Spectral Learning with Hypernode Graphs Application to Skill Rating for Multiple Players Games*

Thomas Ricatte thomas@ricatte.fr
Rémi Gilleron remi.gilleron@univ-lille3.fr
Marc Tommasi marc.tommasi@univ-lille3.fr
Université de Lille et INRIA Lille Nord Europe
40, Av. Halley, 59650 Villeneuve d'Ascq - France

Editor:

Abstract

The aim of this paper is to propose methods for learning from interactions between groups in networks. For this, we introduce hypernode graphs as a formal tool able to model group interactions. A hypernode graph is a set of weighted binary relations between disjoint sets of nodes. We define Laplacians and kernels for hypernode graphs. And we propose spectral learning algorithms over hypernode graphs allowing to infer node ratings or node labelings. As a proof of concept, we model multiple players games with hypernode graphs and we define skill-rating learning algorithms. We apply these algorithms on multiple players games and obtain very promising results compared to Elo Duelling and Trueskill. We then study the expressive power of hypernode graphs and the associated spectral theory. That is, we prove that hypernode graph kernels strictly generalize over graph kernels. Also, we prove that hypernode graphs correspond to signed graphs such that the matrix $D - W$ is positive semidefinite. This shows that homophilic relations between groups may lead to non homophilic relations between individuals. Finally, we define the notion of connected hypernode graphs and we define a resistance distance generalizing over the resistance distance for graphs.

Keywords: Hypergraphs, Graph Laplacians, Graph Kernels, Spectral Learning, Semi-supervised Learning, Skill Rating Algorithms.

1. Introduction

Networks are commonly modeled by graphs where nodes correspond to individual objects and edges correspond to binary relationships between individuals. This is for instance the case for social networks with friendships between users, or computer networks with connections between machines. But in many situations, only interactions between groups are observed without any information on individuals besides group membership. As an example, in team sports and online games only outcomes of games between teams of multiple players are filed. Also, group interactions are observed when dealing with clusters in computer networks or communities in social networks. However in such contexts, the simultaneous evaluation of functions on groups and on individuals play an important role for applications. This is exactly the case for functions that measure CPU or cluster loads, individual players or team skills, users or communities ratings. The two evaluation levels of functions on items

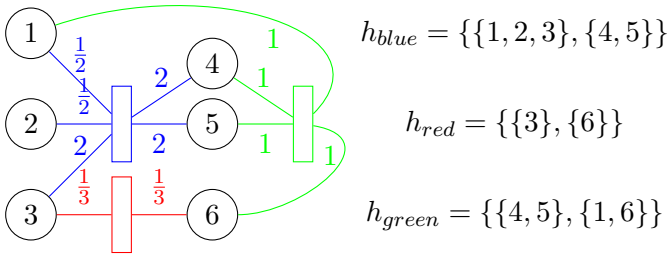


Figure 1: A hypernode graph with three hyperedges. A hyperedge is represented by a rectangle. Nodes in one of the two hypernodes are connected to the same side of the rectangle.

and functions on groups are however related by assuming an additive model. For instance, in multiple player games, the team skill is assumed to be the weighted sum of player skills as in Herbrich et al. (2006). A task that can be defined under these assumptions is to evaluate the function values of all groups and individuals when only a few values are observed. In the case of multiple player games, skills must be evaluated from a limited number of game outcomes, thus allowing to predict game outcomes for new games or to do matchmaking with well-balanced games.

In this paper, we propose a new model called hypernode graphs as a formal tool able to model group interactions. Hypernode graphs allow to consider node valuation problems as well as group valuation problems by assuming an additive model. We define a *hypernode graph* to be a set of hyperedges. A hyperedge is a pair of disjoint hypernodes, where a hypernode is a set of nodes. Every node of a hypernode is given a non negative real-valued weight and weights in a hyperedge satisfy an equilibrium condition. Roughly speaking, a hypernode models a group, a hyperedge models a relation between two groups, and individual weights correspond to the contribution of each individual to the relation between the two groups. An example of hypernode graph is shown in Figure 1. Hypernode graphs generalize over graphs because there is a one to one correspondence between undirected graphs and hypernode graphs in which all hypernodes are singleton sets.

Then, a learning task is to infer node function values over a hypernode graph. We considera semi-supervised setting where function values are given for only a few nodes or a few hypernodes. For semi-supervised learning on hypernode graphs, we assume that connected hypernodes tend to have similar values. For real-valued node functions over discrete structures, the gradient is a discrete derivative that allows to measure such a similarity. Therefore, we define a gradient for hypernode graphs. If we denote by G the gradient of a hypernode graph $\mathbf{h}$, we define the Laplacian Δ of $\mathbf{h}$ to be $\Delta = G^T G$, and we define the kernel $\Delta^\dagger$ to be the Moore-Penrose pseudoinverse of the Laplacian Δ. Then, the quantity $f^T \Delta f$ measures the smoothness of a real-valued node function f on the hypernode graph $\mathbf{h}$, i.e. $f^T \Delta f$ is small when connected hypernodes have close values. As the Laplacian is positive semidefinite, (graph) spectral learning algorithms (see Von Luxburg, 2007; Zhou et al., 2005; Zhu et al., 2003) can be applied for learning in hypernode graphs.

As a proof-of-concept, we consider the skill rating problem for multiple players games, that is how to infer individual skills of players from game outcomes in multiple players games. We consider competitive games between two teams where each team is composed of an arbitrary number of players. We model a set of games by a hypernode graph where hypernodes are teams of players. A player skill rating function is thus a function on nodes of the hypernode graph. Then, in order to model game outcomes, we add outcome nodes. An outcome node is added to every losing team in a hyperedge for a game. Such an outcome node will be given a positive value in order to model the relation between team skills given by the game outcome. Thus, the inductive bias used to learn skill ratings is to choose a smooth function on the hypernode graph such that outcome nodes values are fixed. Therefore, the skill rating problem is equivalent to a semi-supervised learning problem over a hypernode graph. The SSL problem can be solved by minimizing $\Omega(s) = s^T \Delta s$ where Δ is the Laplacian of the hypernode graph and where function values for game outcome nodes are given. The inferred skill rating function allows to compute team ratings and to predict game outcomes for new games. We consider real datasets of multiple players games (double tennis and online games) and apply our method in a batch setting. Experimental results show that we obtain very competitive results compared to specialized algorithms such as Elo duelling (see Elo, 1978) and TrueSkill (see Herbrich et al., 2006).

One question raised by the introduction of our new framework is whether or not problems on hypernode graphs can be solved using graphs. Before answering this question, let us consider the case of hypergraphs, a popular generalization of graphs where a hyperedge is a set of nodes. Hypergraphs have been studied from a machine learning perspective and have been used in many applications (see Klamt et al., 2009; Zhang et al., 1993; Cai and Strube, 2010). It has been proved in Agarwal et al. (2006) that the hypergraph Laplacians introduced so far do not provide a strict gain of expressiveness with respect to graph Laplacians since we can reduce any learning problem based on a hypergraph Laplacian to a learning problem based on a graph Laplacian using an adequate graph construction. This is not true for hypernode graphs. Indeed, we show that no graph construction allows to define a set of smooth functions over the graph which coincides with the set of smooth functions over a given hypernode graph. Thus, we state that hypernode graphs Laplacians strictly generalize over graph Laplacians.

It can be noted that the class of hypernode graph Laplacians strictly contains the class of graph Laplacians and also contains the class of graph kernels (a graph kernel is the Moore-Penrose pseudoinverse of a graph Laplacian). We can also observe that a convex linear combination of graph kernels (used in multiple graph kernel learning as in Argyriou et al. (2005)) is a hypernode graph kernel (i.e., the Moore-Penrose pseudoinverse of a hypernode graph Laplacian). We also show that, for every hypernode graph, we can construct a signed graph with adjacency matrix W such that $D - W$ is the Laplacian of the hypernode graph. The fact that W potentially contains negative entries should be related to the strict gain of expressiveness provided by our new framework. Let us recall that, for arbitrary signed graphs, the matrix $D - W$ can be indefinite. Thus, many works have studied the notion of Laplacian for signed graphs. Herbster (2008) has considered the class of symmetric diagonally dominant matrices with non negative diagonal entries. While others (see Hou, 2005; Goldberg et al., 2007; Kunegis et al., 2010) have used a modified definition of the Laplacian to obtain a positive semidefinite Laplacian. Last, Koren et al. (2002) have considered the class of signed

graphs such that $D - W$ is positive semidefinite for the problem of drawing graphs. A signed graph in this class can therefore be represented by a hypernode graph with the same Laplacian.

We finally study the definition of a distance between nodes in hypernode graphs. Indeed, it has been shown that, for connected graphs, a resistance distance can be defined from the graph kernel and that it can be expressed in term of the commute time distance in the graph (see Klein and Randić, 1993; Chandra et al., 1996). We define a pseudo-distance between nodes of a hypernode graph using the hypernode graph kernel. We show that it is a distance for a well chosen notion of connected hypernode graphs. But, it should be noted that we leave open the question of finding an algorithmic definition of connected components in hypernode graphs. Also, we introduce a diffusion operator for hypernode graphs and we show that the distance can be expressed from the differences of potentials. As a special case, we get the result on the commute time distance for graphs. But, when the hypernode graph is not a graph, the question of finding an interpretation of the distance between two nodes with random walks is left open because negative terms are involved.

2. Hypernode Graphs

The following definition is our contribution to the modeling of binary relationships between sets of entities.

Definition 1 *A* hypernode graph $\mathbf{h} = (V, H)$ *is a set of nodes* V *and 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* V*). 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*

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

In the following, we denote by n the number of nodes, we denote by p the number of hyperedges, and we assume an ordering of the set of nodes and of the set of hyperedges. We say that a node i *belongs to* a hyperedge h, that we denote by $i \in h$, if $w_h(i) \neq 0$. We define the degree of a node i by $\deg(i) = \sum_h w_h(i)$. The degree of a node is positive when it belongs to at least one hyperedge. We define the diagonal degree matrix by $D = \mathrm{diag}(\deg(1), \ldots, \deg(n))$ and the volume of the hypernode graph by $\mathrm{Vol}(\mathbf{h}) = \sum_{i \in N} \deg(i) = \mathrm{Tr}(D)$.

An example of hypernode graph is shown in Figure 1. The blue hyperedge h_{blue} links the sets $\{1, 2, 3\}$ and $\{4, 5\}$. The weights of h_{blue} satisfy the Equilibrium condition: $\sqrt{1/2} + \sqrt{1/2} + \sqrt{2} = \sqrt{2} + \sqrt{2}$. The green hyperedge h_{green} links the sets $\{4, 5\}$ and $\{1, 6\}$ and the Equilibrium condition is obviously satisfied. The red hyperedge h_{red} links the two singleton sets $\{3\}$ and $\{6\}$ and the weights of h_{red} are equal, thus the hyperedge h_{red} can be viewed as an edge with edge weight $1/3$.

As noted in the previous example, 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 equal. Therefore, every hypernode graph such that all hyperedges are unordered

pairs of singleton nodes can be viewed as a graph with adjacency matrix W defined by $W_{i,j} = W_{j,i} = w_h(i) = w_h(j)$ for every hyperedge $\{\{i\},\{j\}\}$, and 0 otherwise. Conversely, a graph can be viewed as hypernode graph where every hypernode is a singleton set.

We are mainly interested in evaluating real-valued functions on nodes and on hypernodes. We assume a weighted linear model such that any real-valued node function f can be extended to a hypernode u of a hyperedge h by $f(u) = \sum_{i \in u} f(i)\sqrt{w_h(i)}$. Moreover, we will consider an homophilic assumption stating that connected hypernodes tend to have similar values. In order to implement this assumption, we introduce a gradient for hypernode graphs by

Definition 2 *Let $\mathbf{h} = (V, H)$ be a hypernode graph and f be a real-valued node function, the* (hypernode graph) unnormalized gradient *of $\mathbf{h}$ is a linear application, denoted by* grad*, that maps every real-valued node function f into a real-valued hyperedge function* $\mathrm{grad}(f)$ *defined, for every $h = \{s_h, t_h\}$ in H, by*

$$\mathrm{grad}(f)(h) = f(t_h) - f(s_h) = \sum_{i \in t_h} f(i)\sqrt{w_h(i)} - \sum_{i \in s_h} f(i)\sqrt{w_h(i)} \ , \tag{1}$$

where an arbitrary orientation of the hyperedges has been chosen. We denote by $G \in \mathbb{R}^{p \times n}$ the matrix of grad*.*

Because of the Equilibrium Condition, the gradient of every constant node function is the zero-valued hyperedge function. The square $n \times n$ real valued matrix $\Delta = G^T G$ is defined to be the *unnormalized Laplacian* of the hypernode graph $\mathbf{h}$. When the hypernode graph is a graph (all hypernodes are singleton sets), then the hypernode graph Laplacian is equal to the unnormalized graph Laplacian. The Laplacian Δ does not depend on the arbitrary orientation of the hyperedges used for defining the gradient. By definition, the Laplacian Δ is positive semidefinite. We define the *smoothness* of a real-valued node function f over a hypernode graph $\mathbf{h}$ to be $\Omega(f) = f^T \Delta f$. Last, we define the *hypernode graph kernel* of a hypernode graph $\mathbf{h}$ to be the Moore-Penrose pseudoinverse $\Delta^\dagger$ of the hypernode graph Laplacian Δ.

Because a hypernode graph Laplacian is positive semidefinite, we can leverage the spectral learning algorithms defined in Von Luxburg (2007), Zhou et al. (2005), and Zhu et al. (2003) from graphs to hypernode graphs. In the next section we show how to use spectral learning algorithms over hypernode graphs for the skill rating problem in multiple players games.

3. Skill Rating for Multiple player Games

We consider competitive games between two teams where each team is composed of an arbitrary number of players. A first objective is to compute the skill ratings of individual players from a batch of games with their outcomes. A second objective is to predict a game outcome for a new game. In this section, we show how to model a batch of games by a hypernode graph and we apply learning algorithms over the hypernode graph to infer player skills that allow to predict game outcomes of new games.

3.1 Multiplayer Games

Let us consider a set of players $P = \{1, \dots, n\}$ and a set of games Γ together with their respective game outcome. Each game is between two teams of an arbitrary number of players. Let us also consider that a player i contributes to a game γ with a positive real value $c(i)$. Let us assume that each player has a skill $s(i)$, we suppose an additive model as in Herbrich et al. (2006) stating that the skill of a team is the weighted sum of the skills of the players in the team. More formally, given two teams of players $A = \{a_1, a_2, \dots, a_\ell\}$ and $B = \{b_1, b_2, \dots, b_k\}$ playing game γ, then A is predicted to be the winner if

$$\sum_{i=1}^{\ell} c(a_i)s(a_i) > \sum_{i=1}^{k} c(b_i)s(b_i) \ . \tag{2}$$

Equivalently, one can rewrite this inequality by introducing a positive real number o on the right hand side such that

$$\sum_{i=1}^{\ell} c(a_i)s(a_i) = o + \sum_{i=1}^{k} c(b_i)s(b_i) \ , \tag{3}$$

where the real number o quantifies the game outcome. In the case of a draw, there is an equality between the team skills, thus the game outcome o is set to 0. Given a set of games with their respective outcome, the goal is to infer a skill rating function $s \in \mathbb{R}^n$ that respects equations 3 as much as possible. We define the cost of a game γ with outcome o for a skill function s by

$$C_\gamma(s) = \| \sum_{i=1}^{\ell} c(a_i)s(a_i) - (o + \sum_{i=1}^{k} c(b_i)s(b_i)) \|^2 \ .$$

Consequently, given a set of games Γ and the corresponding game outcomes, the goal is to find a skill rating function s^* that minimizes the sum of the different costs, i.e. search for

$$s^* = \arg\min_s \sum_{\gamma \in \Gamma} C_\gamma(s) \ . \tag{4}$$

3.2 Modeling Games with Hypernode Graphs

We introduce the general construction by considering an example. Let us consider a game γ between two teams $A = \{1, 2, 3\}$ and $B = \{4, 5\}$. Let us also assume that all the players contribute to the game with weight 1. Such a game can be modeled by a hyperedge between sets of nodes $\{1, 2, 3\}$ and $\{4, 5\}$ with weights equal to 1. Now, let us suppose that B wins the game, then the skill rating function s must satisfy Equation (3), that is $s(1) + s(2) + s(3) + o = s(4) + s(5)$ where o is positive. In order to model the game outcome in the hyperedge, we introduce a virtual player that plays along with team A with a contribution equal to 1. The virtual player is represented by a new node O, called outcome node, with weight 1. For the outcome node, a skill rating function must satisfy $s(O) = o$. Last, for the hyperedge to satisfy the equilibrium condition, we add a node Z, called lazy node, to the hypernode corresponding to B. For the lazy node, a skill rating function must satisfy $s(Z) = 0$. The construction is illustrated in Figure 2. If the game outcome is $o = 2$,

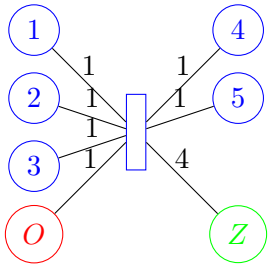


Figure 2: Hyperedge h for a game $A = \{1, 2, 3\}$ against $B = \{4, 5\}$ where B wins. The outcome node O is added to A (the loser). The lazy node is added to B (the winner). All weights are set to 1 except the weight for Z chosen in order to ensure the Equilibrium condition.

then the skills must satisfy $s(1) + s(2) + s(3) + 2$ is equal to $s(4) + s(5)$. And, if we constraint a skill function on the hyperedge to satisfy $s(O) = 2$ and $s(Z) = 0$, then the homophilic assumption on the hyperedge states that $s(1) + s(2) + s(3) + 2$ must be close from $s(4) + s(5)$ which expresses the expected relations between player skills given the game outcome.

In the general case, let us consider a set of players $P = \{1, \ldots, n\}$ and a set of games Γ. Each game γ is between two teams A and B of multiple players, the winning team is known as well as the game outcome o. Let us also consider that a player i contributes to a game γ with a non negative weight $c(i)$. We can define, for every game γ a hyperedge h as follows

1. The players of A define one of the two hypernodes of h, the weight of a player node i is defined to be $c(i)^2$;
2. do the same construction for the second team B;
3. add a *outcome node* O to the set of player nodes corresponding to the losing team, its weight is set to 1;
4. add a *lazy node* Z 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 define the hypernode graph $\mathbf{h} = (V, H)$ as the set of all hyperedges h for all games γ in Γ. Now, a skill rating function is the restriction to player nodes of a real-valued node function over the hypernode graph. In order to model the game outcomes, we fix the skill rating function values over the additional nodes. A skill function s over $\mathbf{h}$ modeling Γ must satisfy: for every lazy node Z, the function value $s(Z)$ must be 0, and, for every outcome node O, the function value $s(Z)$ must be the outcome value o. Let Δ be the unnormalized Laplacian of $\mathbf{h}$, and let s be a real-valued node function on h. Then, it is easy to show that the skill rating problem (4) is equivalent to find a real valued node function s over $\mathbf{h}$ solution of

$$\begin{array}{ll} \underset{s}{\text{minimize}} & s^T \Delta s \\ \text{subject to} & \text{for every outcome node } O, \;\; s(O) = o; \text{ for every lazy node } Z \,, \;\; s(Z) = 0 \end{array} \tag{5}$$

3.3 Regularizing the hypernode graph

When the number of games is small, the number of hyperedges is small and some node players belong to only one hyperedge. Thus, player skills can be defined independently while satisfying the constraints and it will be irrelevant to compare them. In order to solve this issue, we introduce in Equation (5) a regularization term based on the standard deviation $\sigma(s_p)$ of players skills where $s_p = (s(1), \ldots, s(n))$. This allows us to reduce the spread of the player skills and leads us to find a real valued node function s over $\mathbf{h}$ solution of

$$\begin{array}{ll} \underset{s}{\text{minimize}} & s^T \Delta s + \mu \sigma(s_p)^2 \\ \text{subject to} & \text{for every outcome node } O,\ \ s(O) = o; \text{ for every lazy node } Z\ ,\ s(Z) = 0 \end{array} \tag{6}$$

where μ is a regularization parameter. That is, we control the spread of s_p avoiding extreme values for players involved in a small number of games.

We show in Appendix A that the regularized optimization problem 6 can be rewritten as find a real valued node function s over $\mathbf{h}_\mu$ solution of

$$\begin{array}{ll} \underset{s}{\text{minimize}} & s^T \Delta_\mu s \\ \text{subject to} & \text{for every outcome node } O,\ \ s(O) = o; \text{ for every lazy node } Z\ ,\ s(Z) = 0 \end{array} \tag{7}$$

where Δ_μ is the Laplacian of the hypernode graph $\mathbf{h}_\mu$ obtained from the hypernode graph $\mathbf{h}$ by: add a regularizer node R; for every player node, add an hyperedge between the player node and the regularizer node R and set the (node) weights to μ/n. The added hyperedges can be viewed as edges with weight μ/n.

Note that, assuming unitary contributions and a Gaussian distribution for skill ratings, we can prove that the value of μ/n should have the same order of magnitude than the average number of games played by a player. Indeed, this will allow us to obtain similar expected values for the two terms $s^T \Delta s$ and $\mu \sigma(s_p)^2$. We follow this guideline in the experiments presented below.

3.4 Inferring Skill Ratings and Predicting Game Outcomes

The problem (7) can be viewed as a semi-supervised learning problem on the hypernode graph $\mathbf{h}_\mu$ because the question is to predict player node values (skill ratings) from known values for lazy nodes and outcome nodes. The Laplacian Δ_μ is positive semidefinite because it is a hypernode graph Laplacian. Therefore, we can use the semi-supervised learning algorithm presented in Zhu et al. (2003). This algorithm was originally designed for graphs and solves exactly the problem (7) by putting hard constraints on the outcome nodes and on the lazy nodes. It should be noted that the algorithm is parameter free. We denote this learning method by H-ZGL.

Another approach to solve the semi-supervised learning problem is to use 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), we train a regression support vector machine using outcome nodes values and lazy nodes values, and we predict node player values. The main parameter is the soft margin parameter. We denote this method by H-SVR.

Using one of these two methods, we can infer players skills, then deduce teams skills, and finally 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 test set of games Γ_u for which game outcomes are hidden. The goal is to predict game outcomes for the test set Γ_u. We use the following

Algorithm 1 Predicting game outcomes

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

1: Build, as described in Sections 3.2 and 3.3, the regularized hypernode graph $\mathbf{h}_\mu$ from Γ_l
2: Compute an optimal skill rating s^* using H-ZGL or H-SVR.
3: Compute the mean skill $\bar{s}$ among players in Γ_l
4: **for** each game in Γ_u **do**
5: Assign skills given by s^* to players involved in Γ_l, and $\bar{s}$ otherwise
6: Evaluate the inequality (2) and predict the winner
7: **end for**

3.5 Experiments

In this section, we report experimental results for the inference of player skills and the prediction of game outcomes for different datasets in the batch setting described above. Note that other works have considered the online setting as in Herbrich et al. (2006) or Elo (1978).

Tennis Doubles

We consider a dataset of tennis doubles collected between January 2009 and September 2011 from ATP tournaments (World Tour, Challengers and Futures). Tennis doubles are played by two teams of 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 team during the game. The dataset Γ consists in 10028 games with 1834 players.

Along the experimental process, given a proportion ρ, we draw randomly a training set Γ_l of size $\rho\%$ of the number of games in Γ. The remaining games define the test set Γ_u. We present in Figure 3 several statistics related to this process on the Tennis dataset. First, it can be noticed that many players have played a small number of games. Second, when the number of games in the training set is small, half of the players are involved in games in the test set. Therefore, the skill rating problem and the game outcome prediction problem become far more difficult to solve when few games are used for learning.

The experimental setting is defined as follows. Let ρ be a proportion varying from 10% to 90% by 10%. For every value of ρ, we repeat ten times: draw a random training set Γ_l of size $\rho\%$ of the number of games in Γ; let the remaining games define the test set Γ_u; infer player skills and predict game outcomes following Algorithm 1. For the definition of the hypergraph, we set all node player weights to 1 because we do not have information on player contributions. The node outcome values o are set to be the difference between the number of sets won by the two teams, thus the possible outcome node values are 1, 2 or 3. Therefore, there are 3 outcome nodes and one lazy node shared by all hyperedges. The resulting hypernode graph has at most 1839 nodes: at most 1834 player nodes, 1 lazy node, 3 outcome nodes, and 1 regularizer node. In order to complete the definition of the

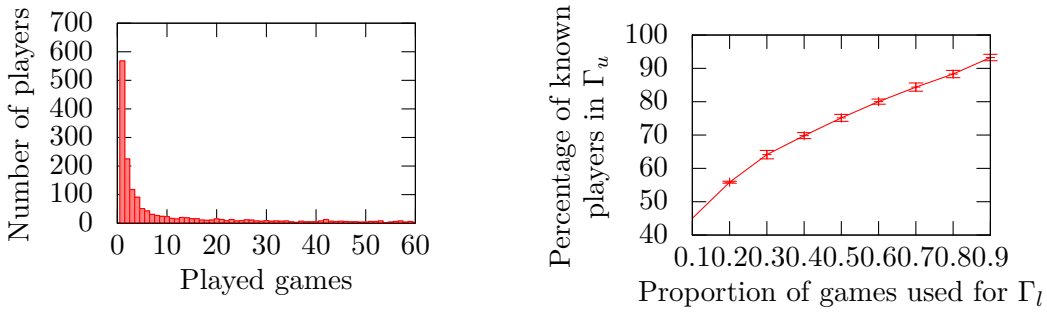


Figure 3: [*left*] Distribution of the number of players against the number of played games; [*right*] Average percentage of players in Γ_u which are involved in some game in Γ_l

hypernode graph $\mathbf{h}_\mu$, it remains to fix the regularization node weight μ/n. As announced above, we choose the value of μ so that μ/n is close to the average number of games played by a player in the training set. The experiments presented in this section use $\mu/n = 16$.

We report in Figure 4 prediction results using: algorithm 1 and H-ZGL, algorithm 1 and H-SVR, Elo Duelling, and Trueskill [1]. It can be noted that Elo Duelling performs poorly but Elo was designed for one against one games. It should be noted that we have done similar experiments for tennis singles. The results are not reported here but they show that Elo Duelling and True skill obtain similar results but are outperformed by H-ZGL and H-SVR. For the dataset of tennis doubles, it must be noted that H-ZGL and H-SVR outperform Trueskill for a small number of training games. Also, H-ZGL outperforms trueskill whatever is the number of training games. On this set of experiments H-ZGL outperforms H-SVR but it should be noted that H-ZGL is parameter free and that we use H-SVR with the default soft margin parameter value.

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 a random guess for the prediction of draw.

We consider the same experimental process. For the construction of the hypernode graph in Algorithm 1, we fix all players contributions in games to 1 and we again set the parameter value μ/n to 16. In the optimization problem 7, the game outcomes o are set to 1 when

1. 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).

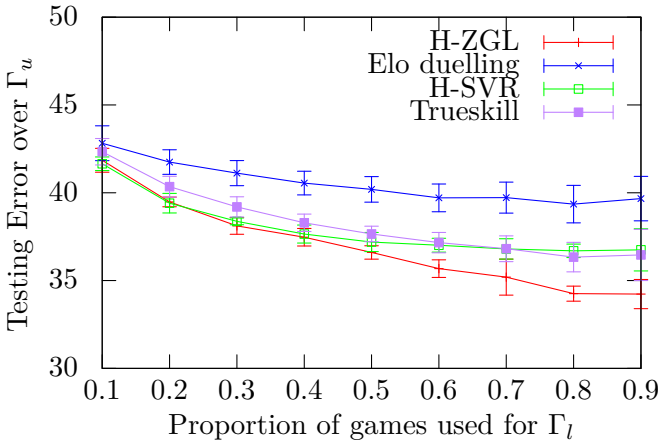


Figure 4: Predictive error for Double Tennis dataset

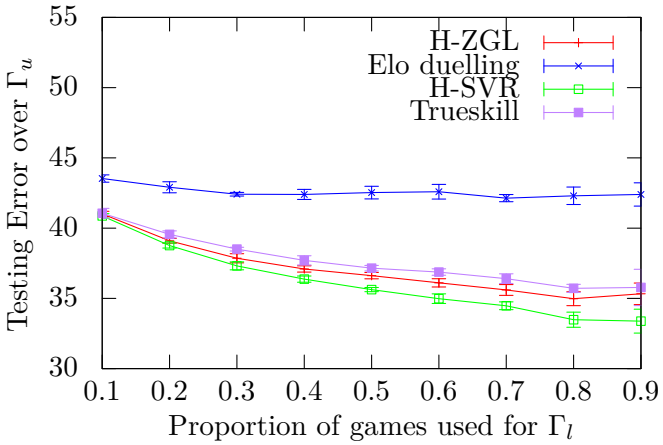


Figure 5: Predictive error for Halo2 dataset

the game has a winner and 0 otherwise because game scores vary depending on the type of game. Therefore in the hypernode graph there are one lazy node and two outcome nodes (for 0 and 1) shared by all hyperedges. We again compare the skill rating algorithms H-ZGL, H-SVR, Elo Duelling and Trueskill. The number of prediction errors over game outcomes is computed assuming that a draw can be regarded as half a win, half a loss as in Lasek et al. (2013). We present the experimental results in Figure 5. 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 (with default parameter value) outperforms H-ZGL.

4. Hypernode Graphs, Graphs and Signed Graphs

In this section, we study the class of hypernode graph Laplacians. In particular, we show that any hypernode graph Laplacian can be seen as the Laplacian matrix of a signed graph (i.e., a graph that potentially contains negative edge weights).

4.1 The Class of Hypernode Graphs Laplacians

Proposition 3 *The class of hypernode graph Laplacians is the class of symmetric positive semidefinite real-valued matrices M such that $\mathbf{1} \in \text{Null}(M)$, where $\text{Null}(M)$ is the null space of M.*

Proof It is easy to verify from the Laplacian definition that a Laplacian M is a symmetric positive semidefinite real-valued matrix such that $\mathbf{1} \in \text{Null}(M)$. Conversely, let us consider the following algorithm

Algorithm 2 Construction of a hypernode graph

Input: $M \in \mathbb{R}^{n \times n}$ symmetric positive semidefinite such that $\mathbf{1} \in \text{Null}(M)$
Set V to $\{1, \ldots, n\}$; set H to $\emptyset$
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 $s_h = t_h = \emptyset$
  for each node $i \in V$ **do**
    Define $w_h(i) = \mathbf{r}(i)^2$
    if $\mathbf{r}(i) < 0$ **then** Add i to s_h **else if** $\mathbf{r}(i) > 0$ Add i to t_h **end if**
  end for
  Add h to H
end for
return The hypernode graph $\mathbf{h} = (V, H)$

From a square root decomposition $G^T G$ of M, for each line of G, we define a hyperedge $h = \{s_h, t_h\}$ where nodes in s_h (respectively in t_h) have positive (respectively negative) values in the line, and node weights are chosen to be squares of values in the line. Then, the Equilibrium condition 1 is satisfied because $\mathbf{1} \in \text{Null}(M)$. And it is easy to check that G is a gradient matrix of $\mathbf{h}$ and, consequently, that $M = G^T G$ is the Laplacian of $\mathbf{h}$. ■

As a consequence of Proposition 3, the class of hypernode graph Laplacians

- is closed under convex linear combination,
- is closed under pseudoinverse,
- and strictly contains the class of graph Laplacians and the class of graph kernels (a graph kernel is the Moore-Penrose pseudoinverse of a graph Laplacian).

Linear combinations of graph kernels have been used for semi-supervised learning (see for instance Argyriou et al., 2005). Another consequence of Proposition 3 is that a convex

linear combination of graph kernels is a hypernode graph kernel (also a hypernode graph Laplacian). To conclude this section, we propose

Conjecture: The class of hypernode graph Laplacians is the smallest class closed by convex linear combinations which contains all graph kernels.

The difficult part is to prove that every hypernode graph Laplacian is a convex linear combination of graph kernels. This is because a graph kernel, which is defined to be the pseudoinverse of a graph Laplacian, has no simple analytic form.

4.2 Hypernode Graph Laplacians Strictly Generalize Graph Laplacians

As said above, the class of hypernode graph Laplacians strictly contains the class of graph Laplacians. But this does not allow us to claim that hypernode graph Laplacians provide a gain of expressiveness over graph Laplacians. Indeed, it could be the case that through a well chosen graph construction, the set of smooth functions defined on hypernode graphs could be made to coincide exactly with the set of smooth functions defined on graphs. The class of hypergraph Laplacians has been studied in that perspective of expressiveness. Indeed, it has been shown in Agarwal et al. (2006) that all hypergraph Laplacians defined so far – among them the Laplacians Δ_{B} from Bolla (1993), Δ_{R} from Rodríguez (2003) and Δ_{ZHS} from Zhou et al. (2007) – can be defined as (restrictions of) graph Laplacians using a graph construction such as the clique expansion (where each hyperedge is replaced by a clique graph with uniform weights) or the star expansion (where, for every hyperedge, a new node is added and is linked with all the nodes in the hyperedge).

While one can think of similar constructions for the case of hypernode graphs, we prove that there does not exist a (finite) graph expansion of a hypernode graph which defines the same set of smooth functions over the original set of nodes. The proof is based on the very simple hypernode graph $\mathbf{h}$ shown in Figure 6. Note that a function is smooth over $\mathbf{h}$ if and only if it satisfies

$$(f(1) + f(2) - f(3) - f(4))^2 = 0 \qquad (\mathcal{C})$$

We show by contradiction in Appendix B that there does not exist a finite graph $\mathbf{g}$ whose node set contains $\{1, 2, 3, 4\}$ and that satisfies the following conditions:

1. Every smooth function over $\mathbf{g}$ satisfies $(\mathcal{C})$
2. Every function satisfying $(\mathcal{C})$ can be extended into a smooth function over $\mathbf{g}$.

The first condition ensures that smooth functions on $\mathbf{g}$ are related to smooth functions over the original hypernode graph $\mathbf{h}$. The second condition ensures that smooth functions on the original hypernode graph can be defined from smooth functions over extended finite graphs. The combination of these two conditions ensure that the problem of finding smooth functions on $\mathbf{h}$ can be rephrased as a graph problem (finding the smooth functions on $\mathbf{g}$ and compute their restrictions to $\{1, 2, 3, 4\}$). The fact that no graph $\mathbf{g}$ can satisfy both conditions at the same time for a simple hypernode graph $\mathbf{h}$ shows that the smoothness conditions defined by the hypernode graphs can not be defined with undirected extended graphs.

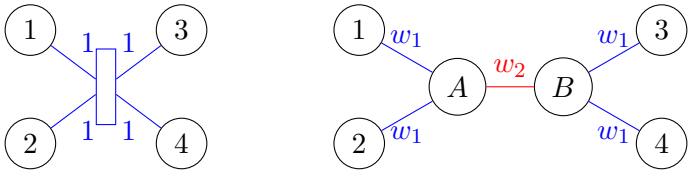


Figure 6: Hypernode graph $\mathbf{h}$ and a candidate star expansion.

$$G = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \qquad G = \begin{pmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & \sqrt{2} & 0 \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & \sqrt{2} \end{pmatrix}$$

Figure 7: two L-equivalent hypernode graphs with their respective gradient which are square root of the Laplacian Δ shown in Figure 8.

4.3 L-equivalent Hypernode Graphs and Signed Graphs

Let us consider a symmetric positive semidefinite real-valued matrix M such that $\mathbf{1} \in \text{Null}(M)$, M is the Laplacian of a hypernode graph by Proposition 3. But, as the square root decomposition is not unique, there are several hypernode graphs with the same Laplacian that we called *L-equivalent*. Examples of L-equivalent hypernode graphs are given in Figure 7. In order to study the L-equivalent relation and to obtain results showing whether a hypernode graph is L-equivalent to a graph, we first show that hypernode graphs are related to signed graphs.

It is known that the Laplacian matrix Δ of a graph can be written $D - W$ where W is the adjacency matrix of the graph and D is the corresponding degree matrix. Let us consider a hypernode graph $\mathbf{h}$ with Laplacian Δ, we define the *pairwise weight matrix* W of $\mathbf{h}$ by $W_{i,j} = -\Delta_{i,j}$ if $i \neq j$, and 0 otherwise. Note that the pairwise weight matrix coincides with the classic adjacency matrix when the hypernode graph is a graph. Let us define the degree of a node i to be $\deg(i) = \sum_{j \in V} W_{i,j}$ and let us denote by D the diagonal matrix of all $\deg(i)$. Then, because of the property $\mathbf{1} \in \text{Null}(M)$ in Proposition 3, it is immediate that, for every i, $\Delta_{i,i} = \sum_{j \in V} W_{i,j}$. As a consequence, we have

Proposition 4 *Let $\mathbf{h} = (V, H)$ be a hypernode graph, let W be the pairwise weight matrix of $\mathbf{h}$, and let D be the diagonal degree matrix of $\mathbf{h}$. Then, the Laplacian of $\mathbf{h}$ is $\Delta = D - W$.*

An example is shown in Figure 8. As a consequence of the above proposition, we can leverage the pairwise weight matrix to characterize L-equivalent hypernode graphs by

Proposition 5 *Two hypernode graphs are L-equivalent if and only if they have the same pairwise weight matrix. Two L-equivalent hypernode graphs have the same degree matrix.*

$$\Delta = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix} ; W = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} ; D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}.$$

Figure 8: Laplacian matrix, pairwise weight matrix, and degree matrix for the two L-equivalent hypernode graphs shown in Figure 7.

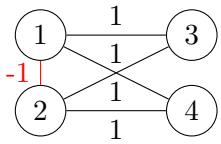


Figure 9: The reduced signed graph with adjacency matrix W of Figure 8 of the two L-equivalent hypernode graphs shown in Figure 7.

The pairwise weight matrix W contains in general negative weights and can be thus interpreted as the adjacency matrix of a signed graph. Therefore, we define the *reduced signed graph* of a hypernode graph $\mathbf{h}$ to be the signed graph $\tilde{\mathbf{g}}$ with adjacency matrix W. An example is shown in Figure 9. Then, we can show that

- the reduced signed graph of a graph $\mathbf{g}$ (viewed as a hypernode graph) is the graph $\mathbf{g}$,
- a hypernode graph $\mathbf{h}$ is L-equivalent to a graph if and only if its reduced signed graph is a graph,
- a signed graph with adjacency matrix W is the reduced signed graph of a hypernode graph if and only if the matrix $D - W$ is positive semidefinite.

The definition of the pairwise weight matrix may seem ad hoc to mimic the definition of the Laplacian in the graph case. But, we can show that the pairwise weight matrix can be defined directly from the hypernode graph using the following formula

$$\forall i \neq j, \quad W_{i,j} = \sum_{h \in H} P(h,i,j)\sqrt{w_h(i)}\sqrt{w_h(j)} \ , \tag{8}$$

where $P(h,i,j) = 1$ if i and j belongs to two different ends of h, $P(h,i,j) = -1$ if i and j belongs to the same end of h, and 0 otherwise. And the formula can be interpreted as follows. The pairwise weight $W_{i,j}$ is computed as a sum over all the hyperedges, then it can be understood as an aggregation of all the information over hyperedges involving the nodes i and j. For every term in the sum, the quantity $\sqrt{w_h(i)}$ can be viewed as the cost of entering the hyperedge h at node i, the quantity $\sqrt{w_h(j)}$ as the cost of exiting the hyperedge h at node j, and $P(h,i,j)$ as a sign depending whether i and j are in the same end or in different ends of the hyperedge.

5. Resistance Distance and Random Walks in Hypernode Graphs

In this section, we study whether a distance can be defined between nodes of a hypernode graph and how such a distance can be interpreted in the hypernode graph. Throughout the section, we consider a hypernode graph $\mathbf{h} = (V, H)$ with Laplacian Δ.

5.1 Defining a distance in Hypernode Graphs

Let $\Delta^\dagger$ be the Moore-Penrose pseudo-inverse of Δ, let us define d by, for every i, j in V,

$$d(i,j) = \left(\sqrt{\Delta^\dagger_{i,i} + \Delta^\dagger_{j,j} - 2\Delta^\dagger_{i,j}}\right) \quad . \tag{9}$$

Because $\Delta^\dagger$ is symmetric positive semidefinite, we have

Proposition 6 *d defines a pseudometric on $\mathbf{h}$, i.e., it is positive, symmetric and satisfies the triangle inequality (for all i, j, k the inequality $d(i,j) \leq d(i,k) + d(k,j)$ holds).*

For the pseudometric d to be a distance, d should satisfy also the coincidence axiom: $d(i,j) = 0 \Rightarrow i = j$. This is not true in general as $d(1,2) = 0$ for the hypernode graph in Figure 6. The intuition is that the nodes 1 and 2 can not be distinguished because the smoothness condition is on the sum $f(1) + f(2)$. Nevertheless we can show that

Proposition 7 *When $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$, d defines a metric (or distance) on $\mathbf{h}$.*

Let us recall that, in the graph case, Proposition 6 holds and that Proposition 7 holds when the graph is connected. By analogy with the case of graphs, the condition $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$ can be viewed as an algebraic definition of a connected hypergraph. So far, we have not found an alternative algorithmic definition of connected components in hypernode graphs.

Finally, let us note that, in the graph case, for connected graphs, both d and d^2 are metrics while, for hypernode graphs, d^2 does not satisfy the triangle inequality even if $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$.

5.2 Resistance Distance and Diffusion in Hypernode Graphs

Let us suppose that $\mathbf{h}$ satisfies $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$. Our goal is to study whether d^2 can be written in terms of a diffusion function in the hypernode graph. First, let us consider the Poisson equation $\Delta f = \mathbf{In}$ that models the diffusion of an input charge $\mathbf{In}$ through a system associated with the Laplacian operator Δ. Let us consider a node j called *sink* node, we consider the input function $\mathbf{In}_j$ defined by $\mathbf{In}_j(j) = \deg(j) - \mathrm{Vol}(\mathbf{h})$ and $\mathbf{In}_j(i) = \deg(i)$ if $i \neq j$. We can prove that

Lemma 8 *The solutions of $\Delta f = \mathbf{In}_j$ are the functions $f = \mu\mathbf{1} + \Delta^\dagger\mathbf{In}$ where $\mu \in \mathbb{R}$.*

The compete proof is presented in Appendix C. It is based on properties of pseudoinverse matrices and on the hypothesis $\mathrm{Null}(\Delta) = \mathrm{Span}(\mathbf{1})$. Then, for every pair of nodes (k, ℓ) in V, we define $\mathrm{V}_j(k,\ell)$ as the difference of potential between k and ℓ, i.e., we define $\mathrm{V}_j(k,\ell) = f(k) - f(\ell)$ where f is a solution of the Poisson equation $\Delta f = \mathbf{In}_j$. Lemma 8

allows to show that the definition of $\mathrm{V}_j(k,\ell)$ does not depend on the choice of the solution of the Poisson equation and that an equivalent definition is

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

where $\mathbf{e}_i$ is the unit vector with 1 in component i. We can now relate the distance d^2 and the diffusion potential V_j by

Proposition 9 *For every i, j in V, we have $\mathrm{Vol}(\mathbf{h})d^2(i,j) = \mathrm{V}_j(i,j) + \mathrm{V}_i(j,i)$, where $\mathrm{V}_j(i,i) = 0$, and for $i \neq j$,*

$$\mathrm{V}_j(i,j) = \sum_{h|i\in h} \frac{w_h(i)}{\deg(i)} \left[1 + \sum_{k\in h, k\neq i} P(h,i,k) \sqrt{\frac{w_h(k)}{w_h(i)}} \mathrm{V}_j(k,j) \right] \ . \tag{10}$$

The proof is given in Appendix D.

5.3 Resistance Distance and Random Walks in Hypernode Graphs

Proposition 9 provides an expression of d^2 in terms of diffusion potentials. We first show that the proposition generalizes the result of Klein and Randić (1993); Chandra et al. (1996) relating the resistance distance and random walks in graphs.

Indeed, let us consider a hypernode graph $\mathbf{h}$ where all hypernodes are singleton sets. The hypernode graph can be viewed as a graph and every hyperedge h that contains a node i is an edge $\{\{i\},\{k\}\}$ with $k \in N$. Thus, we have $w_h(i) = w_h(k) = W_{i,k}$ and Equation (10) can be rewritten as

$$\mathrm{V}_j(i,j) = \sum_{k\in N} \frac{W_{i,k}}{\deg(i)} (1 + \mathrm{V}_j(k,j)) \ .$$

Thus, $\mathrm{V}_j(i,j)$ 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). Consequently, Proposition 9 states that, when $\mathbf{h}$ is a graph, the distance $d^2(i,j)$ between two nodes i and j is equal to the commute-time distance between i and j divided by the overall volume, which is the expression found in Klein and Randić (1993); Chandra et al. (1996).

Now, let us consider the general case where the hypernode graph is not a graph. Let us define $p(h|i) = \frac{w_h(i)}{\deg(i)}$ and $p(k|h,i) = P(h,i,k)\sqrt{\frac{w_h(k)}{w_h(i)}}$. Then, we can rewrite Equation (10) as

$$\mathrm{V}_j(i,j) = \sum_{h|i\in h} p(h|i) \left[1 + \sum_{k\in h, k\neq i} p(k|h,i) \mathrm{V}_j(k,j) \right] \ .$$

The term $p(h|i)$ can be interpreted as a jumping probability from node i to the hyperedge h because $p(h|i)$ is non-negative and $\sum_h p(h|i) = 1$. Let us now consider the term $p(k|h,i)$, it can be shown that $\sum_k p(k|h,i) = 1$. But the term $p(k|h,i)$ is negative when i and k belong to the same end of h. Consequently, the quantity $p(k|h,i)$ can not be interpreted as a jumping probability from node i to node k with the hyperedge h. Therefore, there is no easy interpretation of d^2 in terms of random walks in the hypernode graph because of possible negative values in the expression of $d^2(i,j)$.

6. Conclusion

We have introduced hypernode graphs for learning from binary relations between groups in networks. We have defined a spectral theory allowing to model homophilic relations between groups. Properties of Laplacians and kernels have allowed us to define learning algorithms and an application to skill rating for multiple players games has been presented. This paper opens many research problems. From a machine learning perspective, we hope that the model will open the way to solving new learning problems in networks. From a fundamental perspective, many questions must be investigated. For instance, we have seen that connectivity properties and random walks in hypernode graphs need more research. Also, the definition of cuts in hypernode graphs and the generalization of the Max-Flow Min-Cut theorem are open questions.

References

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, Mark Herbster, and Massimiliano Pontil. Combining Graph Laplacians for Semi-Supervised Learning. In *Proceedings of the 19th Annual Conference on Neural Information Processing Systems (NIPS-05)*, pages 67–74, 2005.

Marianna Bolla. Spectra, euclidean representations and clusterings of hypergraphs. *Discrete Mathematics*, 117(1):19–39, 1993.

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.

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.

Arpad Emrick Elo. *The Rating of Chess Players, Past and Present.* Arco Publishing, 1978.

Andrew B Goldberg, Xiaojin Zhu, and Stephen J Wright. Dissimilarity in graph-based semi-supervised classification. In *International Conference on Artificial Intelligence and Statistics*, pages 155–162, 2007.

Scott Hamilton. PythonSkills: Implementation of the TrueSkill, Glicko and Elo Ranking Algorithms. `https://pypi.python.org/pypi/skills`, 2012.

Ralf Herbrich, Tom Minka, and Thore Graepel. TrueSkill$^{\text{TM}}$: A Bayesian Skill Rating System. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS-06)*, pages 569–576, 2006.

Mark Herbster. Exploiting cluster-structure to predict the labeling of a graph. In *Proceedings of the 19th International Conference on Algorithmic Learning Theory (ALT-08)*, pages 54–69, 2008.

Yao Ping Hou. Bounds for the least Laplacian eigenvalue of a signed graph. *Acta Mathematica Sinica*, 21(4):955–960, 2005.

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.

Yeruda Koren, Liran Carmel, and David Harel. ACE: a fast multiscale eigenvectors computation for drawing huge graphs. In *IEEE Symposium on Information Visualization (INFOVIS-02)*, pages 137–144, 2002.

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)*, pages 559–559, 2010.

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.

Thomas Ricatte, Rémi Gilleron, and Marc Tommasi. Hypernode Graphs for Spectral Learning on Binary Relations over Sets. In *Proceedings of the 7th European Conference on Machine Learning and Data Mining (ECML-PKDD-14)*, pages 662–677, 2014.

JA Rodríguez. On the Laplacian spectrum and walk-regular hypergraphs. *Linear and Multilinear Algebra*, 51(3):285–297, 2003.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4): 395–416, 2007.

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, 2005.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS-06)*, pages 1601–1608, 2007.

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)*, pages 912–919, 2003.

Appendix A. Regularized Hypernode Graph for Skill Rating

Formally, we assume a numbering of V such that $V = \{1, \dots, N\}$ where N is the total number of nodes, the first n nodes are the player nodes followed by the t lazy nodes, then followed by the outcome nodes, that is, $V = \{1, \dots, n\} \cup \{n+1, \dots, n+t\} \cup \{n+t+1, \dots, N\}$. 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 $\mathbb{R}^N$ where the first n entries represent the skills of the n players. Then, Problem 6 can be rewritten as

$$\begin{array}{ll} \underset{s \in \mathbb{R}^N}{\text{minimize}} & s^T \Delta s + \mu \sigma(s_p)^2 \\ \text{subject to} & \forall n+1 \leq j \leq n+t,\ s(j) = 0 \text{ (for lazy nodes)} \\ & \forall n+t+1 \leq j \leq N,\ s(j) = o_j \text{ (for outcome nodes)}, \end{array} \tag{11}$$

where μ is a regularization parameter and s_p denotes the vector of player skills $(s(1), \dots, s(n))$. 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 will 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

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

Thus, in the problem 6, 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 can be written as the minimization of $r^T \Delta_\mu r$ for some vector r and well chosen hypernode graph Laplacian Δ_μ. For this, let us consider the $p \times N$ gradient matrix G of the hypernode graph $\mathbf{h}$ associated with the set of games Γ, and let us define the $(p+n) \times (N+1)$ matrix G_μ by

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

where B is the $n \times (N+1)$ matrix defined by, for every $1 \leq i \leq n$, $B_{i,i} = -1$, $B_{i,N+1} = 1$, and 0 otherwise.

The matrix G_μ is the gradient of the hypernode graph $\mathbf{h}_\mu$ obtained from the hypernode graph $\mathbf{h}$ by: add a regularizer node $N+1$; add, for every player node, a new hyperedge

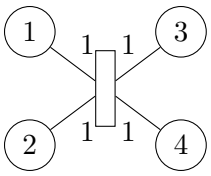
between the player node and the regularizer node $N+1$ with node weights μ/n. Note that such a hyperedge can be viewed as an edge with edge weight μ/n.

Let us denote by r the vector $(s(0),\dots,s(N),q)$, then since $\Delta = G^T G$, we can write $r^T G_\mu^T G_\mu r = s^T \Delta s + \frac{\mu}{n} r B^T B r$. As $rB^T Br = \sum_i (s_i - q)^2$, if we denote by $\Delta_\mu = G_\mu^T G_\mu$ the $(N+1)\times(N+1)$ unnormalized Laplacian of the hypernode graph $\mathbf{h}_\mu$, we can finally rewrite the regularized problem (11) as

$$\begin{array}{ll} \underset{r\in\mathbb{R}^{N+1}}{\text{minimize}} & r^T \Delta_\mu r \\ \text{subject to} & \forall n+1 \le j \le n+t,\ r(j) = 0 \text{ (for lazy nodes)} \\ & \forall n+t+1 \le j \le N,\ r(j) = o_j \text{ (for outcome nodes)} \end{array} \tag{12}$$

Appendix B. Expressiveness of the hypernode graph Laplacian

Let us consider the simple hypernode graph $\mathbf{h}$ first introduced in Figure 6 and recalled in Figure 10 below.



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

Figure 10: Hypernode graph $\mathbf{h}$ and corresponding Laplacian Δ

As stated in Section 4.2, a function is smooth on $\mathbf{h}$ if and only if it satisfies

$$(f(1) + f(2) - f(3) - f(4))^2 = 0 \tag{$\mathcal{C}$}$$

Proposition 10 *There do not exist a finite graph* $\mathbf{g}$ *whose node set contains* $\{1,2,3,4\}$ *and that satisfies the conditions:*

1. *All the smooth functions on* $\mathbf{g}$ *satisfy* $(\mathcal{C})$.
2. *Any function that satisfy* $(\mathcal{C})$ *can be extended to a smooth function on* $\mathbf{g}$.

Proof Let us define $S = \{1,3\}$ and denote by Δ the Laplacian matrix of $\mathbf{h}$. The indicator vector $\mathbf{1}_S$ is in $\mathrm{Null}(\Delta)$ and, thus, define a smooth function on $\mathbf{h}$. Let us consider a graph $\mathbf{g}_e = (N_e, E_e)$ whose nodeset N_e contains the nodeset $N = \{1,2,3,4\}$ and that satisfies the two conditions presented in Proposition 10. We denote by Δ_e the Laplacian matrix of $\mathbf{g}_e$. Because of the first condition, the function $\mathbf{1}_S$ can be extended to a smooth function f_e on $\mathbf{g}$.

Since $\mathbf{g}_e$ is a graph, we know that $\mathrm{Null}(\Delta_e)$ is spanned by the indicator vectors of the connected components of $\mathbf{g}_e$ (see for instance Von Luxburg (2007)). 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 components C of $\mathbf{g}_e$ that contains the node 1. As stated above, we have necessarily $f'e(2) = f'e(3) = f'e(4) = 0$. The indicator function f'_e of C is smooth on $\mathbf{g}_e$ but we have

$$(f'_e(1) + f'_e(2) - f'_e(3) - f'_e(4))^2 = (1 + 0 - 0 - 0)^2 \neq 0 \ .$$

Consequently, f'_e does not satisfy $(\mathcal{C})$, which violates the second condition and concludes the proof. ■

Appendix C. Proof of Lemma 8

Proof

Let us consider a hypergraph $\mathbf{h}$ with Laplacian Δ such that $\text{Null}(\Delta) = \text{Span}(\mathbf{1})$. Therefore, $\mathbb{R}^n$ is the direct sum of the space $\text{Span}(\mathbf{1})$ and of the space $\text{Null}(\Delta)^\perp$. Let us consider $f \in \mathbb{R}^n$, 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 $\mathbf{In}_j \in \text{Null}(\Delta)^\perp$. Since Δ is symmetric, $\Delta\Delta^\dagger$ is also the orthogonal projector on $\text{Null}(\Delta)^\perp$ 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. ■

Appendix D. Proof of Proposition 9

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

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

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 (10). 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 $\mathbf{In}_j=\Delta f=G^TGf$ so we can rewrite the previous equality as $d(i)=\mathbf{e}_i^TG^TGf=(G\mathbf{e}_i)^T(Gf)$. Now, because of (1), we have $G\mathbf{1}=\mathbf{0}$, thus $Gf=G(f-f(j)\mathbf{1})$. This leads to

$$d(i)=(G\mathbf{e}_i)^T(G(f-f(j)\mathbf{1})=\sum_h\left(G\mathbf{e}_i\right)(h)\cdot\left(G(f-f(j)\mathbf{1}\right)(h) \tag{13}$$

For any oriented hyperedge $h=(s_h,t_h)$, we define $\epsilon_h(i)$ to be equal to 1 if $i\in t_h$ and -1 if $i\in s_h$ (0 if the node $i.$ does not belong to the hyperedge). We can observe that we always have

$$P(h,i,j)=-\epsilon_h(i)\epsilon_h(j)\ \ . \tag{14}$$

Using this notation, we can develop the expression of $d(i)$ from Equation (13) and write

$$\begin{aligned}d(i) &= \sum_h\left(\sqrt{w_h(i)}\epsilon_h(i)\right)\cdot\left(\sum_{k\in h}\sqrt{w_h(k)}\epsilon_h(k)(f(k)-f(j))\right)\\ &= \sum_{h|i\in h}\sqrt{w_h(i)}\left\{\sum_{k\in h}(-P(h,k,i))\sqrt{w_h(k)}(f(k)-f(j))\right\}\quad\text{(see Equation (14))}\\ &= \sum_{h|i\in h}\sqrt{w_h(i)}\left\{\sum_{k\in h}(-P(h,k,i))\sqrt{w_h(k)}\mathrm{V}_j(k,j)\right\}\\ &= \mathrm{V}_j(i,j)\sum_{h|i\in h}w_h(i)(-P(h,i,i))+\sum_{h|i\in h}\sqrt{w_h(i)}\left\{\sum_{k\in h,k\neq i}(-P(h,k,i))\sqrt{w_h(k)}\mathrm{V}(k,j)\right\}\ \ .\end{aligned}$$

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

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

which concludes the proof. ■