Hypernode Graphs for Spectral Learning on Binary Relations over Sets*

Thomas Ricatte[1], Rémi Gilleron[2], and Marc Tommasi[2]

[1] SAP Research, Paris
[2] Lille University, LIFL and Inria Lille

Abstract. We introduce hypernode graphs as weighted binary relations between sets of nodes: a hypernode is a set of nodes, a hyperedge is a pair of hypernodes, and each node in a hypernode of a hyperedge is given a non negative weight that represents the node contribution to the relation. Hypernode graphs model binary relations between sets of individuals while allowing to reason at the level of individuals. We present a spectral theory for hypernode graphs that allows us to introduce an unnormalized Laplacian and a smoothness semi-norm. In this framework, we are able to extend spectral graph learning algorithms to the case of hypernode graphs. We show that hypernode graphs are a proper extension of graphs from the expressive power point of view and from the spectral analysis point of view. Therefore hypernode graphs allow to model higher order relations whereas it is not true for hypergraphs as shown in [1]. In order to prove the potential of the model, we represent multiple players games with hypernode graphs and introduce a novel method to infer skill ratings from game outcomes. We show that spectral learning algorithms over hypernode graphs obtain competitive results with skill ratings specialized algorithms such as Elo duelling and TrueSkill.

Keywords: Graphs, Hypergraphs, Semi Supervised Learning, Multiple Players Games.

1 Introduction

Graphs are commonly used as a powerful abstract model to represent binary relationships between individuals. Binary relationships between individuals are modeled by edges between nodes. This is for instance the case for social networks with the friendship relation, or for computer networks with the connection relation. The hypergraph formalism [2] has been introduced for modeling problems where relationships are no longer binary, that is when they involve more than two individuals. Hypergraphs have been used for instance in bioinformatics [11], computer vision [17] or natural language processing [3]. But, graphs and hypergraphs are limited when one has to consider relationships between sets of individual objects. A typical example is the case of multiple players games

* This work was supported by the French National Research Agency (ANR). Project Lampada ANR-09-EMER-007.

T. Calders et al. (Eds.): ECML PKDD 2014, Part II, LNCS 8725, pp. 662–677, 2014.

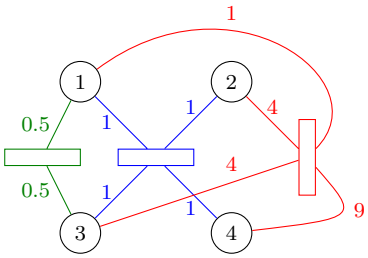


Fig. 1. A hypernode graph modeling 3 tennis games with 4 players. Each of the three hyperedges has one color and models a game for which players connected to the same long edge of a rectangle are in the same team.

where a game can be viewed as a relationship between two teams of multiple players. Other examples include relationships between groups in social networks or between clusters in computer networks. For these problems, considering both the group level and the individual level is a requisite. For instance for multiple players games, one is interested in predicting game outcomes for games between teams as well as in predicting player skills. Graphs and hypergraphs fail to model higher order relations considering both the individual level and the level of sets of individuals. This paper is a proposition to overcome this limitation.

A first contribution of this paper is to introduce a new class of undirected hypergraphs called *hypernode graphs* for modeling binary relationships between sets of individual objects. A relationship between two sets of individual objects is represented by a hyperedge which is defined to be a pair of disjoint hypernodes, where a hypernode is a set of nodes. Nodes in a hypernode of a hyperedge are given a non negative weight that represents the node contribution to the binary relationship. An example of hypernode graph is presented in Figure 1. There are four nodes that represent four tennis players and three hyperedges representing three games between teams: $\{1\}$ against $\{3\}$, $\{1,2\}$ against $\{3,4\}$, and $\{1,4\}$ against $\{2,3\}$. For each hyperedge, each player has been given a weight which can be seen as the player's contribution. It can be noted that the hyperedge between singleton sets $\{1\}$ and $\{3\}$ can be viewed as an edge between nodes 1 and 3 with edge weight 0.5. Undirected graphs are shown to be hypernode graphs where hypernodes are singleton sets.

Given a hypernode graph modeling binary relationships between sets of individuals, an important task, as said above, is to evaluate individuals by means of node labelling or node scoring functions. The second contribution of this paper is to propose machine learning algorithms in the semi-supervised, batch setting on hypernode graphs for predicting node labels or node scores. To this aim, we develop a spectral learning theory for hypernode graphs. Similarly to the case of graph spectral learning, our approach relies on the homophilic assumption [4, Chapter 4] (also called assortative mixing assumption) which says that two linked nodes should have the same label or similar scores. For graphs,

this assumption is reflected in the choice of smooth node functions for which linked nodes get values that are close enough. For hypernode graphs, we assume an additive model, and we will say that a real-valued node function over a hypernode graph is smooth if, for linked hypernodes, the weighted sum of function values over the two node sets are close enough. As an example, let us consider the blue hyperedge in Figure 1 between the two sets $\{1, 2\}$ and $\{3, 4\}$ and a real-valued node function f, the function f is said to be smooth over the hyperedge if $f(1) + f(2)$ is close to $f(3) + f(4)$.

For defining the smoothness, we introduce an unnormalized gradient for hypernode graphs. Then, we define an unnormalized Laplacian Δ for hypernode graphs by $\Delta = G^T G$ where G is the gradient. We show that the class of Laplacians of hypernode graphs is the class of symmetric positive semidefinite real-valued matrices M such that $\mathbf{1} \in \text{Null}(M)$, where $\text{Null}(M)$ denotes the null space of M and $\mathbf{1}$ is the vector full of 1's. Note that there exist hypernode graphs whose Laplacians do not match that of a graph (we can easily obtain extra-diagonal values that are positive as shown in Figure 2) whereas it has been proved in [1] that hypergraph Laplacians can be defined from graph Laplacians using adequate graph construction. The smoothness of a real-valued node function f on a hypernode graph can be characterized by the *smoothness semi-norm* defined by $\Omega(f) = f^T \Delta f$. We define the kernel of a hypernode graph to be the Moore-Penrose pseudoinverse [15] of its Laplacian. The spectral theory for hypernode graphs and its properties allow us to use spectral graph learning algorithms [16], [18], [20] for hypernode graphs.

We apply hypernode graph spectral learning to the rating of individual skills of players and to the prediction of game outcomes in multiple players games. We consider competitive games between two teams where each team is composed of an arbitrary number of players. Each game is modeled by a hyperedge and a set of games is represented by a hypernode graph. We define a skill rating function of players as a real-valued node function over the hypernode graph. And we show that finding the optimal skill rating function reduces to finding the real-valued function s^* minimizing $\Omega(s) = s^T \Delta s$, where Δ is the unnormalized Laplacian of the hypernode graph. The optimal individual skill rating function allows to compute the rating of teams and to predict game outcomes for new games. We apply this learning method on real datasets of multiple players games to predict game outcomes in a semi-supervised, batch setting. Experimental results show that we obtain very competitive results compared to specialized algorithms such as Elo duelling and TrueSkill.

Related Work. Hypernode graphs that we introduced can be viewed as an undirected version of directed hypergraphs popularized by [6] where a directed hyperedge consists in an oriented relation between two sets of nodes. As far as we know, this 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. Hypernode graphs can also be viewed as an extension of hypergraphs. The question of learning with hypergraphs has been studied and, for an overview, we refer the reader to [1]. In this paper, the authors show that various

formulations of the semi-supervised and the unsupervised learning problem on hypergraphs can be reduced to graph problems. For instance, the hypergraph Laplacian of [19] can be defined as a graph Laplacian by an adequate graph construction. To the best of our knowledge, no hypergraph Laplacian which cannot be reduced to a graph Laplacian has been defined so far. A very recent tentative to fully use the hypergraph structure was proposed by [8]. In this paper, the authors propose to use the hypergraph cut, and they introduce the total variation on a hypergraph as the Lovasz extension of the hypergraph cut. This allows to define a regularization functional on hypergraphs for defining semi-supervised learning algorithms.

2 Graphs and Hypernode Graphs

2.1 Undirected Graphs and Laplacians

In the following, we recall the commonly accepted definitions of undirected graphs and graph Laplacians. An *undirected graph* $\mathbf{g} = (V, E)$ is a set of nodes V with $|V| = n$ together with a set of undirected edges E with $|E| = p$. Each edge $e \in E$ is an unordered pair $\{i, j\}$ of nodes and has a non negative weight $w_{i,j}$. In order to define the smoothness of a real-valued node function f over a graph $\mathbf{g}$, we define the gradient function grad for f by, for every edge (i, j),

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

We can note that $|\,\text{grad}(f)(i, j)|$ is small whenever $f(i)$ is close to $f(j)$. Then, the smoothness of a real-valued node function f over a graph $\mathbf{g}$ is defined by

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

where G is the matrix of the linear mapping grad from $\mathbb{R}^n$ into $\mathbb{R}^p$. The symmetric matrix $\Delta = G^T G$ is called *undirected graph Laplacian*, which is also proved to be defined by $\Delta = D - W$ where D is the degree matrix of $\mathbf{g}$ and W the weight matrix of $\mathbf{g}$. $\Omega(f) = f^T \Delta f$ has been used in multiple works (see for example [20], [16]) to ensure the smoothness of a node labeling function$f$.

Additional information concerning the discrete analysis on graphs can be found in [18], which develop a similar theory with a normalized version of the gradient and Laplacian (G is replaced by $GD^{-1/2}$).

2.2 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 *with* $|V| = n$ *and a set of hyperedges* H *with* $|H| = p$. *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*

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

An example of hypernode graph is shown in Figure 1. The red hyperedge links the sets $\{1, 4\}$ and $\{2, 3\}$. The weights satisfy the Equilibrium condition which ensures that constant node functions have a null gradient as we will see in the next section. The green hyperedge is an unordered pair $\{\{1\}, \{3\}\}$ of two singleton sets with weights 0.5 for the nodes 1 and 3. It can be viewed as an edge between nodes 1 and 3 with edge weight 0.5. Indeed, when a hyperedge h is an unordered pair $\{\{i\}, \{j\}\}$ involving only two nodes, the Equilibrium Condition states that the weights $w_h(i)$ and $w_h(j)$ are equal. Thus, such a hyperedge can be seen as an edge with edge weight $w_{i,j} = w_h(i) = w_h(j)$. Therefore, a hypernode graph such that every hyperedge is an unordered pair of singleton nodes can be viewed as an undirected graph, and conversely.

2.3 Hypernode Graph Laplacians

In this section, we define the *smoothness* of a real-valued node function f over a hypernode graph with the gradient that we define now.

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*

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

where an arbitrary orientation of the hyperedges has been chosen.

As an immediate consequence of the gradient definition and because of the Equilibrium Condition, the gradient of a constant node function is the zero-valued hyperedge function. Also, it can be noted that, for a hyperedge $h \in H$, $|\mathrm{grad}(f)(h)|^2$ is small when the weighted sum of the values $f(i)$ for nodes i in s_h is close to the weighted sum of the values $f(j)$ for nodes j in t_h. Thus, if we denote by $G \in \mathbb{R}^{p \times n}$ the matrix of grad, the smoothness of a real-valued node function f over a hypernode graph $\mathbf{h}$ is defined by $\Omega(f) = f^T G^T G f$.

Let $\mathbf{h}$ be a hypernode graph with unnormalized gradient G, the square $n \times n$ real valued matrix $\Delta = G^T G$ is defined to be the *unnormalized Laplacian* of the hypernode graph $\mathbf{h}$. It should be noted that, as in the graph case, the Laplacian Δ does not depend on the arbitrary orientation of the hyperedges used for defining the gradient. When the hypernode graph is a graph, the unnormalized hypernode

graph Laplacian matches the unnormalized graph Laplacian. Last, we define the *hypernode graph kernel* of a hypernode graph $\mathbf{h}$ to be the Moore-Penrose pseudoinverse $\Delta^\dagger$ [15] of the hypernode graph Laplacian Δ .

2.4 Hypernode Graph Laplacians and Learning

We can characterize hypernode graph Laplacians by

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

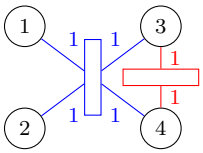
Proof. It is an immediate consequence of the definitions of the hypernode graph gradient and the hypernode graph Laplacian that a hypernode graph Laplacian is a symmetric positive semidefinite real-valued matrix, and that a constant function has a null gradient. For the other direction, let us consider a symmetric positive semidefinite real-valued matrix M such that $\mathbf{1} \in \mathrm{Null}(M)$. Then, consider a square root decomposition $M = G^T G$ of M. For each line of G, one can define a hyperedge $h = \{s_h, t_h\}$ with s_h the set of nodes with positive values in the line of G, t_h the set of nodes with negative values in the line of G, and weights equal to the square of values in the line of G. The Equilibrium condition is satisfied because $\mathbf{1} \in \mathrm{Null}(M)$ and it is easy to verify that the Laplacian of the resulting hypernode graph $\mathbf{h}$ is M.

As a consequence of the construction in the previous proof, it should be noted that there are several hypernode graphs with the same hypernode graph Laplacian because the square root decomposition is not unique. One can also find hypernode graphs whose Laplacian matches that of a graph. One can prove that this is not however the general case. For this, it suffices to consider a hypernode graph Laplacian with an extradiagonal term which is positive. For instance, consider the hypernode graph and its Laplacian matrix Δ in Figure 2, the Laplacian matrix has 1 as extradiagonal term, thus Δ is not a graph Laplacian.

As said in Proposition 1, hypernode graph Laplacians are positive semidefinite. This allows to leverage most of the spectral learning algorithms defined in [16] , [18], [20] from graphs to hypernode graphs. Note, however, that spectral hypernode graph learning can not be reduced to spectral graph learning since hypernode graph Laplacians are strictly more general than graph Laplacians.

2.5 Hypernode Graph Laplacians and Signed Graphs

In this section we present additional properties of hypernode graph Laplacians and kernels. As in the graph case, we have defined the kernel of a hypernode graph to be the Moore-Penrose pseudoinverse of its Laplacian. Because the pseudoinversion preserves semidefiniteness and symmetry, as a consequence of Proposition 1, one can show that the class of hypernode graph kernels is closed under the pseudoinverse operation. As a consequence, the class of hypernode graph kernels is equal to the class of hypernode graph Laplacians. It is worth noticing that the class of graph kernels is not closed by pseudoinversion.



$$\Delta = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix} \quad W = \begin{pmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

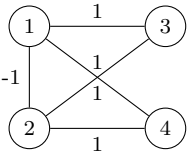


Fig. 2. From left to right : a hypernode graph, its Laplacian Δ, the pairwise weight matrix W, and the corresponding signed graph

It can also be shown that the class of hypernode graph Laplacians is closed by convex linear combination. This is an important property in the setting of learning from different sources of data. As graph kernels are hypernode graph kernels, it should be noted that the convex linear combination of graph kernels is a hypernode graph kernel, while it is not a graph kernel in general because the class of graph kernels is not closed by convex linear combination. This explains why problems for hypernode graphs can not be solved using graph constructions.

We have shown above that there does not exist in general a graph whose Laplacian is equal to the Laplacian of a given hypernode graph. Nevertheless, given a hypernode graph $\mathbf{h}$ and its Laplacian Δ, using Proposition 1, one can define a symmetric matrix W of possibly negative weights for pairs of nodes of $\mathbf{h}$ such that $\Delta = D - W$, where D is the degree matrix associated with W (the construction is illustrated in Figure 2). This means that, for every hypernode graph $\mathbf{h}$, there is a unique signed graph with weight matrix W such that $D - W$ is the hypernode graph Laplacian of $\mathbf{h}$. This result highlights the subclass of signed graphs whose Laplacian computed with the formula $D - W$ is positive semidefinite. This result also shows that homophilic relations between sets of nodes lead to non homophilic relations between nodes.

3 Hypernode Graph Model for Multiple Players 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 game outcomes. A second objective is to predict a game outcome from a batch of games with their outcomes. For that, we will model games by hyperedges assuming that the performance of a team is the sum of the performances of its members as done by the team model proposed in [9].

3.1 Multiplayer Games

Let us consider a set of individual players $P = \{1, \ldots, n\}$ and a set of games $\Gamma = \{\gamma_1, \ldots, \gamma_p\}$ between two teams of players. Let us also consider that a player i contributes to a game γ_j with a non negative weight $c_j(i)$. We assume that each player has a skill $s(i)$ and that a game outcome can be predicted by comparing the weighted sum of the skills of the players of each of the two teams. More formally,

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

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

Equivalently, one can rewrite this inequality by introducing a non negative real number o_j on the right hand side such that

$$\sum_{i=1}^{\ell} c_j(a_i)s(a_i) = o_j + \sum_{i=1}^{k} c_j(b_i)s(b_i) \ . \tag{2}$$

The real number o_j quantifies the game outcome. In the case of a draw, the game outcome o_j is set to 0. Given a set of games, it may be impossible to assert that all constraints (1) can be simultaneously satisfied. Our goal is to estimate a skill rating function $s \in \mathbb{R}^n$ that respects the game outcomes in Γ as much as possible. We define the cost of a game γ_j with outcome o_j for a skill function s by

$$C_{\gamma_j}(s) = \| \sum_{i=1}^{\ell} c_j(a_i)s(a_i) - \sum_{i=1}^{k} c_j(b_i)s(b_i) - o_j \|^2 \ .$$

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

$$s^* = \arg\min_s \sum_{\gamma_j \in \Gamma} C_{\gamma_j}(s) \ . \tag{3}$$

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 $A = \{1, 2\}$ 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 equal skills for all players should lead to a draw. 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 (2), that is $s(1) + s(2) = o + s(3) + s(4)$ where $o > 0$ represents the outcome of the game γ. In order to model the game outcome in the hyperedge, we introduce a virtual player H that plays along with team B with a weight equal to 1 and we fix the skill rating function on H to be $s(H) = o > 0$. The virtual player is modeled by a node H, called outcome node, added to the set $\{3, 4\}$. Last, for the hyperedge to satisfy the equilibrium condition, we add a node Z, called lazy node, to the set $\{1, 2\}$. In this example, the weight of Z is set to 1. The skill $s(Z)$ of the lazy node Z is fixed to be 0 such as the equation

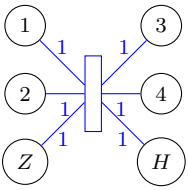


Fig. 3. Hyperedge h for a game γ between team $A = \{1, 2\}$ and $B = \{3, 4\}$. A wins and there is an additional outcome node H for the virtual player and an additional lazy node for the lazy virtual player. The node contributions are set to 1.

between skills can be rewritten as $s(1) + s(2) + s(Z) = s(3) + s(4) + s(H)$. And 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 represented in Figure 3 where s satisfies $s(H) = o$ and $s(Z) = 0$.

In the general case, let us consider a set of individual players $P = \{1, \ldots, n\}$, a set of games $\Gamma = \{\gamma_1, \ldots, \gamma_p\}$. Each game γ_j is between two teams (sets of players) A_j and B_j, the winning team is known as well as the game outcome o_j. Let us also consider that a player i contributes to a game γ_j with a non negative weight $c_j(i)$. We can define, for every game γ_j a hyperedge h_j as follows

1. The players of A_j define one of the two hypernodes of h_j. The weight of a player node i is defined to be $c_j(i)^2$,
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 define the hypernode graph $\mathbf{h} = (V, H)$ as the set of all hyperedges h_j for the games γ_j in Γ as defined above. Now, skill rating functions of players correspond to real-valued node 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 function s over $\mathbf{h}$ must thus satisfy, for every lazy node Z_j, the function value $s(Z_j)$ is 0, and, for every outcome node H_j of game γ_j, the function value $s(Z_j)$ is the outcome o_j.

Formally, we assume a numbering of V such that $V = \{1, \ldots, N\}$ 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, $V = \{1, \ldots, n\} \cup \{n+1, \ldots, n+t\} \cup \{n+t+1, \ldots, N\}$. 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, it is

easy to show that the *skill rating problem* (3) is equivalent to find the optimal vector s solving the optimization problem

$$\begin{array}{ll} \underset{s\in\mathbb{R}^N}{\text{minimize}} & s^T\Delta s \\ \text{subject to} & \forall n+1 \leq j \leq n+t,\ s(j)=0 \text{ (for lazy nodes)} \\ & \forall n+t+1 \leq j \leq N,\ s(j)=o_j \text{ (for outcome nodes)} \end{array} \tag{4}$$

3.3 Regularizing the Hypernode Graph

When the number of games is small, many players will participate to at most one game. Thus, in this case, the number of connected components can be quite large. The player skills in every connected component can be defined independently while satisfying the constraints. Thus, it will be irrelevant to compare player skills in different connected components. In order to solve this issue, we introduce in Equation (4) a regularization term based on the standard deviation of the players skills $\sigma(s_p)$, where $s_p = (s(1), \ldots, s(n))$. This leads to the new formulation

$$\begin{array}{ll} \underset{s\in\mathbb{R}^N}{\text{minimize}} & s^T\Delta s + \mu\sigma(s_p)^2 \\ \text{subject to} & \forall n+1 \leq j \leq n+t,\ s(j)=0 \text{ (for lazy nodes)} \\ & \forall n+t+1 \leq j \leq N,\ s(j)=o_j \text{ (for outcome nodes)}, \end{array} \tag{5}$$

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 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 $s_p = (s(0), \ldots, s(n))$, then, for all $q \in \mathbb{R}$, we have

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

Thus, in the problem 5, 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 matrix G_μ by

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

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 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 with the hypernode graph reduced to a single hyperedge of Figure 3.

Let us denote by r the vector $(s(0), \ldots, s(N), q)$, 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 $r B^T B r = \sum_i (s_i - q)^2$, 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 (5) as

$$\begin{array}{ll} \underset{r \in \mathbb{R}^{N+1}}{\text{minimize}} & r^T \Delta_\mu r \\ \text{subject to} & \forall n+1 \leq j \leq n+t,\ r(j) = 0 \text{ (for lazy nodes)} \\ & \forall n+t+1 \leq j \leq N,\ r(j) = o_j \text{ (for outcome nodes)} \end{array} \tag{6}$$

3.4 Inferring Skill Ratings and Predicting Game Outcomes

We have shown that predicting skill ratings can be written as the optimization problem (6). 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. Using Proposition 1, we get that Δ_μ is a positive semidefinite real-valued matrix because it is a hypernode graph Laplacian. Therefore, we can use the semi-supervised learning algorithm presented in [20]. This algorithm was originally designed for graphs and solves exactly the problem (6) by putting hard constraints on the outcome nodes and on the lazy nodes. We denote this method by H-ZGL.

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. We denote this method by H-SVR.

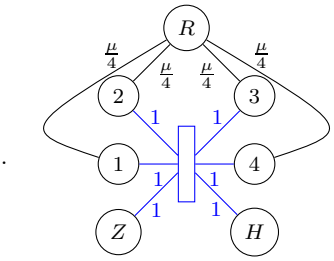


Fig. 4. Adding a regularizer node R to the hypergraph of Figure 3

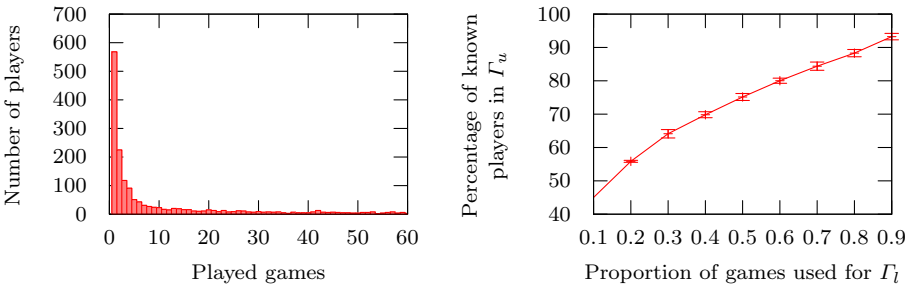


Fig. 5. [*left*] Distribution of the number of players against the number of played games; [*right*] Average percentage of players in Γ_u which are involved in some game in Γ_l

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 [9], [5] while we consider the batch setting. For the prediction of game outcomes, first we apply a skill rating prediction algorithm presented above given the training set Γ_l and output a skill rating function s^*. Then, for each game in Γ_u, we evaluate the inequality (1) with the skills defined by s^* and decide the winner. For every player which do not appear in the training set, the skill value is fixed a priori to the mean of known player skills.

Algorithm 1. Predicting game outcomes

Input: 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 3.2 and 3.3
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 (1) and predict the winner
7: **end for**

4 Experiments

4.1 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 consists in 10028 games with 1834 players.

In every experiment, we select randomly a training subset Γ_l of games and all 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. We present in Figure 5 several statistics related to the Tennis dataset. It is worth noticing that many players have played only 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. Moreover, it should be noted that when the number of games in the training set is small, the number of players in the test set which are involved in a game of the training set is small. In this case many players will have a skill estimated to be the average skill.

Given a training set of games Γ_l and a test set Γ_u, we follow the experimental process described in Algorithm 1. For the definition of the hypergraph, we fix all player contributions in games to 1 because we do not have additional information than final scores. Thus the player nodes weights in every hyperedge are set to 1. In the optimization problem 6, the game outcomes o_j are defined to be the difference between the number of sets won by the two teams. This allows to take account of the score when computing player skills. In order to reduce the number of nodes, all lazy nodes are merged in a single one that is shared by all the hyperedges. We do the same for outcome nodes because score differences can be 1, 2 or 3. The resulting hypernode graph has at most 1839 nodes: at most 1834 player nodes, 1 lazy node, 3 outcome nodes, and 1 regularizer node.

To complete the definition of the hypernode graph $\mathbf{h}_\mu$ constructed from the game set Γ_l, it remains to fix the regularization node weights μ/n, i.e. fix the value of the regularization parameter μ. For this, assuming a Gaussian distribution for skill ratings and comparing expected values for the two terms $s^T \Delta s$ and $\mu\sigma(s_p)^2$, we can show that the value of μ/n should have the same order of magnitude than the average number of games played by a player. We fix the default value to be 16 for μ/n and use this default value in all experiments.

Given $\mathbf{h}_\mu$, following Algorithm 1, 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 Duelling and Trueskill [9][1] Then, we predict game outcomes from the inferred skill ratings. The results are given in Figure 6 (for each value of ρ, we repeat the experiment 10 times). It can be noted that Elo duelling performs poorly. Also, it can be noted that H-ZGL is significantly better than Trueskill whatever is the chosen proportion.

[1] TrueSkill and Elo implementations are from [7]. Results were double-checked using [14] and [13]. Parameters of Elo and TrueSkill are the default parameters of [7] ($K = 32$ for Elo, $\mu_0 = 25, \beta = 12.5, \sigma = 8.33$ and $\tau = 0.25$ for TrueSkill).

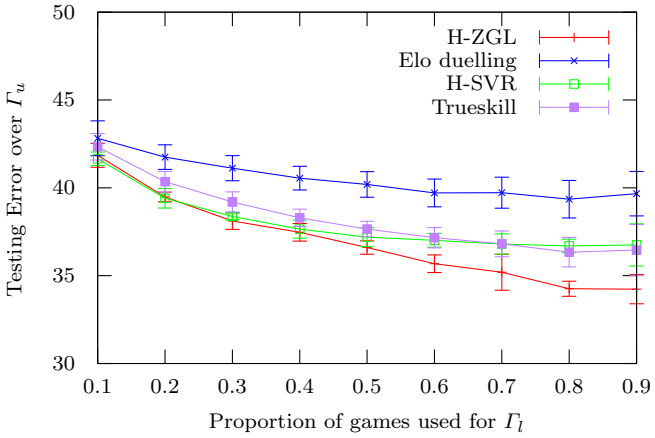


Fig. 6. Predictive error depending on the proportion of games used to build Γ_l

4.2 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 [9] 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 [9], 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 again consider the experimental process described in Algorithm 1. As for the Tennis dataset, we fix all players contributions in games to 1. In the optimization problem 6, the game outcomes o_j are defined to be equal to 1 when the game has a winner and 0 otherwise because game scores in vary depending on the type of game. As above, we merge the lazy nodes into a single one and do the same for outcome nodes. The value of μ/n is again set to 16.

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 [12]. We present the experimental results in Figure 7. 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, and we did not search for the better value of C.

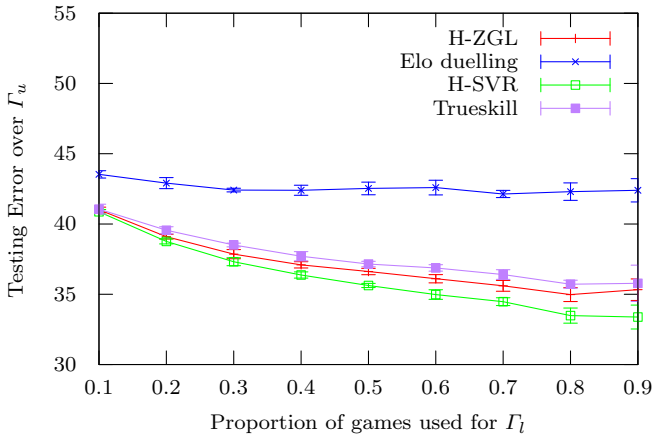


Fig. 7. Predictive error depending on the proportion of games used to build Γ_l

5 Conclusion

We have introduced hypernode graphs, defined a spectral theory for hypernode graphs, and presented an application to the problem of skill rating and game outcome prediction in multiple players games. This paper opens many research questions both from a theoretical perspective and from an applicatory perspective. First, the class of directed hypernode graphs should be investigated from a machine learning perspective. Second, following [8], it should be interesting to study the notion of cut for hypernode graphs. Third, we should define online learning algorithms for hypernode graphs following [10] which would be useful for large datasets for massive online games. Last, we are confident in the capability of our model to handle new applications in networked data.

References

1. Agarwal, S., Branson, K., Belongie, S.: Higher Order Learning with Graphs. In: Proc. of ICML, pp. 17–24 (2006)
2. Berge, C.: Graphs and hypergraphs. North-Holl Math. Libr. North-Holland, Amsterdam (1973)
3. Cai, J., Strube, M.: End-to-end coreference resolution via hypergraph partitioning. In: Proc. of COLING, pp. 143–151 (2010)
4. Easley, D., Kleinberg, J.: Networks, crowds, and markets: Reasoning about a highly connected world. Cambridge University Press (2010)
5. Elo, A.E.: The Rating of Chess Players, Past and Present. Arco Publishing (1978)
6. Gallo, G., Longo, G., Pallottino, S., Nguyen, S.: Directed hypergraphs and applications. Discrete Applied Mathematics 42(2-3), 177–201 (1993)
7. Hamilton, S.: PythonSkills: Implementation of the TrueSkill, Glicko and Elo Ranking Algorithms (2012), `https://pypi.python.org/pypi/skills`

8. Hein, M., Setzer, S., Jost, L., Rangapuram, S.S.: The Total Variation on Hypergraphs - Learning on Hypergraphs Revisited. In: Proc. of NIPS, pp. 2427–2435 (2013)
9. Herbrich, R., Minka, T., Graepel, T.: TrueSkill^{TM}: A Bayesian Skill Rating System. In: Proc. of NIPS, pp. 569–576 (2006)
10. Herbster, M., Pontil, M.: Prediction on a Graph with a Perceptron. In: Proc. of NIPS, pp. 577–584 (2006)
11. Klamt, S., Haus, U.-U., Theis, F.: Hypergraphs and Cellular Networks. PLoS Computational Biology 5(5) (2009)
12. Lasek, J., Szlávik, Z., Bhulai, S.: The predictive power of ranking systems in association football. International Journal of Applied Pattern Recognition 1(1), 27–46 (2013)
13. Lee, H.: Python implementation of Elo: A rating system for chess tournaments (2013), `https://pypi.python.org/pypi/elo/0.1.dev`
14. Lee, H.: Python implementation of TrueSkill: The video game rating system (2013), `http://trueskill.org/`
15. Penrose, R.: A generalized inverse for matrices. In: Proc. of Cambridge Philos. Soc., vol. 51, pp. 406–413. Cambridge University Press (1955)
16. Von Luxburg, U.: A tutorial on spectral clustering. Statistics and computing 17(4), 395–416 (2007)
17. Zhang, S., Sullivan, G.D., Baker, K.D.: The automatic construction of a view-independent relational model for 3-D object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(6), 531–544 (1993)
18. Zhou, D., Huang, J., Schölkopf, B.: Learning from labeled and unlabeled data on a directed graph. In: Proc. of ICML, pp. 1036–1043 (2005)
19. Zhou, D., Huang, J., Schölkopf, B.: Learning with hypergraphs: Clustering, classification, and embedding. In: Proc. of NIPS, pp. 1601–1608 (2007)
20. Zhu, X., Ghahramani, Z., Lafferty, J., et al.: Semi-supervised learning using gaussian fields and harmonic functions. In: Proc. of ICML, vol. 3, pp. 912–919 (2003)