

# SimpleHypergraphs.jl — Novel Software Framework for Modelling and Analysis of Hypergraphs <sup>\*</sup>

Alessia Antelmi<sup>1</sup>, Gennaro Cordasco<sup>2</sup>, Bogumił Kamiński<sup>3</sup>, Paweł Prałat<sup>4</sup>,  
Vittorio Scarano<sup>2</sup>, Carmine Spagnuolo<sup>2</sup>, and Przemysław Szufel<sup>3</sup>

<sup>1</sup> Dipartimento di Informatica, Università degli Studi di Salerno, Italy  
`aantelmi@unisa.it, vitsca@unisa.it, cspagnuolo@unisa.it`

<sup>2</sup> Dipartimento di Psicologia, Università degli Studi della Campania “Luigi Vanvitelli”, Italy `gennaro.cordasco@unicampania.it`

<sup>3</sup> SGH Warsaw School of Economics, Poland `bkamins@sgh.waw.pl,`  
`pszufe@sgh.waw.pl`

<sup>4</sup> Department of Mathematics, Ryerson University, Toronto, ON, Canada  
`pralat@ryerson.ca`

**Abstract.** Hypergraphs are natural generalization of graphs in which a single (hyper)edge can connect any number of vertices. As a result, hypergraphs are suitable and useful to model many important networks and processes. Typical applications are related to social data analysis and include situations such as exchanging emails with several recipients, reviewing products on social platforms, or analyzing security vulnerabilities of information networks. In many situations, using hypergraphs instead of classical graphs allows us to better capture and analyze dependencies within the network. In this paper, we propose a new library, named `SimpleHypergraphs.jl`, designed for efficient hypergraph analysis. The library exploits the Julia language flexibility and direct support for distributed computing in order to bring a new quality for simulating and analyzing processes represented as hypergraphs. In order to show how the library can be used we study two case studies based on the Yelp dataset. Results are promising and confirm the ability of hypergraphs to provide more insight than standard graph-based approaches.

**Keywords:** Hypergraphs · Modelling hypergraphs · Software library · Julia programming language

## 1 Introduction

Many human-technology interaction situations generate data that can be viewed, based on the type of interaction, as a self-organizing network. In these networks (for example, the Yelp on-line social network) nodes not only contain some useful information (such as user’s profile, photos, reviews) but are also internally

---

<sup>\*</sup> The research is financed by NAWA — The Polish National Agency for Academic Exchange.

connected to other nodes (relations based on similar user’s behaviour, similar taste, age, geographic location). Indeed, the proliferation of cellular usage has given rise to massive amounts of data that, through data mining and analytics, promises to reveal a wealth of information on how users interact with one another and shape the preferences of others.

Hypergraphs are of particular interest in the field of knowledge discovery where most problems currently modelled as graphs would be more accurately modelled as hypergraphs. Indeed, hypergraphs are natural generalization of graphs where one edge consists of several vertices instead of just a pair of vertices. This feature makes the hypergraphs particularly useful for modeling real world systems in which many references occur simultaneously. Examples include sending emails to many people, co-authorships of scientific publications, or several parties participating in a crypto-currency transaction. All of these complex real-world systems can be efficiently modelled with hypergraphs. Moreover, hypergraphs can also be extremely helpful, in computational social science, for the development of computer simulations [18,19]. Indeed, hypergraphs can be used to model any complex interaction among a group of simulated agents. Despite this fact, the theory and tools are not sufficiently developed to allow most problems to be tackled directly within this context.

The goal of this paper is to introduce a new library designed for efficient hypergraph analysis in the Julia language named `SimpleHypergraphs.jl`. The library makes an excessive use and is built on top of `LightGraphs.jl`, which is an efficient high-performance engine for graph analytics. Combined with language flexibility and direct support for distributed computing, the `SimpleHypergraphs.jl` library can bring a new quality for simulating and analyzing processes represented as hypergraphs.

The paper is structured as follows. In Section 2, we start with a review of the existing frameworks dealing with hypergraphs, describe the motivation and introduce the `SimpleHypergraphs.jl` library and its functionality. In Section 3, a use case with analysis of Yelp reviews is presented with the aim to show a real-life application of the developed library. Finally, we sum up the paper in the Conclusions section.

## 2 Modelling and hypergraphs with `SimpleHypergraphs.jl`

In this section we start by introducing motivation for the hypegraph library and next we move towards describing its functionality.

### 2.1 Motivation

Despite the fact that hypergraphs are natural representations of many real-world systems, there are currently very few software frameworks that are suitable for modelling and mining hypergraphs. In this Section, we give a brief overview of several software libraries, focusing on their code availability and capability in modelling and analyzing hypergraphs.

- **Chapel HyperGraph Library (CHGL)** [3] that has been developed by the Pacific Northwest National Laboratory since 2018 and is released under

MIT license. CHGL is a library for the emerging parallel language Chapel. The library provides the `AdjListHyperGraph` module that allows to store hypergraphs on shared and distributed memory. The library is not well documented and does not provide an easy mechanism for the 2-section and bipartite view analyses. However, it is worth mentioning for its functionality for parallel and distributed computing.

- **HyperX** [6] is a scalable framework for processing hypergraphs and learning algorithm built on top of Apache Spark. This library supports the same design model of GraphX, the Apache Spark API for graphs and graph-parallel computation written in Scala language. An interesting feature of this library is that it provides native support for hypergraph elaboration. The standard approach uses the bipartite or the 2-section representation of hypergraphs and exploits GraphX library, while HyperX directly processes hypergraph data obtaining significant speedup compared to the standard approach.
- **Pygraph** [11] is a pure Python library for graph manipulation released under the MIT license. It has almost all basic functionalities on graphs implemented but also supports hypergraphs by exposing the class `hypergraph`. This library does not provide any specific optimization and functionalities for hypergraphs.
- **Multihypergraph** [9] is a Python package for graphs released under GPL license. The library emphasizes the mathematical understanding of graphs rather than the algorithmic efficiency and provides support for hyper-edges, multi-edges, and looped-edges. This library provides only graph model memory definition and isomorphism functionalities without implementing any other functionalities and algorithms for graphs and hypergraphs.
- **HyperNetX** [5] is a Python preliminary library released in 2018 under the Battelle Memorial Institute licence<sup>5</sup>. The library generalizes traditional graph metrics (such as vertex and edge degrees, diameter, distance, etc.) to hypergraphs, and provides good documentation and tutorials. The library supports the bipartite representation of a hypergraph, along with the possibility to load hypergraphs from their bipartite view. Furthermore, it provides some simple visualization functionalities for hypergraph.
- **Halp** [2] is a Python software package providing both a directed and an undirected hypergraph implementation as well as several important and classical algorithms. The library is developed by Murali’s Research Group at Virginia Tech released under GPL license. The library provides several statistics on hypergraphs and model transformations in graphs supported by the NetworkX Python library. In addition, several algorithms for hypergraphs, such as  $k$ -shortest-hyperpaths, random walk, directed paths, are implemented.
- **HyperGraphLib** [4] is a C++ implementation of hypergraphs that exploits the Boost Library, which also defines the library license. This library provides basic functionalities for hypergraphs and implements some simple metrics. Moreover, it also provides some isomorphisms functionalities and path-finding algorithms. However, it does not implement any kind of hyper-

<sup>5</sup> <https://github.com/pnml/HyperNetX/blob/master/LICENSE.rst>

graph representations (such as bipartite or 2-section) nor software integration with other graph libraries.

- **Iper** [7] is a JavaScript library for hypergraphs released under MIT license. The library provides the definitions of hypergraphs and allows the user to define meta information for vertices. However, it does not include any kind of hypergraph transformation/representations and integration with other graph libraries for classical statistics and algorithms.
- **NetworkR** [10] is an R package with a set of functions for analyzing social and economic networks including hypergraphs. It includes analyses such as degree distribution, diameter, and density of the network, as well as microscopic level analysis such as power, influence, and centrality of individual nodes. The library does not provide support for meta information on vertices and hyperedges and provides only hypergraphs projection into graphs.
- **Gspbox** [1] is an easy to use Matlab toolbox that performs a wide variety of operations on a graph. It is based on spectral graph theory and many of the implemented features can scale to very large graphs. Gspbox supports hypergraphs modeling, including ability for hyperedges to have weights assigned, and for vertices to have coordinates in the space. The hypergraph manipulation is obtained by representing the model as a graph. For this reason, despite the fact that all graph functionalities are available, the library does not provide any kind of specific solutions or optimization for hypergraphs.

Overall, all the considered libraries settles a compromise between efficiency (which characterizes low level languages, such as C/C++) and easy-of-use/ expressiveness (which characterized interpreted and/or scripting languages like Python and R).

In this work, we are proposing a library, which exploiting the Julia language ensures both efficiency and expressiveness. Julia is a new programming language developed at MIT [16]. The language uses a syntax similar to popular and easy-to-use scientific computing languages like Python or R. This means that experience in those languages can be directly applied in Julia by computational scientists [21,29]. Moreover, a distinguishing feature of Julia is that while keeping mathematics-oriented syntax it makes it possible to compile the code to a binary form. In result it means that the observed performance of Julia programs is very similar to that of C++, however with around 4 times less lines of code.

The library `SimpleHypergraphs.jl` is available on a GitHub public repository<sup>6</sup>, where the library documentation is also provided<sup>7</sup>. Additionally, several tutorials are available in the form of Jupyter Notebooks<sup>8</sup>. This Section describes the library design and motivations behind its implementation. Furthermore, library functionalities of the 1.0 version will be discussed.

<sup>6</sup> <https://github.com/pszufe/SimpleHypergraphs.jl>.

<sup>7</sup> <https://pszufe.github.io/SimpleHypergraphs.jl/latest/reference/>.

<sup>8</sup> <https://tinyurl.com/y5btobdk>.

## 2.2 Definitions and Notation

Hypergraphs are natural generalization of well-known and widely used graphs. Formally, a hypergraph is an ordered pair  $H = (V, E)$  where  $V$  is a set of vertices and  $E$  is a set of edges. Each edge is a non-empty subset of vertices; that is,  $E \subseteq 2^V \setminus \{\emptyset\}$ , where  $2^V$  is the power set of  $V$ . We will use  $n = |V|$  and  $m = |E|$  for the size of the vertex set and, respectively, the edge set. Indeed, hypergraphs are generalization of graphs in which each edge is a two element subset of  $V$ ; that is, hypergraph  $G = (V, E)$  is a graph if  $E \subseteq \binom{V}{2} \subseteq 2^V \setminus \{\emptyset\}$ .

## 2.3 Library design and functionalities

`SimpleHypergraphs.jl` represents a hypergraph  $H = (V, E)$  as an  $n \times k$  matrix, where  $n$  is the number of vertices and  $k$  is the number of hyperedges. In other words, each row of the matrix is associated with a vertex and indicates the hyperedges the vertex belongs to. The proposed library stores in-memory a hypergraph using its matrix representation. Vertices and hyperedges are uniquely identified by progressive integer ids, corresponding to rows  $(1, \dots, n)$  and columns  $(1, \dots, k)$ , respectively. Each position  $(i, j)$  of the matrix denotes the weight of vertex  $i$  within the hyperedge  $j$ . In addition, the library provides several constructors for defining meta information type and enables to attach meta-data values of arbitrary type to both vertices and hyperedges.

The library APIs are designed in similar fashion of the popular library for graph manipulation `LightGraphs.jl`, this provide to the programmers a familiar environment.

**Hypergraph constructors.** Based on the previous consideration, the Julia hypergraph object is defined as:

```
Hypergraph{T, V, E} <: AbstractMatrix{Union{T, Nothing}}
```

where  $T$  represents the type of the weights stored in the structure while  $V$  and  $E$  are the type of values stored in the vertices and edges of the hypergraph, respectively.

**Functions.** `SimpleHypergraphs.jl` provides several accessing and manipulating functions:

- `add_vertex!`, adds a vertex to a given hypergraph  $H$ . Optionally, the vertex can be added to existing hyperedges. Additionally, a value can be stored with the vertex using the `vertex_meta` keyword parameter.
- `set_vertex_meta!`, sets a new meta value `new_value` for vertex `id` in  $H$ .
- `get_vertex_meta`, returns a meta value stored at vertex `id` in  $H$ .
- `get_vertices`, returns vertices from a  $H$  for a given hyperedge  $he_{id}$ .

The same functionalities are provided for the hyperedges.

**Hypergraph Transformations.** The library provides two hypergraph transformations into the corresponding graph representation:

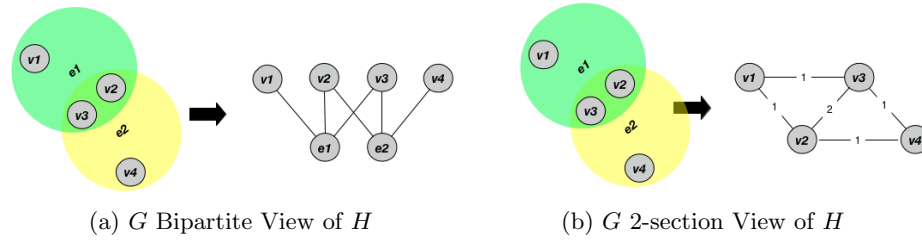


Fig. 1: (H)ypergraph transformations.

1. **BipartiteView** is a bipartite representation of a hypergraph  $H$ . As described in Bretto [17], this representation is an incidence graph of hypergraph  $H = (V, E)$ ; that is, a bipartite graph  $IG(H)$  with vertex set  $S = V \cup E$ , and where  $v \in V$  and  $e \in E$  are adjacent if and only if  $v \in e$ . Figure 1a (on the left) depicts a simple example of bipartite view.
2. **TwoSectionView** is a 2-section representation of a hypergraph  $H$ . As described in Bretto [17], this representation of a hypergraph  $H = (V, E)$ , denoted by  $[H]_2$ , is a graph whose vertices are the vertices of  $H$  and where two distinct vertices form an edge if and only if they are in the same hyperedge of  $H$ . As a result, each hyperedge from  $H$  occurs as a complete graph in  $G$ . The weight of an edge corresponds to the number of hyperedges that contain both the endpoints of the edge. Figure 1b (on the right) shows a simple example of the 2-section view.

Both **Views** are instances of the **AbstractGraph** graph object defined by the **LightGraphs.jl** library [8]. When the view is materialized—according to **LightGraphs.jl** specifics—the generated graph does not include any meta information.

**Hypergraph I/O.** The library currently offers a basic mechanism to load/save a hypergraph from/to a stream. Given hypergraph  $H$  is stored using the following format. The first line consists of  $n$  and  $k$ , the number of vertices and, respectively, the number of edges of  $H$ . The following  $k$  rows describe the actual structure of  $H$ . Each row represents one hyperedge as a list of all vertex-weight pairs within that hyperedge.

## 2.4 Hypergraph Modularity

One of the most important properties of complex networks is their community structure, that is, the organization of vertices in clusters, with many edges joining vertices of the same cluster and comparatively few edges joining vertices of different clusters. In social networks communities may represent groups by interest, in citation networks they correspond to related papers, in the Web communities are formed by pages on related topics, etc. Being able to identify communities in a network could help us to exploit this network more effectively. In our example, clusters in *Yelp* hypergraph may help to find similar restaurants, discovering users with similar interests that is important for targeted advertisement.

The key ingredient for many clustering algorithms is modularity, which is at the same time a global criterion to define communities, a quality function of community detection algorithms, and a way to measure the presence of community structure in a network. Modularity was introduced by Newman and Girvan [27] and it is based on the comparison between the actual density of edges inside a community and the density one would expect to have if the vertices of the graph were attached at random, regardless of community structure. The modularity function was recently generalized to hypergraphs [24] but no fast, heuristic algorithms are developed yet for this hypergraph counterpart. Our goal is to propose a number of potential solutions in the forthcoming paper and in this paper we present applicability of this method that has been already implemented in the `SimpleHypergraphs.jl` library.

### 3 Use case—*Yelp* dataset

In this section, we present a practical application of the `SimpleHypergraphs.jl` library. We especially focus on and analyze *Yelp* dataset consisting of reviews of restaurants. A natural representation of such data is a hypergraph in which vertices are associated with restaurants and hyperedges are associated with reviewers who reviewed various restaurants. The topology of this hypergraph allows us to find clusters of restaurants that are commonly reviewed together. As hypergraph clustering is an example of an unsupervised learning technique, our goal is to learn if such clusters are related to some natural characteristics of underlying restaurants. Such analysis allows us to better understand which factors (ground-truth) influence the changes that two restaurants are reviewed together. To that end we propose a methodology to measure and then to compare the results of hypergraph clustering against various possible ground-truth variables (here the main challenge is to develop a measure comparable across different ground truths). Since the *Yelp* dataset is used only as an example, the proposed approach can be used to identify ground-truths in other datasets that are represented as a hypergraph. As side effect of this use case, we also show that the hypergraph based approach conveys more information about the ground-truth properties of a hypergraph than a standard graph analysis approach. In particular, we compare the results obtained for hypergraphs with the corresponding results for 2-section, and show that hypergraph clusters provide uniformly more information than their graph counterpart. Additionally, when analyzing the data we consider different sub-hypergraphs, namely, we examine hypergraphs containing only reports with a given number of stars, from 1 to 5. This approach sheds some light on how review linkages are formed; in particular, we test how the mechanism behind those linkages differs across different review classes.

An interesting property that is worth to investigate, typical to many such networks, is the community structure, that is, the division of networks into groups of vertices that are similar among themselves but dissimilar from the rest of the network. The capability to detect the partitioning of a network into communities can give important insights into the organization and behaviour of the system that the network models.

Data	Instances	Description
Business	192,609	Business data including location, attributes, and categories.
User	1,637,138	User data including the user’s friend mapping and all the metadata associated with the user.
Review	6,685,900	Full review text including the <code>user_id</code> that wrote the review and the <code>business_id</code> the review is written for.
Picture	200,000	Photo data including caption and classification (one of “food”, “drink”, “menu”, “inside” or “outside”).
Tip	1,223,094	Tips written by users on businesses. Tips are shorter than reviews and tend to convey quick suggestions.
Check-in	192,609	Aggregated check-ins over time for each business.

Table 1: *Yelp* entities contained in the dataset.

### 3.1 The Yelp Open Dataset

*Yelp* is an online platform where customers can share their experiences about local businesses by posting reviews, tips, photos, and videos. It allows businesses and customers to engage and transact [12]. Every year, the Yelp Inc. Company releases part of their data as an open dataset to grant the scientific community to conduct research and analysis on them. Some interesting articles that use the Yelp dataset for their analysis can be found in [22,23,25,26]. As a use case, we analyzed the 2019 Yelp Challenge dataset [13], containing information about businesses, reviews, and users. Table 1 describes all the accessible dataset *entities*. A more detailed description can be found on the official page [14].

Figure 3 (on the left) presents business categories distribution, where a category is a label describing the typology of the business such as *Bars* or *Shopping* along with the number of reviews associated with each category. It highlights the category distribution evaluated over all businesses. As clearly visible from the plot, the most common business typology is *Restaurant*. For this reason, we focused our analysis on this business subgroup. Figure 3 (on the right) shows the category distribution evaluated only within the *Restaurant* macro-category. Both Figures show top-20 most common categories.

### 3.2 The Yelp Hypergraph

We model *Yelp* dataset using a hypergraph  $H = (V, E)$ , where  $V$  represents businesses and  $E$  represents users of Yelp. In particular, each hyperedge representing user  $u$  contains businesses  $u$  has written at least one review for. Figure 2 shows an example hypergraph representing a Yelp data subset. As shown in the figure, the hypergraph  $H$  is defined by four businesses ( $V = \{b_1, b_2, b_3, b_4\}$ ) and three users ( $E = \{u_1, u_2, u_3\}$ ). For instance, hyperedge  $u_1$  connects businesses  $b_1, b_2$ , and  $b_4$ , as the corresponding user have written reviews for each of the listed business.

Since processing the entire Yelp dataset is a heavy computationally task, for our purpose we have decided to explore only a subset of it. We have modelled the Yelp hypergraph according to the following building strategies:



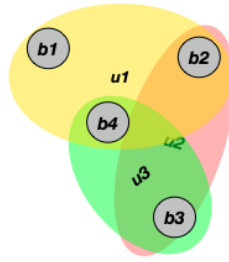


Fig. 2: Yelp Hypergraph defined by the users reviews.

1. **yelpdataset1** is a random selection of reviews of specific sizes. It is worth mentioning that such selection of reviews defines also the number of businesses involved. Indeed, our analysis are executed on connected hypergraphs that are obtained by removing isolated vertices and small components.
2. **yelpdataset2** is a subset of those businesses that belong to the category “restaurant” (note that some businesses have more than one category; in such cases we select one category from its categories set according to the frequencies (highest) in the whole dataset).

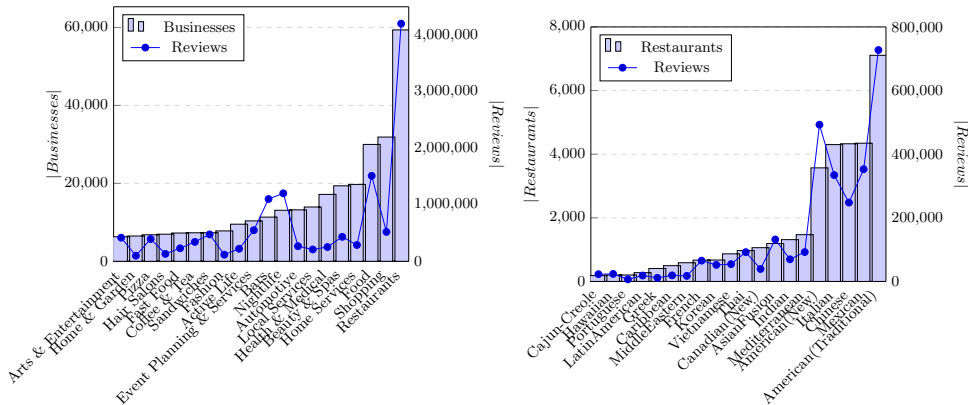


Fig. 3: Businesses (left) and Restaurants (right) distribution and number of reviews associated with each category.

### 3.3 Results

We are interested in the following two research questions. First of all, our goal was to investigate whether modelling the *Yelp* dataset with hypergraphs gives qualitatively more information than looking at the corresponding 2-section graph

representation. Then we compared the information provided by the three hypergraphs consisting of positive, neutral and negative reviews. In this case, the research question is: are the three hypergraphs similar or different? In order to answer the two questions, we set up two experiments explained below.

**Experiment I: “Forecasting stars”.** This experiment tries to forecast the number of stars of a given business  $v$ , based on the information available in the local neighbourhood of  $v$ . Two different strategies have been developed, one is based on the information provided by hypergraph  $H$  defined above, and one is based on the information provided by the weighted 2-section of the same hypergraph. Here, the weight of an edge  $(u, v)$  corresponds to the number of users that reviewed both  $u$  and  $v$ , that is, the number of hyperedges that contain both  $u$  and  $v$ .

For the first strategy (on hypergraph  $H$ ), for each business  $u$ , we first compute the average number of stars for all hyperedges containing  $u$ ; in each hyperedge  $e$ , the average is computed excluding  $u$ . This corresponds to the typical rating given by the user associated with  $e$ . Then, the forecast for the number of stars of  $u$  is obtained as the average over the values computed at the previous step. In other words, the forecast of the number of stars of  $u$  is the average over the averages in each hyperedge involving  $u$ . Formally,

$$s'_i(u) = \frac{1}{|E(u)|} \sum_{e \in E(u)} \left( \frac{1}{|e| - 1} \sum_{v \in e, v \neq u} s(v) \right),$$

where  $s(v)$  denotes the number of stars associated to  $v$ ,  $E(v)$  denotes the set of hyperedges that contains  $v$ , and  $s'_i(u)$  denotes the forecasted value for  $u$  for strategy  $i$ .

The second strategy exploits the weighted 2-section graph. In this case, the forecast of the number of stars of  $u$  is the weighted average over the neighborhood of  $u$ . Formally,

$$s'_2(u) = \frac{\sum_{e=(u,v) \in E} s(v)w(e)}{\sum_{e=(u,v) \in E} w(e)},$$

where  $w(e)$  denotes the weight of edge  $e$ .

In order to compare the two strategies, we computed their average error as follows:

$$err_i = \frac{\sum_{u \in V} |s(u) - s'_i(u)|}{|V|}.$$

We performed our experiment on several instances of **yelpdataset1**, varying the number of reviews used. The left side of Figure 4 depicts the obtained results for stars’ forecast experiment. The error value  $err_2$  using the weighted 2-section graph is always greater than the error value  $err_1$  obtained for the hypergraph representation.

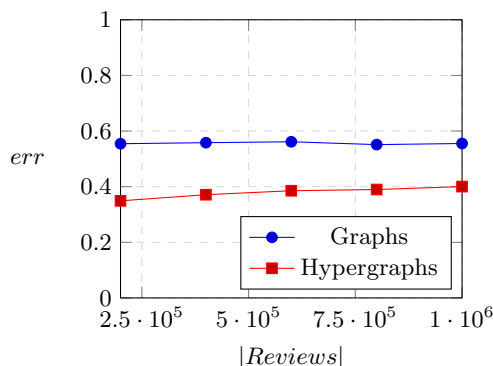


Fig. 4: Stars' forecast varying the dimension of the reviews set on **yelpdataset1**.

We also experimented with forecasting on **yelpdataset2**, obtaining similar results; the error for graphs is always close to 0.6 while the error for hypergraphs is always close to 0.5. Both the results are promising since the average number of stars obtained by businesses are around 0.5 and so it is important to be able to accurately predict low rated instances. Our experiment shows that the information provided by the hypergraph is more accurate than the information provided by the corresponding weighted 2-section.

**Experiment II: Positive, neutral, and negative reviews.** The second experiment examines the amount of information given by different kind of reviews, depending on the number of stars associated to them. We used **yelpdataset2** but due to the performance issues, we restricted the set of businesses to restaurants category, as described in Section 3.2. Five hypergraphs were built after partitioning the reviews into five categories: 1 stars, 2 stars,  $\dots$ , 5 stars.

In the dataset we have 342,044 1-star reviews, 281,307 2-star reviews, 402,053 3-star reviews, 791,068 4-star reviews and 1,188,558 5-star reviews. Hence, we decided to build five hypergraphs, one for each set of reviews. Henceforth, for  $i = 1, 2, \dots, 5$ , we will denote by  $H_i$ , the hypergraph generated using the set of reviews having  $i$  stars and by  $G_i$  the corresponding 2-section view graph.

Stars	$H_i ( V ;  E )$	$G_i ( V ;  E )$	$G_i$ Modularity	$G_i$ Triangles
1	(29479; 244671)	(29479; 240412)	0.6210	1,158,341
2	(28055; 173140)	(28055; 484527)	0.7173	6,491,497
3	(30369; 177792)	(30369; 2636712)	0.6616	289,584,451
4	(32987; 301578)	(32987; 4384044)	0.6857	404,709,664
5	(32558; 590320)	(32558; 2187473)	0.6657	104,128,714

Table 2: Graphs Statistics.

Stars	$H_i ( V ;  E )$	City	State	Alcohol	Noise Level	Take Out	Category
1	(29479; 244671)	0.8833	0.9562	0.8166	0.8104	0.8176	0.8163
2	(28055; 173140)	0.8582	0.9462	0.7744	0.7651	0.7731	0.7702
3	(30369; 177792)	0.8132	0.9226	0.7075	0.6940	0.6966	0.6965
4	(32987; 301578)	0.7812	0.9081	0.6573	0.6385	0.6419	0.6400
5	(32558; 590320)	0.8027	0.9145	0.6963	0.6797	0.6894	0.6841
<b>ALL</b>	(35856; 950488)	0.7500	0.8985	0.6162	0.5919	0.6013	0.5967

Table 3: Hypergraph modularities for various number of stars and various ground truth based partitioning conditioned on properties of restaurants.

First, we computed some statistics on the five hypergraphs and their corresponding 2-section views. The collected information can be found in Table 2. This preliminary analysis shows that the five hypergraphs/graphs are quite different. For instance, for the 2-section graphs, the number of edges, and the number of triangles exhibit a “bell-shaped” trend as a function of the number of stars. As a result, we shift our attention to their ability to detect the community structure, that is, the division of the vertex set into groups of restaurants that are similar among themselves but dissimilar from the rest of the network. In order to evaluate this feature, we decided to run some community detection algorithms on each graph/hypergraph. We then compared the obtained results with a ground truth restaurant partitioning, based on the “type of cuisine” provided by the system. This ground truth partitioning is composed of 55 categories of which the largest (American Traditional) comprises 7,107 restaurants.

The Table 3 contains modularity values for various partitionings of the hypergraph. In order to calculate modularities we used approach presented in [24] that we have implemented as the `modularity` function in the `SimpleHypergraphs.jl` library. One can see that the modularity is strongest when we uses city or state to partition the hypergraph. This means that people doing reviews usually use restaurants within the same city and if restaurants in different cities are reviewed by a single person they are usually in the same state. It can be noted that reviews with one star have the strongest modularity values across all partitionings. This probably means that there is a group of people who have a stronger tendency to submit negative scores on the base of some ground-truth property of a restaurant.

Several community detection algorithms have been proposed in the literature. A review of the various methods available can be found, for example, in [20,15]. For graphs, we decided to opt for a label propagation (LP) strategy proposed by Raghavan et al. [28]. This strategy can be summarized as follows: each node is initially given a unique label (initialization); at each iteration, each node is updated by choosing the label which is the most frequent among its neighbours (propagation rule)—if multiple choices are possible (as, for example, at the very beginning), one among the candidate labels is picked randomly. The algorithm terminates at the first iteration that leaves the label configuration unchanged

or after the predefined number of iterations (termination criteria). We exploited the LP implementation provided by the Julia LightGraphs library [8].

For hypergraphs, we implemented an ad-hoc label propagation strategy which generalizes the algorithm in [28] for hypergraphs. The proposed algorithm shares the initialization phase as well as the termination criteria with the standard label propagation algorithm. On the other hand the propagation rule is, in this case, composed of two phases: hyperedge labelling and vertex labelling. During the hyperedge labeling phase, labels of hyperedges are updated according to the most frequent label among the vertices that belong to the edge. Then, during the vertex labeling phase, label of each vertex is updated by choosing the label that is the most frequent among the hyperedges it belongs to.

Both algorithms have been executed setting the maximum number of iterations to 100. We compared the partitions obtained running the label propagation strategies described above with the ground truth partition in order to learn how much they are related. Several measures to evaluate the correlation between the two partitions have been borrowed from information theory. In particular, by considering a partition as a probability distribution, the *Normalized Mutual Information* (NMI) is often used to measure their correlation. Several variants of the NMI have been defined (see, for example, [30] for a detailed discussion). In this paper we use the *sum* variant which is defined as follows:

$$NMI(X, Y) = \frac{I(X, Y)}{H(X) + H(Y)}, \quad (1)$$

where  $I(X, Y)$  denotes the *Mutual Information* (that is, the shared information between the two distributions  $X$  and  $Y$ ) and  $H(X)$  denotes the Shannon Entropy (that is, the information contained in the distribution) of  $X$ . NMI enjoys several interesting properties: namely it is a *metric* and lies within a fixed range  $[0, 1]$ . Specifically it equals 1 if the partitions are identical whereas it has an expected value of 0 if the two partitions are independent.

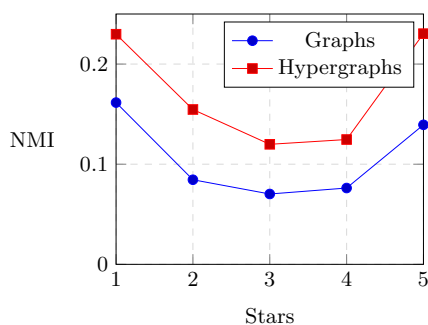


Fig. 5: Then NMI between the ground truth partition and the 10 partitions obtained running the label propagation algorithm on the five hypergraphs and on the corresponding 2-section views.

Results appear in Figure 5. Although the correlation in general is not very high (the best result is 0.23 for  $H_5$ ), the figure provides two interesting points. First, in all the five considered cases, the quality of partitioning provided by hypergraphs is always better than that provided by the corresponding 2-section view graph. Moreover, also in this case, the results appear in the form of an “inverted bell shape” (the best results in this case are given by the two external values). In a sense, very good as well as very bad reviews are much better able to identify restaurants genre.

## 4 Conclusion

In this work we have presented a novel library for the manipulation and analysis of hypergraph structures. Hypergraphs have been shown to be much better than standard graphs to model many natural phenomena, such as collaborative activities, which involves group based interactions.

The library, named `SimpleHypergraphs.jl`, provides Hypergraph views built exploiting the popular package `LightGraphs.jl` a Julia library for graphs manipulation. Several functionalities for the I/O, the manipulation and the transformation of hypergraphs have already been developed and are available on a public GitHub repository. In addition, the library enables the user defining meta information type as well as attaching meta-data values of arbitrary type to both vertices and hyperedges. This approach enables for an efficient analysis of structural properties of the network, combined to the possibility to perform semantic analysis based on the attached meta-data. The Yelp dataset case studies show that it scales well when analyzing thousands of nodes connected by millions of edges. We plan to expand the library by developing novel functionalities and a visualization engine which will enable the exploration of the hypergraph networks as well as of the enclosed meta-information. We have presented also a case study based on the Yelp dataset showing some of the functionalities available on `SimpleHypergraphs.jl` and, at the same time, that hypergraph networks convey much information with respect to their corresponding graph representation.

## References

1. gspbox, MatLab. <https://github.com/epfl-lts2/gspbox> (2019), [Online; 2019]
2. halp, Python. <https://github.com/Murali-group/halp> (2019), [Online; 2019]
3. Hypergraph, Chapel. <https://github.com/pnnl/chgl> (2019), [Online; 2019]
4. HyperGraphLib, C++ . <https://github.com/alex-87/HyperGraphLib> (2019), [Online; 2019]
5. HyperNetX, Python. <https://github.com/pnnl/HyperNetX> (2019), [Online; 2019]
6. HyperX, Scala. <https://github.com/jinhuang/hyperx> (2019), [Online; 2019]
7. iper, JavaScript. <https://github.com/fibo/iper> (2019), [Online; 2019]
8. LightGraphs.jl, Julia. <https://github.com/JuliaGraphs/LightGraphs.jl> (2019), [Online; 2019]
9. multihypergraph, Python. <https://github.com/vaibhavkarve/multihypergraph> (2019), [Online; 2019]
10. networkR, R. <https://github.com/O1sims/networkR> (2019), [Online; 2019]
11. pygraph, Python. <https://github.com/jciskey/pygraph> (2019), [Online; 2019]
12. yelp. <https://www.reuters.com/finance/stocks/company-profile/YELP.N> (2019), [Online; 2019]

13. yelp-dataset. <https://www.yelp.com/dataset/challenge> (2019), [Online; 2019]
14. yelp-dataset-docs. <https://www.yelp.com/dataset/documentation/main> (2019), [Online; 2019]
15. Antelmi, A., Cordasco, G., Spagnuolo, C., Vicidomini, L.: On evaluating graph partitioning algorithms for distributed agent based models on networks. In: Euro-Par 2015: Parallel Processing Workshops. pp. 367–378. Springer International Publishing, Cham (2015)
16. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. *SIAM review* **59**(1), 65–98 (2017)
17. Bretto, A.: *Hypergraph Theory: An Introduction*. Springer Publishing Company, Incorporated (2013)
18. Cordasco, G., Spagnuolo, C., Scarano, V.: Toward the new version of d-mason: Efficiency, effectiveness and correctness in parallel and distributed agent-based simulations. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1803–1812 (2016)
19. Cordasco, G., De Chiara, R., Raia, F., Scarano, V., Spagnuolo, C., Vicidomini, L.: Designing computational steering facilities for distributed agent based simulations. In: Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. pp. 385–390 (2013)
20. Danon, L., Díaz-guilera, A., Duch, J.: Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment* (2005)
21. Edelman, A.: Julia: A fresh approach to technical computing and data processing. Tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE CAMBRIDGE (2019)
22. Gulati, A., Eirinaki, M.: Influence propagation for social graph-based recommendations. In: 2018 IEEE International Conference on Big Data (Big Data). pp. 2180–2189 (2018)
23. Ji, Z., Pi, H., Wei, W., Xiong, B., Woźniak, M., Damasevicius, R.: Recommendation based on review texts and social communities: A hybrid model. *IEEE Access* **7**, 40416–40427 (2019)
24. Kaminski, B., Poulin, V., Pralat, P., Szufel, P., Theberge, F.: Clustering via hypergraph modularity. arXiv preprint arXiv:1810.04816 (2018)
25. Li, R., Jiang, J.Y., Ju, C.J.T., Wang, W.: Corals: Who are my potential new customers? tapping into the wisdom of customers’ decisions. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. pp. 69–77. WSDM ’19 (2019)
26. Lu, X., Qu, J., Jiang, Y., Zhao, Y.: Should i invest it?: Predicting future success of yelp restaurants. In: Proceedings of the Practice and Experience on Advanced Research Computing. pp. 64:1–64:6. PEARC ’18 (2018)
27. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. *Physical review E* **69**(2), 026113 (2004)
28. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. *Physical review. E, Statistical, nonlinear, and soft matter physics* **76** (2007)
29. Regier, J., Fischer, K., Pamnany, K., Noack, A., Revels, J., Lam, M., Howard, S., Giordano, R., Schlegel, D., McAuliffe, J., et al.: Cataloging the visible universe through bayesian inference in julia at petascale. *Journal of Parallel and Distributed Computing* (2019)
30. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.* **11**, 2837–2854 (2010)