \neq i}} W_{i,j} = - \sum_{\substack{1 \leqslant j \leqslant n \\ j \neq i}} M_{i,j} \ .$$

Since $M\mathbf{1} = 0$ by definition of $\mathcal{L}(n)$, then, for every $i \in N$, $\sum_{1 \leqslant j \leqslant n} M_{i,j} = 0$, and thus $D_{i,i} = M_{i,i}$. That is $M = D - W$, and therefore M is a graph Laplacian.

Conversely if $M = D - W$ is a graph Laplacian, then $M = M^T$ and $M\mathbf{1} = D\mathbf{1} - W\mathbf{1} = 0$ by definition of the degree matrix. Finally, $\text{extradiag}\,(M) = -W \leqslant 0$, which concludes the proof. □

The *class $\mathcal{K}(n)$ of graph kernels* is the set of all matrices which are pseudoinverse of some matrix in $\mathcal{L}(n)$. Since the pseudoinverse operator is involutive, we can write equivalently $\mathcal{K}(n) = \{M \in \mathbb{R}^{n \times n} \mid M^\dagger \in \mathcal{L}(n)\}$. Then, the classes $\mathcal{L}(n)$ and $\mathcal{K}(n)$ satisfy

Proposition 27.

(i) the set $\mathcal{L}(n)$ of graph Laplacians is closed by convex linear combination but is not closed by linear combination,

(ii) the set $\mathcal{K}(n)$ of graph kernels is not closed by convex linear combination,

(iii) the sets $\mathcal{L}(n)$ and $\mathcal{K}(n)$ are not closed under pseudoinverse, and

(iv) $\mathcal{L}(n) \cap \mathcal{K}(n) \neq \varnothing$

Proof. (i) Let us consider $\Delta_1, \Delta_2 \in \mathcal{L}(n)$ and let us denote by W_1 and W_2 the corresponding adjacency matrices. For every $\alpha \in [0, 1]$, $W = \alpha W_1 + (1 - \alpha)W_2$ is symmetric and satisfies $W \geqslant 0$ and W is therefore the adjacency matrix of an undirected graph. Then $\Delta = \alpha\Delta_1 + (1-\alpha)\Delta_2$ is the unnormalized Laplacian associated with W and thus $\Delta \in \mathcal{L}(n)$. Hence, $\mathcal{L}(n)$ is closed by convex linear combination.

When $\alpha \notin [0, 1]$, then W, defined as above, can have negative weights, thus $\Delta \notin \mathcal{L}(n)$. Therefore, $\mathcal{L}(n)$ is not closed by linear combination.

(ii) Let us consider the matrices Δ_1 and Δ_2 in $\mathcal{L}(n)$ defined by

$$\Delta_1 = \begin{pmatrix} 2 & 0 & -1 & -1 \\ 0 & 1 & -1 & 0 \\ -1 & -1 & 2 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \quad \Delta_2 = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}.$$

The matrices $\Delta_1^\dagger$ and $\Delta_2^\dagger$ are in $\mathcal{K}(n)$, but, let us consider the matrix $\frac{1}{2}(\Delta_1^\dagger + \Delta_2^\dagger)$, its pseudoinverse is

$$\left[\frac{1}{2}(\Delta_1^\dagger + \Delta_2^\dagger)\right]^\dagger = \frac{1}{13}\begin{pmatrix} 16 & 2 & -12 & -6 \\ 2 & 10 & -8 & -4 \\ -12 & -8 & 22 & -2 \\ -6 & -4 & -2 & 12 \end{pmatrix},$$

which is not in $\mathcal{L}(n)$ since some extra-diagonal elements are non-negative. Hence $\mathcal{K}(n)$ is not closed by convex linear combination.

(iii) Let us consider the Laplacian $\Delta_1 \in \mathcal{L}(n)$ from (ii). Its pseudoinverse is

$$\Delta_1^\dagger = \frac{1}{8}\begin{pmatrix} 3 & -3 & -1 & 1 \\ -3 & 7 & 1 & -5 \\ -1 & 1 & 3 & -3 \\ 1 & -5 & -3 & 7 \end{pmatrix},$$

which is not in $\mathcal{L}(n)$ since some extra-diagonal elements are non-negative. Hence $\mathcal{L}(n)$ is not closed by pseudoinverse.

The class $\mathcal{K}(n)$ is not closed by pseudoinverse. Indeed, let us consider $\Delta_1^\dagger$ in $\mathcal{K}(n)$, its pseudoinverse is $(\Delta_1^\dagger)^\dagger = \Delta_1$. And, Δ_1 does not belong to $\mathcal{K}(n)$ since its pseudoinverse $\Delta_1^\dagger \notin \mathcal{L}(n)$.

(iv) Let us consider the matrices Δ in $\mathcal{L}(n)$ and $\Delta^\dagger$ in $\mathcal{K}(n)$ defined by

$$\Delta = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 3 & -2 \\ 0 & -2 & 2 \end{pmatrix} \qquad \Delta^\dagger = \frac{1}{6}\begin{pmatrix} 3 & -1 & -2 \\ -1 & 1 & 0 \\ -2 & 0 & 2 \end{pmatrix}.$$

We notice that $\Delta^\dagger \in \mathcal{L}(n)$, which concludes the proof: $\mathcal{L}(n) \cap \mathcal{K}(n) \neq \varnothing$. □

5.3.2 The class of Hypernode graph Laplacians

As shown in Proposition 7 from Section 2.2, the class of hypernode graph Laplacian is the class $\mathcal{H}(n)$

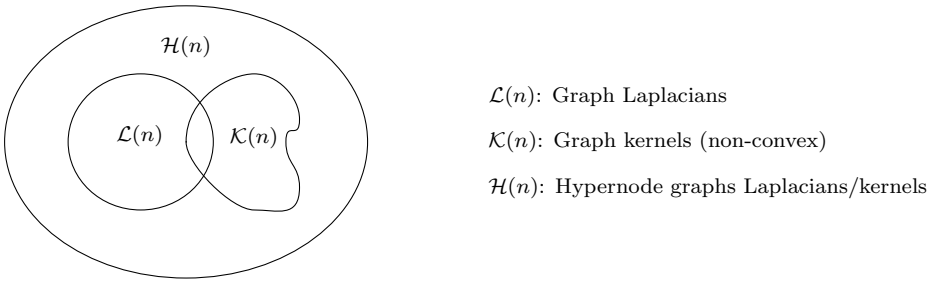


Figure 5.9: Schematic representation of the classes $\mathcal{L}(n)$, $\mathcal{K}(n)$ and $\mathcal{H}(n)$

$$\mathcal{H}(n) = \{M \in \mathbb{R}^{n\times n} \mid M = M^T,\ \mathbf{1} \in \mathrm{Null}(M),\ M \succeq 0\} \ . \tag{5.15}$$

We review the properties of $\mathcal{H}(n)$ in the next proposition.

Proposition 28.

(i) the class $\mathcal{H}(n)$ is closed under pseudoinverse,

(ii) the class $\mathcal{H}(n)$ contains the class of graph Laplacians $\mathcal{L}(n)$ and the class of graph kernels $\mathcal{K}(n)$,

(iii) the class $\mathcal{H}(n)$ is closed by convex linear combination but is not closed by linear combination.

Proof. (i) The pseudoinverse operation preserves the symmetry and the semidefiniteness property since it does not modify the sign of the eigenvalues. Moreover, for every real-valued symmetric matrix M, $\mathrm{Null}(M^\dagger) = \mathrm{Null}(M)$ so $\mathbf{1} \in \mathrm{Null}(M^\dagger)$.

(ii) $\mathcal{H}(n)$ contains $\mathcal{L}(n)$ because graph Laplacians are positive semidefinite. From (i), $\mathcal{H}(n)$ is closed by pseudoinverse then it contains also

$$\mathcal{K}(n) = \{M \in \mathbb{R}^{n\times n} \mid M^\dagger \in \mathcal{L}(n)\} \ .$$

(iii) Let us consider $M_1, M_2 \in \mathcal{H}(n)$ and a real α, and let us denote by M_α the matrix $\alpha M_1 + (1-\alpha)M_2$. M_α is symmetric and $M_\alpha \mathbf{1} = 0$. Now, when $\alpha \in [0,1]$, for every $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{x}^T M_\alpha \mathbf{x} = \alpha \mathbf{x}^T M_1 \mathbf{x} + (1-\alpha)\mathbf{x}^T M_2 \mathbf{x} \geqslant 0 \ ,$$

because $M_1 \succeq 0$ and $M_2 \succeq 0$. That is $M_\alpha \succeq 0$, hence $\mathcal{H}(n)$ is closed by convex linear combination. However, it is easy to see that there exist $M_1, M_2, \mathbf{x}$ and $\alpha \in \mathbb{R}$ such that $\mathbf{x}^T M_\alpha \mathbf{x} < 0$. Thus, $\mathcal{H}(n)$ is not closed by linear combination, which concludes the proof. □

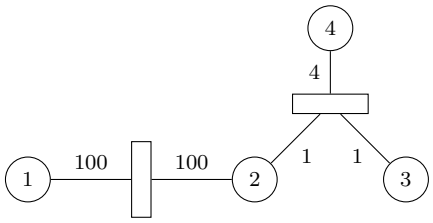


Figure 5.10: Hypernode graph $\mathbf{h}_{17}$

5.3.3 A convex hull conjecture and an intermediate class $\mathcal{F}(n)$

We have shown in the previous section that the class $\mathcal{H}(n)$ of hypernode graph Laplacians (and kernels) was a convex relaxation of the class of graph kernels $\mathcal{K}(n)$ that also contains the class of graph Laplacians $\mathcal{L}(n)$. We propose the following conjecture:

Conjecture 1. *$\mathcal{H}(n)$ is the convex hull of $\mathcal{K}(n)$.*

Note that this result does not hold in the case of connected graphs. Indeed, let us consider the space of connected graph kernels $\mathcal{K}^+(n) \subsetneq \mathcal{K}(n)$. If we denote by $\mathcal{F}(n)$ the set of *hypernode graph kernels such that d^2 is a pseudometric (i.e., d^2 is positive, symmetric and satisfies the triangle equality)*, then the following property holds:

Proposition 29. *$\mathcal{F}(n)$ is convex and $\mathcal{K}^+(n) \subsetneq \mathcal{F}(n) \subsetneq \mathcal{H}(n)$.*

Proof. We have $\mathcal{F}(n) \subset \mathcal{H}(n)$ by definition and the inclusion is strict since $\mathcal{H}(n)$ contains the kernels of the disconnected graphs for which the triangle inequality does not hold (see Section 2.1.2). Moreover, because of Proposition 4, we know that d^2 is always a metric in the case of a connected graph. Consequently, we also have $\mathcal{K}^+(n) \subset \mathcal{F}(n)$. In order to prove that $\mathcal{K}^+(n) \neq \mathcal{F}(n)$, let us consider the hypernode graph $\mathbf{h}_{17}$ depicted in Figure 5.10. $\mathbf{h}_{17}$ consists of two hyperedges: the first one h_1 is a classic graph edge that links 1 and 2 while the second one h_2 links the set $\{2,3\}$ to the singleton $\{4\}$. We can observe that the hypernode graph is "dominated" by h_1 because of its high weight but cannot reduced to a graph since the pairwise weight $W_{2,3}$ is negatice (equal to -1). We can verify that d^2 is a pseudometric (actually, even a metric) on the nodeset $= \{1,2,3,4\}$.

It remains to show that $\mathcal{F}(n)$ is convex. Let us consider, K_1 and K_2 in $\mathcal{F}(n)$. For all $\alpha \in [0,1]$, $K = \alpha K_1 + (1-\alpha)K_2$ is in $\mathcal{H}(n)$ so the quantity

$$d^1_K(i,j) = \sqrt{K_{i,i} + K_{j,j} - 2K_{i,j}} \ ,$$

defines a pseudometric on $N = \{1,\ldots,n\}$ as stated in Proposition 16 (i.e., it is positive, symmetric and satisfies the triangle equality). Consequently,

the quantity d^2 defined for all (i, j) by

$$d^2_K(i,j) = K_{i,i} + K_{j,j} - 2K_{i,j} \ ,$$

is also positive and symmetric. We have to show that it satisfies the triangle equality. Let us consider $(i, j, k) \in \mathbb{R}^3$, we have:

$$\begin{aligned} d^2_K(i,k) &= \alpha d^2_{K_1}(i,k) + (1-\alpha) d^2_{K_2}(i,k) \\ &\leqslant \alpha \left(d^2_{K_1}(i,j) + d^2_{K_1}(j,k) \right) + (1-\alpha) \left(d^2_{K_2}(i,j) + d^2_{K_2}(j,k) \right) \\ &\quad \text{since } K_1 \text{ and } K_2 \text{ are in } \mathcal{F}(n) \\ &\leqslant d^2_K(i,j) + d^2_K(j,k) \ , \end{aligned}$$

which concludes the proof. □

5.3.4 A Riemanian geometry for strongly connected hypernode graphs

In this section, we consider the case of strongly connected hypernode graphs as described in Section 3.2. It is worth noting that all connected graphs are also strongly connected hypernode graphs. The goal of this section is to show that the class of Laplacians of strongly connected hypernode graphs can be embedded with a complete Riemannian structure. The class of Laplacians that corresponds to the strongly connected hypernode graphs is

$$\mathcal{H}^+(n) = \{M \in \mathbb{R}^{n \times n} \mid M = M^T, \ \text{Null}(M) = \text{Span}(\mathbf{1}), \ M \succeq 0\} \ .$$

As mentioned above, the class $\mathcal{H}(n)$ is closed by convex linear combination. This is not the case for the class $\mathcal{H}^+(n)$ which is a strict subspace of $\mathcal{H}(n)$. The geodesic in the Euclidean space $\mathbb{R}^{n \times n}$ is a "straight line" but the Euclidean geometry does not fit the class $\mathcal{H}^+(n)$. This is illustrated with a simple metaphor for $\mathbb{R}^2$ in Figure 5.11 below where the class $\mathcal{H}^+(n)$ is a curved space which can be seen as a boundary space of $\mathcal{H}(n)$.

Thus, our goal is to exhibit a Riemannian geometry in which $\mathcal{H}^+(n)$ is geodesic complete. Geodesic completeness generalizes the notion of closure by convex linear combination. The notion of shortest route in a curved space derives from the notion of metric tensor that generalizes the inner product of the Euclidean space. At any point of a given manifold, a tangent space can be defined and the metric tensor defines an inner product for all tangent spaces, which leads to the notion of Riemannian metric. In order to define the Riemannian geometry over $\mathcal{H}^+(n)$, we use the property that, for every Δ in $\mathcal{H}^+(n)$, we have $\text{Null}(\Delta) = \text{Span}(\mathbf{1})$. We also define a smooth mapping between $\mathcal{H}^+(n)$ and the space of symmetric positive

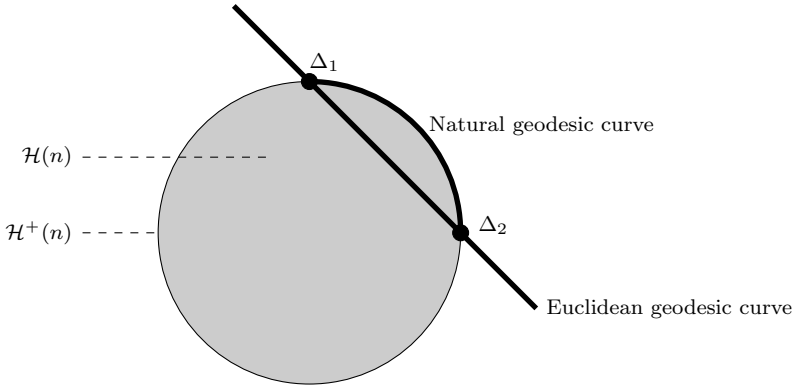


Figure 5.11: Schematic representation of convex space $\mathcal{H}(n)$ and non-convex space $\mathcal{H}^+(n)$. Euclidean geometry of $\mathbb{R}^{n\times n}$ is not suited for $\mathcal{H}(n)$ and $\mathcal{H}^+(n)$.

definite matrices $\mathcal{P}_{n-1} = \{R \in \mathbb{R}^{n-1\times n-1} \mid M = M^T,\ M \succ 0\}$ which can be embedded in a Riemannian geometry.

Formally, as for every $\Delta \in \mathcal{H}^+(n)$, we have $\text{Null}(\Delta) = \text{Span}(\mathbf{1})$, we deduce that the restriction of Δ to the vector space $\text{Span}(\mathbf{1})^\perp$ is positive definite, where $\text{Span}(\mathbf{1})^\perp$ denotes the vector space orthogonal to $\text{Span}(\mathbf{1})$. It is important to note that the space $\text{Span}(\mathbf{1})^\perp$ does not depend on Δ. In order to define the mapping between $\mathcal{P}_{n-1}$ and $\mathcal{H}^+(n)$, let us denote by $\mathcal{B} = (\mathbf{e}_1, \ldots, \mathbf{e}_n)$ the canonical basis of $\mathbb{R}^{n\times n}$ (the i-th component of $\mathbf{e}_i$ is one, the other components are zeros) and let us consider the orthogonal basis $\mathcal{B}' = (\mathbf{1}, \mathbf{e}'_2, \ldots, \mathbf{e}'_n)$ where $(\mathbf{e}'_2, \ldots, \mathbf{e}'_n)$ is an orthogonal basis of $\text{Span}(\mathbf{1})^\perp$. Let us now consider P, the change-of-coordinates operator $\mathcal{B} \to \mathcal{B}'$. We define the mapping f between $\mathcal{P}_{n-1}$ and $\mathcal{H}^+(n)$ by

$$f : A \to P^T \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix} P \ . \tag{5.16}$$

We show some important properties of f that will allow us to transfer the Riemannian structure from $\mathcal{P}_{n-1}$ to $\mathcal{H}^+(n)$.

Proposition 30. *f is a C^∞-diffeomorphism between $\mathcal{P}_{n-1}$ and $\mathcal{H}^+(n)$. For any $A \in \mathcal{P}_{n-1}$, we have $f(A^{-1}) = f(A)^\dagger$ and $\text{Tr}(f(A)) = \text{Tr}(A)$.*

Proof. Let us consider $A \in \mathcal{P}_{n-1}$, $\text{Null}(f(A)) = \text{Span}(\mathbf{1})$ by construction. $f(A)$ is symmetric positive semidefinite since A is symmetric positive definite so we have $f(A) \in \mathcal{H}^+(n)$. Conversely, let us consider $M \in \mathcal{H}^+(n)$ and $U = PMP^T$. Since $\mathbf{1} \in \text{Null}(M)$, we can write

$$U\mathbf{e}_1 = PMP^T\mathbf{e}_1 = PM\mathbf{1} = \mathbf{0} \ . \tag{5.17}$$

Thus, U can be written as

$$U = \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & Q \end{pmatrix} \ , \tag{5.18}$$

with $Q \in \mathbb{R}^{n-1 \times n-1}$. Moreover $Q \in \mathcal{P}_{n-1}$ since the spectrum of Q is equal to the spectrum of M without the null eigenvalue associated with $\mathbf{1}$. Since $\text{Null}(M) = \text{Span}(\mathbf{1})$, the spectrum of Q is strictly positive. Finally, we have $M = f(Q)$ and f is a bijection between $\mathcal{P}_{n-1}$ and $\mathcal{H}^+(n)$. f and f^{-1} are infinitely differentiable as change-of-coordinates operators.

It remains to show the properties of f. First, let us consider $A \in \mathcal{P}_{n-1}$, then

$$\begin{aligned} f(A)^\dagger &= \left(P^T \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix} P \right)^\dagger \\ &= P^\dagger \left(P^T \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix} \right)^\dagger \quad \text{since } P \text{ is orthogonal} \\ &= P^\dagger \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix}^\dagger (P^T)^\dagger \quad \text{since } P^T \text{ is orthogonal} \\ &= P^T \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A^{-1} \end{pmatrix} P \\ &= f(A^{-1}) \ . \end{aligned}$$

Second, the property over traces can be proved by

$$\text{Tr}(f(A)) = \text{Tr}\left(P^T \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix} P \right) = \text{Tr}\left(PP^T \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & A \end{pmatrix} \right) = \text{Tr}(A) \ .$$

□

We will use Proposition 30 to transfer the geometrical structure of $\mathcal{P}_{n-1}$ to $\mathcal{H}^+(n)$. But, before let us review some results concerning the geometrical structure of $\mathcal{P}_{n-1}$. The set $\mathcal{P}_{n-1}$ is a space of matrices which can also be viewed as a space of multivariate Gaussian distributions with null mean (via the covariance matrix). These two views can be used to derive a Riemannian metric on $\mathcal{P}_{n-1}$. The first one is based on a pure geometrical approach and the second one is based on Fisher information theory. It is worth noting that both approaches produce the same metric tensor (see for instance Bonnabel and Sepulchre 2009) defined by, for every $A \in \mathcal{P}_{n-1}$,

$$g_A^{\mathcal{P}_{n-1}}(D_1, D_2) = \text{Tr}(D_1 A^{-1} D_2 A^{-1}) \ , \tag{5.19}$$

where D_1 and D_2 are in the tangent space at point A in $\mathcal{P}_{n-1}$ denoted by $\mathcal{T}_{\mathcal{P}_{n-1}}(A)$. A distance $d_{\mathcal{P}_{n-1}}$ over $\mathcal{P}_{n-1}$ can be derived from the definition of the metric tensor (5.19) by, for every A_1, A_2 in $\mathcal{P}_{n-1}$,

$$d_{\mathcal{P}_{n-1}}(A_1, A_2) = \| \log(A_1^{-1/2} A_2 A_1^{-1/2}) \|_2 \ . \tag{5.20}$$

and, also the metric tensor defined in (5.19) allows defining a geodesic curve from a matrix A_1 in $\mathcal{P}_{n-1}$ to a matrix A_2 in $\mathcal{P}_{n-1}$ by

$$\gamma_{A_1,A_2}^{\mathcal{P}_{n-1}}(\alpha) = A_1^{1/2} \exp(\alpha \log(A_1^{-1/2} A_2 A_1^{-1/2})) A_1^{1/2} \quad .$$

It should be noted the geodesic from A_1 to A_2 is not, in general, equal the geodesic from A_2 to A_1. As noted in Bonnabel and Sepulchre (2009), $\mathcal{P}_{n-1}$ embedded with this natural geometry is geodesic complete, i.e., every geodesic can be extended to a maximal geodesic defined for $\alpha \in \mathbb{R}$. This property allows to use efficient short-step methods to solve complex optimization problems.

We are ready to define a Riemannian geometry over $\mathcal{H}^+(n)$ using Proposition 30 and the Riemannian geometry over $\mathcal{P}_{n-1}$. First, the mapping f between $\mathcal{P}_{n-1}$ and $\mathcal{H}^+(n)$ defined in Equation (5.16) can be extended to a C^∞-diffeomorphism between Span($\mathcal{P}_{n-1}$) and Span($\mathcal{H}^+(n)$). This allows to define the tangent space at point M in $\mathcal{H}^+(n)$, denoted by $\mathcal{T}_{\mathcal{H}^+(n)}(M)$, by

$$\mathcal{T}_{\mathcal{H}^+(n)}(M) = f(\mathcal{T}_{\mathcal{P}_{n-1}}(f^{-1}(M))) \tag{5.21}$$

and to define the metric tensor for M in $\mathcal{H}^+(n)$, denoted by $g_M^{\mathcal{H}^+(n)}$. Let D_1 and D_2 be in the tangent space $\mathcal{T}_{\mathcal{H}^+(n)}(M)$. Then,

$$g_M^{\mathcal{H}^+(n)}(D_1, D_2) = g_{f^{-1}(M)}^{\mathcal{P}_{n-1}}(f^{-1}(D_1), f^{-1}(D_2)) \quad . \tag{5.22}$$

Proposition 31. *$g_M^{\mathcal{H}^+(n)}(D_1, D_2)$ can be expressed in function of M, D_1 and D_2 using the formula*

$$g_M^{\mathcal{H}^+(n)}(D_1, D_2) = \mathrm{Tr}(D_1 M^\dagger D_2 M^\dagger) \quad .$$

Proof. First, note that the extension of the mapping f into a mapping between Span($\mathcal{P}_{n-1}$) and Span($\mathcal{H}^+(n)$) can still be expressed using Equa-

tion (5.16). Thus, we get from Equation (5.22),

$$
\begin{aligned}
g_M^{\mathcal{H}^+(n)}(D_1, D_2) &= g_{f^{-1}(M)}^{\mathcal{P}_{n-1}}(f^{-1}(D_1), f^{-1}(D_2)) \\
&= \mathrm{Tr}\left[f^{-1}(D_1)(f^{-1}(M))^{-1} f^{-1}(D_2)(f^{-1}(M))^{-1}\right] \quad \text{(see Eq. (5.19))} \\
&= \mathrm{Tr}\left[P \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & f^{-1}(D_1)(f^{-1}(M))^{-1} f^{-1}(D_2)(f^{-1}(M))^{-1} \end{pmatrix} P^T\right] \\
&= \mathrm{Tr}\left[P \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & f^{-1}(D_1) \end{pmatrix} P^T P \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & (f^{-1}(M))^{-1} \end{pmatrix} P^T \right. \\
&\qquad \left. P \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & f^{-1}(D_2) \end{pmatrix} P^T P \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & (f^{-1}(M))^{-1} \end{pmatrix} P^T\right] \\
&= \mathrm{Tr}\left[(f \circ f^{-1}(D_1))(f \circ f^{-1}(M))^\dagger (f \circ f^{-1}(D_2))(f \circ f^{-1}(M))^\dagger\right] \\
&\quad \text{(see Prop. 30)} \\
&= \mathrm{Tr}(D_1 M^\dagger D_2 M^\dagger) \ .
\end{aligned}
$$

□

We can now define the Riemannian structure of $\mathcal{H}^+(n)$. The Riemannian distance over $\mathcal{H}^+(n)$ is defined for every M_1, M_2 in $\mathcal{H}^+(n)$ by

$$d_{\mathcal{H}^+(n)}(M_1, M_2) = d_{\mathcal{P}_{n-1}}(f^{-1}(M_1), f^{-1}(M_2))$$

and the geodesic curve from M_1 to M_2 in $\mathcal{H}^+(n)$ is defined by

$$\gamma_{M_1,M_2}^{\mathcal{H}^+(n)}(\alpha) = f(\gamma_{f^{-1}(M_1),f^{-1}(M_2)}^{\mathcal{P}_{n-1}}(\alpha)) \ .$$

Proposition 32. *The Riemannian distance over $\mathcal{H}^+(n)$ between M_1, M_2 in $\mathcal{H}^+(n)$ can be expressed as*

$$d_{\mathcal{H}^+(n)}(M_1, M_2) = \| \log\left(\mathbf{1}\mathbf{1}^T + (M_1^\dagger)^{1/2} M_2 (M_1^\dagger)^{1/2}\right) \|_2$$

and the Riemannian distance is invariant by pseudoinverse.

Proof. Let M_1, M_2 in $\mathcal{H}^+(n)$, then

$$
\begin{aligned}
d_{\mathcal{H}^+(n)}(M_1, M_2) &= d_{\mathcal{P}_{n-1}}(f^{-1}(M_1), f^{-1}(M_2)) \quad \text{by definition of } d_{\mathcal{H}^+(n)} \\
&= \left\| \begin{pmatrix} 0 & 0 \\ 0 & \log(f^{-1}(M_1)^{-1/2} f^{-1}(M_2) f^{-1}(M_1)^{-1/2}) \end{pmatrix} \right\|_2 \quad \text{by Equation (5.20)} \\
&= \left\| \log \left[\begin{pmatrix} 1 & 0 \\ 0 & f^{-1}(M_1)^{-1/2} f^{-1}(M_2) f^{-1}(M_1)^{-1/2} \end{pmatrix} \right] \right\|_2 \\
&= \left\| P^T \log \left[\begin{pmatrix} 1 & 0 \\ 0 & f^{-1}(M_1)^{-1/2} f^{-1}(M_2) f^{-1}(M_1)^{-1/2} \end{pmatrix} \right] P \right\|_2 \quad \text{since } P \text{ is orthogonal} \\
&= \left\| \log \left[P^T \begin{pmatrix} 1 & 0 \\ 0 & f^{-1}(M_1)^{-1/2} f^{-1}(M_2) f^{-1}(M_1)^{-1/2} \end{pmatrix} P \right] \right\|_2 \\
&= \left\| \log \left[\mathbf{1}\mathbf{1}^T + P^T \begin{pmatrix} 0 & 0 \\ 0 & f^{-1}(M_1)^{-1/2} \end{pmatrix} P P^T \begin{pmatrix} 0 & 0 \\ 0 & f^{-1}(M_2) \end{pmatrix} P P^T \begin{pmatrix} 0 & 0 \\ 0 & f^{-1}(M_1)^{-1/2} \end{pmatrix} P \right] \right\|_2 \\
&= \left\| \log \left(\mathbf{1}\mathbf{1}^T + (M_1^\dagger)^{1/2} M_2 (M_1^\dagger)^{1/2} \right) \right\|_2 \quad .
\end{aligned}
$$

The distance $d_{\mathcal{H}^+(n)}$ is invariant by pseudoinverse since the distance $d_{\mathcal{P}_{n-1}}$ is invariant by matrix inversion (see for example Bonnabel and Sepulchre 2009), which concludes the proof. □

It should be noted that, since f is a C^∞-diffeomorphism, the geodesic curves of $\mathcal{H}^+(n)$ can also be extended for $\alpha \in \mathbb{R}$. Thus, $\mathcal{H}^+(n)$ embedded with our new metric is geodesic complete. Moreover, because of the pseudoinverse invariance property, the Riemannian distance between two graph Laplacians in $\mathcal{H}^+(n)$ is equal to the Riemannian distance between the corresponding two graph kernels. The Riemannian structure of $\mathcal{P}_{n-1}$ with the Riemannian metric, also called the natural metric, has been used in many efficient applications in various fields (object detection in radar processing, bio medical imaging, kernel optimization). The Riemannian structure over $\mathcal{H}^+(n)$ introduced in this section should open new algorithmic perspectives such as Weiszfeld's algorithm for mean and median computation, usage of complete geodesics to express dissimilarities for graph kernels. This line of research remains open for future work.

Chapter 6

Conclusion

In the present work, we have introduced a new model called hypernode graph that generalizes classic graph and that allows to handle binary relations between groups in networks. We have defined a spectral theory allowing to model homophilic relations between groups assuming an additive model for individual valuations. In particular, we have defined the notion of hypernode graph Laplacian that generalizes the classic graph Laplacian operator. Similarly to the graph case, the Laplacian matrix is positive semidefinite, which allows us to leverage many results and algorithms originating from graph theory.

As expected, many classic results evolve when we consider the hypernode graph extension. Several properties that hold in the graph case become untrue in our general framework. Some others need to be modified and refined in order to remain consistent with our definitions. In the past chapters, we have studied the most important aspects of the hypernode graph learning framework. We have shown that working with the extended Laplacian operator provides a strict gain of expressiveness and allows to encode complex notions of smoothness. Moreover, we have established important connections with the theory of signed graphs through the fundamental concept of hypernode graph reduction. We have also shown that our model allows to encode dominance relations by adding auxiliary nodes. As a proof of concept, we have used hypernode graphs to model multiplayer games with known outcomes and obtained competitive results on the problem of skill rating (compared to the state-of-the-art).

Most of the difficulties surrounding the study of hypernode graph stems for the data structure itself. Indeed, hyperedges are based on the concept of group relations and, therefore, are fundamentally different in nature from the pairwise relations that allows usually to define distances, paths and walks. Moreover, the pieces of information brought by different hyperedges

are often interdependent.

In the past chapters, we have often relied on the concept of reduced signed graph mentioned above in order to work at the level of nodes. However, it can be observed that the semantic connection between signed graphs and hypernode graphs still remains unclear as discussed in Section 3.1.2. We believe that a better understanding of this connection will bring new perspectives for the theory of hypernode graphs and will allow us to better apprehend the concept of hypernode graph diffusion defined in Section 3.3.2.

More generally, many theoretical results on hypernode graphs remains to be discovered. In the previous chapters, we have presented several promising research directions. Among others, we can cite the problems of generalizing the Max-Flow Min-Cut theorem and defining new algorithmic approaches to solve the node valuation problems. Many other topics that are beyond the scope of this thesis are also worthy of considering. For instance, an interesting idea is to consider the hypernode graph extension from the perspective of information theory. Indeed, relationships between sets bring only partial information about the individuals. Therefore, interesting connections could be made with the problems of compression and anonymizations in networks. Finally, we hope that the hypernode graph model will open the way to solving new learning problems in networks.

Appendices

Appendix A

Multigraph learning with Hypernode graphs

Chapter abstract In this appendix, we consider the problem of combining different graphs on the same set of nodes. We assume that the input graphs are connected, which allow us to make a link between commute-time distance and Laplacian matrices as stated in Proposition 5. We split the combination process in two parts: first, we use the Laplacian matrix to associate graphs with what we call *embedded vector spaces*; second, we merge these spaces into a new vector space associated to a new positive semi-definite kernel which we ensure to be a hypernode graph kernel.

A.1 Combining Graphs through Euclidean Embedded Spaces

Let us consider k input graphs $\mathbf{g}_1, \ldots, \mathbf{g}_k$ on the same node set N and a set $N_L \subsetneq N$. We assume that we are given a labeling function $y_L : N_L \to \{-1, 1\}$. We denote by n the number of nodes in N and by n_L the number of nodes in N_L. Moreover, we will assume that $n_L \ll n$. Our objective is to combine the k graphs into a hypernode graph that agrees as much as possible with y_L. We will show that the combined hypernode graph outperforms significantly the original graphs when it come to classify the nodes in $N \backslash N_L$. In the next subsection, we introduce the notion of *embedded Euclidean space* that serves as a basis for our combination algorithm.

A.1.1 Embedded vector spaces

Given a graph $\mathbf{g} = (N, E)$, an *embedded vector space* is a tuple $(\mathcal{E}, K(\cdot,\cdot), \phi)$ where

- $\mathcal{E}$ is a finite-dimensional vector space,
- $K(\cdot,\cdot)$ is a positive semidefinite kernel on $\mathcal{E} \times \mathcal{E}$, and
- ϕ is an injective function, called *node map*, from N to $\mathcal{E}$.

The Gram matrix $K = (K(\phi(i), \phi(j)))_{i,j\in N}$ is denoted as the *kernel matrix* of the embedded vector space.

An embedded Euclidean space satisfies *the commute-time property* for $\mathbf{g}$ if, for every pair of nodes $i, j \in N$, we have $K(\phi(i), \phi(j)) = \Delta^{\dagger}_{i,j}$. In this case, we have $\|\phi(i) - \phi(j)\|^2 = c_{i,j}/\operatorname{Vol}(\mathbf{g})$, where $\|.\|$ is the Euclidean norm associated with $K(\cdot,\cdot)$, and $c_{i,j}$ is the commute-time distance between nodes i and j. A simple choice of embedded space that satisfies the commute-time property is

$$\mathcal{V}(\mathbf{g}) = \left(\mathbb{R}^n,\ K(x,y) = x^T\Delta^{\dagger}y,\ \phi : i \to \mathbf{e}_i\right) \ .$$

Indeed, by construction, we have $K(\phi(i), \phi(j)) = \mathbf{e}_i^T\Delta^{\dagger}\mathbf{e}_j = \Delta^{\dagger}_{i,j}$.

Note that we can reduce the inner dimension of $\mathcal{V}(\mathbf{g})$ while preserving *approximately* the commute-time property. Indeed, we can consider the singular value decomposition $\Delta = V\Lambda V^T$ where V is an unitary matrix and where Λ is the diagonal matrix of the eigenvalues of Δ. We have for all $x, y \in \mathbb{R}^n$,

$$K(x,y) = x^T\Delta^{\dagger}y = x^TV\Lambda^{\dagger}V^Ty \ .$$

The diagonal matrix $\Lambda^{\dagger}$ contains the pseudo-inverses of the eigenvalues ($1/x$ if $x \neq 0$, 0 otherwise). We can observe that, the higher the eigenvalue λ_i in Λ, the lesser its contribution to the kernel K of $\mathcal{V}(\mathbf{g})$. Therefore, we can nullify the biggest eigenvalues of Δ without too much of an impact on the commute-time property. The main interest of this operation is to reduce the rank of the kernel matrix, which will lead to a sturdier representation of the input graph $\mathbf{g}$. More information about this dimensionality reduction method can be found in Fouss et al. (2005).

A.1.2 Combining embedded Euclidean spaces.

Using the definitions of Section A.1.1, we associate to each graph $\mathbf{g}_i$, the space

$$\mathcal{V}(\mathbf{g}_i) = (\mathbb{R}^n, K_i(\cdot,\cdot)_i, \phi : i \to \mathbf{e}_i) \ ,$$

that satisfies the commute-time property on graph $\mathbf{g}_i$ Our objective is to combine the different embedded space into a new one called *K-merged space* and defined as

$$\mathcal{V}_{\text{merged}} = (\mathbb{R}^n, K, \phi : i \to \mathbf{e}_i) \ ,$$

where $K = F(K_1, \ldots, K_k)$ for some function F. It should be noted that, when the original spaces satisfy the commute-time property (no dimension reduction), the kernels K_i are equals to the original graph kernels $\Delta_i^\dagger$ and we do not need to compute the full embedding. K allows us to combine the geometries of the different embedded spaces and we will try to choose F such that the resulting geometry agrees as much as possible with the partial labeling y_L. We also want K to be a hypernode graph kernel (i.e., $K \in \mathcal{H}(n)$), which will allow us to use hypernode graph learning algorithms (and preserve the semantic of the output). Note that an alternate goal would be to look for a graph kernel (i.e., $K \in \mathcal{K}(n)$) but this goal is much harder to achieve because of the pathological topology of $\mathcal{K}(n)$ (non-convex, characterization through the Moore-Penrose pseudo-inverse).

A.1.3 Convex Linear Combination

A simple choice for the combination function F is the linear combination operator:

$$K_{\text{LIN}} = F_{\text{LIN}}(K_1, \ldots, K_k) = \sum_{i=1}^{k} w_i K_i, \tag{A.1}$$

with $\forall 1 \leqslant i \leqslant k, w_i \geqslant 0$, and $\sum_{i=1}^{k} w_i = 1$. In this case, the Euclidean distance in the merged space defined by K_{LIN} is the weighted sum of the square roots of the commute-time distances in the original graphs (recall that the original embedded spaces satisfy the commute-time property). As a consequence of Proposition 27, K_{LIN} is a hypernode graph kernel and the matrix $\Delta_{\text{LIN}} = K_{\text{LIN}}^\dagger$ is a hypernode graph Laplacian.

As mentioned above, we consider a partial supervision y_L to drive our combination process and find the best set of values for $w_1, \ldots, w_k$. Argyriou et al. (2006) describe a framework for the convex linear combination based on the minimization of the following objective function:

$$E_\gamma(\Delta_{\text{LIN}}) = \min_f \left\{ f^T \Delta_{\text{LIN}} f + \gamma \mathcal{L}(f, y_L) \right\} \tag{A.2}$$

where $\mathcal{L}(f, y_L)$ is an arbitrary loss term. Thus, the minimization problem is a semi-supervised problem defined as a trade-off between a loss term $\mathcal{L}(f, y_L)$ and a smoothness regularization term $f^T \Delta_{\text{LIN}} f$. $E_\gamma(\Delta_{\text{LIN}})$ is the minimal energy that can be obtained with the smoothness operator Δ_{LIN}. Argyriou et al. (2006) leverage the RKHS framework to simplify the computation

of E_γ: with reasonable assumptions on the loss function $\mathcal{L}$, we can apply the *representer theorem* to state that the optimal point f of (A.2) can be expressed as:

$$f = \sum_{i=1}^{n_L} c_i (K_{\text{LIN}})_i$$

where $(K_{\text{LIN}})_i$ is the i-th column of K_{LIN}. Thus, the energy can be computed using the simpler dual formulation:

$$E_\gamma = -\min_c \left\{ \frac{1}{4\gamma} c^T (K_{\text{LIN}})_L c + V^*(c, y_L) \right\} \tag{A.3}$$

where $\mathcal{L}^* = \sup_{\lambda \in \mathbb{R}} (\lambda c - \mathcal{L}(y, \lambda))$ and $(K_{\text{LIN}})_L$ is the submatrix of K_{LIN} corresponding to the nodes in N_L. This dual optimization problem only involves a $n_L \times n_L$ matrix with $n_L \ll n$.

Leveraging former results from Argyriou et al. (2005), they finally propose an algorithm to reduce the search of the minimum of $E_\gamma(\Delta_{\text{LIN}})$ to a sequence of line-search steps. A major drawback of this method is that we have to choose carefully the parameter γ (trade-off between regularization and loss) in order to get a meaningful energy E_γ.

As an alternative of this method, we consider a non-parametrical objective function cv_svm(K_L, y_L) defined as the cross-validation error on the set of labeled example of an SVM classifier trained with kernel K_L. It should be noted that the framework of Argyriou et al. (2006) that allows to replace the exhaustive search by a sequence of line search do not apply to our objective function. Thus we will consider an exhaustive search of the parameters of the linear combination.

A.1.4 Sigmoid Combination

We also consider the sigmoid combination of kernels defined by:

$$K_{\text{SIG}}(i, j) = 1/(1 + \exp(-(K_{\text{LIN}}(i, j))/\sigma)) \tag{A.4}$$

It has been observed (see Von Luxburg et al. 2010, part 4) that the sigmoid can be used to improve the quality of the commute time distance (as a topological indicator) for highly connected graphs because of its "normalization" effect. Indeed, the main idea is to normalize kernel values in $[0, 1]$. This idea was leveraged in L. et al. (2007) in order to design an efficient kernel clustering algorithm on a single graph.

Let us suppose that the w_i are fixed and let us denote by K_{SIG} the Gram matrix of the combined kernel. The situation is more intricate because the matrix $K^\dagger_{\text{SIG}}$ may not be positive semi-definite and may not satisfy

$K^{\dagger}_{\mathrm{SIG}}\mathbf{1} = 0$. Thus it is not in general a hypernode graph kernel. Thus, the solution is to define a "proxy" matrix in the space of smoothness operators. It must be close to $K^{\dagger}_{\mathrm{SIG}}$ and positive semi-definite. Moreover, it must satisfy the constant gauge property. To do this given as input the matrix $K^{\dagger}_{\mathrm{SIG}}$, we propose the *flip method* defined by:

1. Modify the diagonal terms of $K^{\dagger}_{\mathrm{SIG}}$ in order to have null sum for the rows and columns (or equivalently, $\mathbf{1} \in \mathrm{Null}(K^{\dagger}_{\mathrm{SIG}})$)
2. Compute the eigenvalues of the resulting matrix, and flip the negative eigenvalues (replace λ by $-\lambda$) to get a positive semi-definite matrix.

The resulting matrix Δ_{SIG} is a hypernode graph Laplacian and is considered as the final output of the sigmoid combination.

It should be noted that when the values $K_{\mathrm{LIN}}(i,j))/\sigma$ are small, we observe that $\|K^{\dagger}_{\mathrm{SIG}}\mathbf{1}\|$ is close to 0 (sum of columns/rows). Note that the power series expansion of the sigmoid function is $\frac{1}{1+\exp(-x)} \sim 1/2 + x/4$, which allows us to show that the sums of the rows of K_{SIG} should be close to $N/2$; therefore, from Schmidt and Trenkler (2001), the sums of the rows of $K^{\dagger}_{\mathrm{SIG}}$ should be close to $2/N$ (also observed empirically). Also, it should be noted that we do not use the usual shift trick which shifts the spectrum by minimal eigenvalue $\lambda_{\min}$: $\Delta \leftarrow \Delta + \lambda_{\min} I$ because we want to preserve the nullspace of the matrix..

We tune the parameters $\sigma, w_1, \ldots, w_k$ similarly to the case of the convex linear combination. We consider the objective functions $E_\gamma(\Delta_{\mathrm{SIG}})$ and cv_svm and search for the best parameter values.

A.1.5 Combination Algorithm

We summarize our combination method in Algorithm 3. It should be noted that in the experiments we do not consider the dimensionality reduction mentioned in line 3. The parameter tuning methods of line 6 have been described above. It should be noted that in line 7, for the two types of combination, the output of our algorithm is a smoothness operator Δ_{LIN} or Δ_{SIG} allowing to compute the smoothness of any real valued node function f with the regularization term $f^T \Delta_{\mathrm{LIN}} f$ or $f^T \Delta_{\mathrm{SIG}} f$.

A.2 Experiments

We now present some experimental results for our combination method. We first describe the datasets and the graph construction process that we consider in the experiments. Then, we detail the implementation of Algorithm 3

Algorithm 3 Combining embedded spaces for graphs.

Require: graphs $\mathbf{g}_1, \ldots, \mathbf{g}_k$ on a node set N; labeled sample $N_L \subset N$;

1: **for** each graph $\mathbf{g}_i$ **do**
2: Compute the embedded space $\mathcal{V}(\mathbf{g}_i) = (\mathcal{E}_i, K_i, \phi_i)$
3: [Opt.] reduce the dimensionality: $K_i \leftarrow K_i'$
4: **end for**
5: Consider the parametric combination $K = F(K_1, \ldots, K_k)$ ($F = F_{\text{LIN}}$ or F_{SIG})
6: Tune the parameters of F using one of the objective functions $\{E_\gamma, \text{cv\_svm}\}$
7: **return** A hypernode graph Laplacian Δ computed from K (pseudo-inversion + additional flip trick for the sigmoid)

and the experimental setting. Last, we apply graph-based semi-supervised learning algorithms and compare results obtained using the Laplacian of every graph, using the hypernode graph Laplacians Δ_{LIN} and Δ_{SIG}.

A.2.1 Datasets

We consider here similarity graphs defined from vectorial data or categorical data and graphs given by network data. For every dataset, we identify or build two graphs $\mathbf{g}_1$ and $\mathbf{g}_2$ that are given as input of Algorithm 3.

The datasets used are summarized in Table A.1. For the similarity graphs, we choose UCI datasets with both numerical and categorical attributes. For every UCI dataset, we build the graph $\mathbf{g}_1$ as the *k-nearest neighbor graph* (k-nn) based on the Euclidean distance on normalized numerical attributes: every node n in the graph represents a data point; two nodes i and j are connected if j is among the k-nearest neighbors of i. As the k-nn relation might not be symmetric, a common choice to make the final graph undirected is by ignoring directions (if $\hat{W}$ is the non symmetric adjacency matrix built from the k-nn algorithm, we consider the symmetric adjacency matrix $W = (\max(\hat{W}_{i,j}, \hat{W^T}_{i,j}))_{i,j}$). We also build the graph $\mathbf{g}_2$ using the categorical attributes. For every node i and j, we let

$$W_{ij} = \sum_{a \in \mathcal{A}} \frac{\delta_{(a(i)=a(j))}}{\mathcal{N}_a(a(i))} \ ,$$

where $\mathcal{A}$ is the set of the categorical attributes; $a(i)$ is the value of the categorical attribute $a \in \mathcal{A}$ for the node i; $\delta_{(x=y)}$ is the Kronecker delta; $\mathcal{N}_a(x)$ is the number of times attribute a has value x in the whole set (common values bring less information). This measure based on the work of Zhou et al. (2006) defines the weight adjacency matrix of the graph $\mathbf{g}_2$.

We also consider the network datasets WebKB and IMDB (prodco). The IMDB dataset contains two graphs $\mathbf{g}_1$ and $\mathbf{g}_2$ over a set of movies. For the WebKB dataset, the two graphs $\mathbf{g}_1$ and $\mathbf{g}_2$ are defined over a set of Web pages using respectively the hyperlinks and the co-citation links. See http://netkit-srl.sourceforge.net/data.html for more details.

For the similarity graphs with numerical attributes to be connected, we choose k such that the graph has only one connected component. For the similarity graphs with categorical attributes and for the networks to be connected, we adopt the teleporting random walk approach of Page et al. (1999): we consider if needed a small jumping probability (0.05) and adapt the adjacency matrices accordingly.

For every graph, we compute the graph Laplacian. Graph kernels are normalized by their Frobenius norm as proposed in Argyriou et al. (2006).

Dataset	**Class attribute**	**Size**
Statlog heart	Presence of disease	270
Credit approval	Approval	690
Horse colic	Surgical lesion	368
Flags	Religion Catholic, Oth. Christians vs others	194
Adult (excerpt)	Income $> 50K$	1400
WebKB	Page type: Student vs others	1477
IMDB	Blockbuster	1441

Table A.1: UCI and network datasets, size and class label.

A.2.2 Experimental setting

Because we want to show that our combination method is independent of the learning task, in order to evaluate our combination method, we consider different performance indicators:

- The efficiency of two popular semi supervised learning (SSL) algorithms described respectively in Zhu et al. (2003) (*ZGL*) and Zhou et al. (2005) (*ZHS*) for the semi-supervised problem described in Equation (A.2)
- The compatibility measure between Δ and the labeling function y, given by the *quality* ratio

$$\mathcal{Q}(\Delta, y) = \frac{\Omega(y)}{2\|G\|_F^2} \ ,$$

  where $\|.\|_F$ denotes the Frobenius norm: $\|G\|_F^2 = \mathrm{Tr}(G^T G) = \mathrm{Tr}(\Delta)$.

In each experiment and for each pair of graphs $(\mathbf{g}_1, \mathbf{g}_2)$ we repeat 12 times the following protocol:

(i) Randomly sample unlabeled data preserving the class proportions

(ii) For each objective function cv_svm and E_γ apply Algorithm 3 and output two smoothness operators Δ_{LIN} and Δ_{SIG} (linear and sigmoid combination). We denote by $\Delta_{\text{LIN}(\text{E}_\gamma)}$, $\Delta_{\text{SIG}(\text{E}_\gamma)}$, $\Delta_{\text{LIN}(\text{cv\_svm})}$, $\Delta_{\text{SIG}(\text{cv\_svm})}$ the outcomes of Algorithm 3 depending on the combination operator and on the objective function. We use a basic implementation of the algorithm that performs an exhaustive search over the parameters. At the same time and for each performance indicator, we compute the best possible linear and sigmoid combinations evaluated with fully labeled graphs. This step allows us to add some optimal baselines $\Delta_{\text{LIN}(\text{opt})}$, $\Delta_{\text{SIG}(\text{opt})}$.

(iii) Run the different performance indicators *ZGL*, *ZHS* and quality on each input graph $\mathbf{g}_1$ and $\mathbf{g}_2$ and on the combined objects.

The value γ of E_γ has been manually tuned to 10. $\Delta_{\text{LIN}(\text{E}_\gamma)}$ is equivalent to the algorithm proposed in Argyriou et al. (2006) (for two input graphs, it reduces to a classic line search over the values of E_γ).

A.2.3 Experimental results

We first consider a fixed ratio of labeled examples chosen to be of $\frac{n_L}{n} = 0.3$, thus 70% of nodes are unlabeled. Experimental results are shown in Tables A.1 and A.3 for *ZGL* and quality. Results for *ZHS* are very similar to the results for *ZGL* and therefore we do not report them. We will discuss and compare a combination method (sigmoid or linear) embedded with an objective function (E_γ or cv_svm) for a specific task (*ZGL*, *ZHS* or quality) with the two original graphs. We will also discuss whether we are close to the optimal combination.

We observe that the performance of Δ_{LIN} for *ZGL* is usually equivalent to the best of the two input graphs. This confirms the results obtained in previous works on linear combination of kernels (see for instance Argyriou et al. (2006)). We should note that the objective function cv_svm is slightly better than the energy-based function E_γ. We also observe that, in both cases, our combination method does not allow to obtain the accuracy of the optimal linear combination.

Let us now consider the sigmoid combination. It is worth noting that the accuracy of the SSL algorithm using Δ_{SIG} is better than the accuracy of the SSL algorithms on the two input graphs. To the best of our knowledge, it is the first time that a combination method allows to outperform the best of the input graphs.

Dataset	MR	Δ_1	Δ_2	LIN (opt)	LIN (E_γ)	LIN (cv_svm)	SIG (opt)	SIG (E_γ)	SIG (cv_svm)
Credit appr.	0.45	0.29 ± 0.02	0.42 ± 0.02	0.24 ± 0.02	0.41 ± 0.03	0.27 ± 0.02	0.14 ± 0.01	0.15 ± 0.02	0.15 ± 0.02
Flags	0.48	0.38 ± 0.04	0.20 ± 0.03	0.20 ± 0.02	0.20 ± 0.03	0.22 ± 0.04	0.14 ± 0.02	0.16 ± 0.03	0.17 ± 0.03
Stat. Heart	0.44	0.28 ± 0.02	0.29 ± 0.03	0.24 ± 0.02	0.29 ± 0.03	0.28 ± 0.02	0.16 ± 0.02	0.18 ± 0.02	0.20 ± 0.02
IMDB	0.43	0.27 ± 0.01	0.43 ± 0.0	0.27 ± 0.01	0.33 ± 0.07	0.27 ± 0.01	0.21 ± 0.01	0.22 ± 0.01	0.22 ± 0.01
WebKB	0.42	0.42 ± 0.0	0.15 ± 0.02	0.11 ± 0.01	0.42 ± 0.0	0.11 ± 0.01	0.08 ± 0.0	0.08 ± 0.0	0.08 ± 0.0
Adult	0.26	0.23 ± 0.01	0.26 ± 0.0	0.22 ± 0.01	0.24 ± 0.02	0.25 ± 0.01	0.20 ± 0.01	0.20 ± 0.01	0.22 ± 0.02
Horse	0.36	0.35 ± 0.02	0.36 ± 0.0	0.33 ± 0.02	0.36 ± 0.0	0.35 ± 0.02	0.16 ± 0.02	0.18 ± 0.03	0.16 ± 0.02

Figure A.1: Mean and standard deviation of error rates for the *ZGL* (Zhu et al. (2003)) algorithm for an unlabeled proportion $1 - \frac{n_L}{n}$ equal to 0.7. Column MR corresponds to the majority vote rule; Δ_i are results for *ZGL* on graph $\mathbf{g}_i$ without combination; opt is the optimal value for the combination method ; other comlums give results for the two combination methods with the two objective functions.

We obtain similar observations for the quality measure and the *ZHS* algorithm. It should be noted that *ZHS* also depends on a learning parameter that is usually hard to tune. In all experiments, the sigmoid combination showed improved sturdiness with respect to the choice of this parameter (compared to the original graphs $\mathbf{g}_1, \mathbf{g}_2$ and to the linear combination).

Now, let us vary the proportion $1 - \frac{n_L}{n}$ of unlabeled examples. Figure A.2 shows the evolution of the accuracy of the *ZGL* algorithm when the unlabeled proportion varies between 0.3 and 0.95 on the Credit data set. It can be noted than the accuracy using the objective functions E_γ or cv_svm and the sigmoid combination are closed to the accuracy of the optimal combination. We can also note that are very good up to a unlabeled proportion of 0.9 and still good for 0.95.

Table A.3 reports the results where we can again observe that the sigmoid combination is better, even if we are far from the ideal value. The reason for this "gap" is that the task is harder in this case: the true minimum is computed using the complete knowledge of the labels.

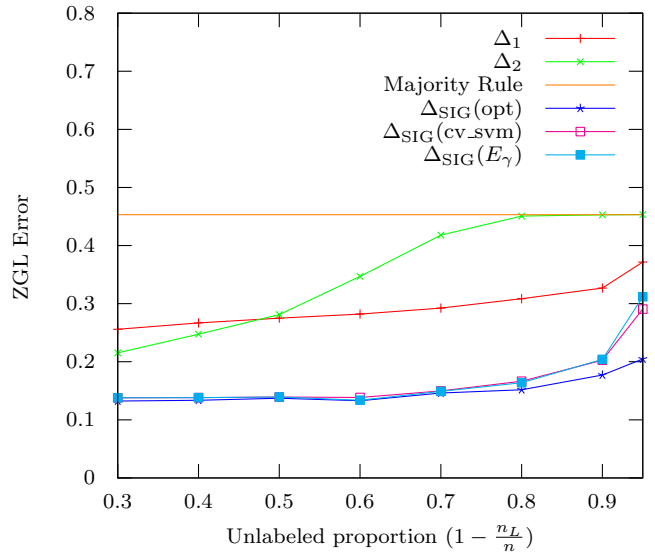


Figure A.2: Evolution of the error rate of the *ZGL* algorithm for the sigmoid combination on the Credit approval dataset depending of the ration of unlabeled examples.

Dataset	MR	Δ_1	Δ_2	LIN (opt)	LIN (E_γ)	LIN (cv_svm)	SIG (opt)	SIG (E_γ)	SIG (cv_svm)
Credit appr.	0.45	0.28	0.44	0.28	0.44 ± 0.0	0.35 ± 0.03	0.08	0.27 ± 0.0	0.27 ± 0.01
Flags	0.48	0.39	0.45	0.39	0.45 ± 0.0	0.44 ± 0.0	0.04	0.22 ± 0.0	0.26 ± 0.0
Stat. Heart	0.44	0.36	0.41	0.36	0.39 ± 0.02	0.36 ± 0.01	0.13	0.28 ± 0.02	0.26 ± 0.05
IMDB	0.43	0.43	0.48	0.43	0.45 ± 0.02	0.43 ± 0.0	0.31	0.32 ± 0.0	0.32 ± 0.0
WebKB	0.42	0.49	0.32	0.32	0.49 ± 0.0	0.38 ± 0.01	0.13	0.13 ± 0.0	0.14 ± 0.01
Adult	0.26	0.30	0.35	0.30	0.31 ± 0.01	0.33 ± 0.01	0.21	0.27 ± 0.0	0.31 ± 0.04
Horse	0.36	0.41	0.42	0.41	0.42 ± 0.0	0.42 ± 0.0	0.21	0.24 ± 0.04	0.21 ± 0.01

Figure A.3: Quality measure $\mathcal{Q}(\Delta)$ of the resulting Laplacians for $1 - \frac{n_L}{n} = 0.7$.

Bibliography

Sameer Agarwal, Jongwoo Lim, Lihi Zelnik-Manor, Pietro Perona, David Kriegman, and Serge Belongie. Beyond pairwise clustering. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005 (CVPR-05)*, volume 2, pages 838–845. IEEE, 2005.

Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher Order Learning with Graphs. In *Proceedings of the 23rd International conference on Machine learning (ICML-06)*, pages 17–24, 2006.

Andreas Argyriou, Charles A Micchelli, and Massimiliano Pontil. Learning convex combinations of continuously parameterized basic kernels. In *Learning Theory*, pages 338–352. Springer, 2005.

Andreas Argyriou, Mark Herbster, and Massimiliano Pontil. Combining Graph Laplacians for Semi-Supervised Learning. In *Proceedings of the 20th conference Annual Conference on Neural Information Processing Systems (NIPS-05)*, pages 67–74, Cambridge, MA, 2006. MIT Press.

Giorgio Ausiello, Paolo Giulio Franciosa, and Daniele Frigioni. Directed Hypergraphs: Problems, Algorithmic Results, and a Novel Decremental Approach. In *Proceedings of the 7th Italian Conference on Theoretical Computer Science*, ICTCS '01, pages 312–327, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42672-8.

Claude Berge. *Hypergraphs: combinatorics of finite sets*. North-Holland Mathematical Library. Elsevier, Burlington, MA, 1989.

Daniel Boley, Gyan Ranjan, and Zhi-Li Zhang. Commute times for a directed graph using an asymmetric Laplacian. *Linear Algebra and its Applications*, 435(2):224–242, 2011.

Marianna Bolla. Spectra, euclidean representations and clusterings of hypergraphs. *Discrete Mathematics*, 117(1):19–39, 1993.

Silvere Bonnabel and Rodolphe Sepulchre. Riemannian metric and geomet-

ric mean for positive semidefinite matrices of fixed rank. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1055–1070, 2009.

Xavier Bresson and Arthur D Szlam. Total Variation, Cheeger Cuts. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1039–1046, 2010.

Niv Buchbinder, M Feldman, J Naor, and R Schwartz. Submodular Maximization with Cardinality Constraints. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-14)*, pages 1433–1452, 2014. doi: 10.1137/1.9781611973402.106.

Mark Burgin. Interpretations of Negative Probabilities. 2010.

Jie Cai and Michael Strube. End-to-end coreference resolution via hypergraph partitioning. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING-10)*, pages 143–151, 2010.

Riccardo Cambini, Giorgio Gallo, and MariaGrazia Scutella. Flows on hypergraphs. *Mathematical Programming*, 78(2):195–217, 1997.

Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 1996.

Kai-Yang Chiang, Joyce Jiyoung Whang, and Inderjit S Dhillon. Scalable clustering of signed networks using balance normalized cut. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 615–624. ACM, 2012.

F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

Fan Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

Harris Drucker, Chris JC Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In *Proceedings of the 10th Conference on Neural Information Processing Systems (NIPS-96)*, volume 9, pages 155–161. Morgan Kaufmann Publishers, 1996.

Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2): 248–264, April 1972.

Arpad Emrick Elo. *The Rating of Chess Players, Past and Present*. Arco Publishing, 1978.

F. Fouss, A. Pirotte, and M. Saerens. A novel way of computing similarities between nodes of a graph, with application to collaborative recom-

mendation. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI-05)*, pages 550–556, 2005.

Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *Knowledge and Data Engineering, IEEE Transactions on*, 19(3):355–369, 2007.

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the 2nd European Conference On Computational Learning Theory (EuroCOLT-95)*, pages 23–37. Springer, 1995.

Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42 (2-3):177–201, 1993.

Giorgio Gallo, Claudio Gentile, Daniele Pretolani, and Gabriella Rago. Max Horn SAT and the minimum cut problem in directed hypergraphs. *Mathematical Programming*, 80(2):213–237, 1998.

John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the Future*, 2012.

MX Goemans and DP Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 1(212):1–27, 1995.

Andrew B Goldberg, Xiaojin Zhu, and Stephen J Wright. Dissimilarity in graph-based semi-supervised classification. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS-07)*, pages 155–162, 2007.

DM Greig, BT Porteous, and Allan H Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 271–279, 1989.

Scott Hamilton. PythonSkills: Implementation of the TrueSkill, Glicko and Elo Ranking Algorithms. https://pypi.python.org/pypi/skills, 2012.

Frank Harary. On the notion of balance of a signed graph. *The Michigan Mathematical Journal*, 2(2):143–146, 1953. doi: 10.1307/mmj/1028989917.

Matthias Hein, Simon Setzer, Leonardo Jost, and Syama Sundar Rangapuram. The Total Variation on Hypergraphs - Learning on Hypergraphs

Revisited. In *Proceedings of the 27th Conference on Neural Information Processing Systems (NIPS-13)*, pages 2427–2435, 2013.

Ralf Herbrich, Tom Minka, and Thore Graepel. TrueSkill™: A Bayesian Skill Rating System. In *Proceedings of the 20th Conference on Neural Information Processing Systems (NIPS-06)*, pages 569–576, 2006.

Mark Herbster. Exploiting cluster-structure to predict the labeling of a graph. *Algorithmic Learning Theory*, 2008. doi: 10.1007/978-3-540-87987-9_9.

Yao Ping Hou. Bounds for the least Laplacian eigenvalue of a signed graph. *Acta Mathematica Sinica*, 21(4):955–960, 2005.

Kun Huang. *Maximum Flow Problem in Assembly Manufacturing Networks*. North Carolina State University, 2011.

Satoru Iwata and James B Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-09)*, pages 1230–1237. Society for Industrial and Applied Mathematics, 2009.

David R Karger. Global min-cuts in RNC, and other ramifications of a simple min-out algorithm. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 21–30. Society for Industrial and Applied Mathematics, 1993.

Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and Cellular Networks. *PLoS Computational Biology*, 5(5), May 2009.

Douglas J. Klein and M. Randić. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, 1993.

Y. Koren, L. Carmel, and D. Harel. ACE: a fast multiscale eigenvectors computation for drawing huge graphs. In *Proceedings of the 8th IEEE Symposium on Information Visualization (INFOVIS 2002)*, pages 137–144, 2002. doi: 10.1109/INFVIS.2002.1173159.

Nathan Krislock, Jérôme Malick, and Frédéric Roupin. Improved semidefinite bounding procedure for solving Max-Cut problems to optimality. *Mathematical Programming*, 143(1-2):61–86, 2014. ISSN 0025-5610. doi: 10.1007/s10107-012-0594-z.

Jérôme Kunegis, Stephan Schmidt, Andreas Lommatzsch, Jürgen Lerner, Ernesto William De Luca, and Sahin Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM-10)*, volume 10, pages 559–559, 2010.

Yen L., F. Fouss, C. Decaestecker, P. Francq, and M. Saerens. Graph Nodes Clustering Based on the Commute-Time Kernel. In *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-07)*, volume 4426, pages 1037–1045, 2007.

Jan Lasek, Zoltán Szlávik, and Sandjai Bhulai. The predictive power of ranking systems in association football. *International Journal of Applied Pattern Recognition*, 1(1):27–46, 2013.

Heungsub Lee. Python implementation of Elo: A rating system for chess tournaments. https://pypi.python.org/pypi/elo/0.1.dev, 2013a.

Heungsub Lee. Python implementation of TrueSkill: The video game rating system. http://trueskill.org/, 2013b.

L Lovász, M Grötschel, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Berlin Heidelberg, 1993.

László Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art*, pages 235–257. Springer, 1983.

Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001.

GL Nemhauser, LA Wolsey, and ML Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14:265–294, 1978. doi: 10.1007/BF01588971.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, 1999.

Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.

Roger Penrose. A generalized inverse for matrices. In *Proceedings of the Cambridge Philosophical Society*, volume 51, pages 406–413. Cambridge Univ Press, 1955.

Jean-Claude Picard and Maurice Queyranne. On the structure of all minimum cuts in a network and applications. In V.J. Rayward-Smith, editor, *Combinatorial Optimization II*, volume 13 of *Mathematical Programming Studies*, pages 8–16. Springer Berlin Heidelberg, 1980. ISBN 978-3-642-00803-0. doi: 10.1007/BFb0120902.

JA Rodríguez. On the Laplacian spectrum and walk-regular hypergraphs. *Linear and Multilinear Algebra*, 51(3):285–297, 2003.

K. Schmidt and G. Trenkler. The Moore-Penrose inverse of a semi-magic square is semi-magic. *International Journal of Mathematical Education in Science and Technology*, 32(4):624–629, 2001.

Koji Tsuda. Propagating distributions on a hypergraph by dual information regularization. In *Proceedings of the 22nd international Conference on Machine learning*, pages 920–927. ACM, 2005.

Vladimir Naumovich Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Ulrike Von Luxburg, Agnes Radl, and Matthias Hein. Getting lost in space: Large sample analysis of the commute distance. In *Proceedings of the 24th Conference on Neural Information Processing Systems (NIPS-10)*, volume 23, pages 2622–2630, 2010.

Shujun Zhang, Geoffrey D. Sullivan, and Keith D. Baker. The automatic construction of a view-independent relational model for 3-D object recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(6):531–544, 1993.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *Proceedings of the 22nd International conference on Machine learning (ICML-05)*, pages 1036–1043. ACM, 2005.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Proceedings of the 20th Conference on Neural Information Processing Systems (NIPS-06)*, pages 1601–1608, Cambridge, MA, 2006. MIT Press.

Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, volume 3, pages 912–919, 2003.