

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Analyzing, exploring, and visualizing complex networks via hypergraphs using simplehypergraphs.Jl

This is the author's manuscript

Original Citation:

Availability:

This version is available <http://hdl.handle.net/2318/1940471> since 2023-12-28T16:51:08Z

Published version:

DOI:10.24166/im.01.2020

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

Analyzing, Exploring, and Visualizing Complex Networks via Hypergraphs using SimpleHypergraphs.jl*

Alessia Antelmi¹, Gennaro Cordasco², Bogumił Kamiński³, Paweł Prałat⁴,
Vittorio Scarano², Carmine Spagnuolo², and Przemysław Szufel³

¹ Dipartimento di Informatica, Università degli Studi di Salerno, Italy
aantelmi@unisa.it, vitsca@unisa.it, cspagnuolo@unisa.it

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

³ SGH Warsaw School of Economics, Poland bkamins@sgh.waw.pl,
pszufe@sgh.waw.pl

⁴ Department of Mathematics, Ryerson University, Toronto, ON, Canada
pralat@ryerson.ca

Abstract. Real-world complex networks are usually being modeled as graphs. The concept of graphs assumes that the relations within the network are binary (for instance, between pairs of nodes); however, this is not always true for many real-life scenarios, such as peer-to-peer communication schemes, paper co-authorship, or social network interactions. For such scenarios, it is often the case that the underlying network is better and more naturally modeled by hypergraphs. A hypergraph is a generalization of a graph in which a single (hyper)edge can connect any number of vertices. Hypergraphs allow modelers to have a complete representation of multi-relational (many-to-many) networks; hence, they are extremely suitable for analyzing and discovering more subtle dependencies in such data structures.

Working with hypergraphs requires new software libraries that make it possible to perform operations on them, from basic algorithms (such as searching or traversing the network) to computing significant hypergraph measures, to including more challenging algorithms (such as community detection). In this paper, we present a new software library, **SimpleHypergraphs.jl**, written in the Julia language and designed for high-performance computing on hypergraphs and propose two new algorithms for analyzing their properties: s -betweenness and modified label propagation. We also present various approaches for hypergraph visualization integrated into our tool. In order to demonstrate how to exploit the library in practice, we discuss two case studies based on the 2019 Yelp Challenge dataset and the collaboration network built upon the Game of Thrones TV series. The results are promising and they confirm the ability of hypergraphs to provide more insight than standard graph-based approaches.

Keywords: Hypergraphs · Analyzing hypergraphs · Exploring hypergraphs · Visualizing hypergraphs · Software library · Julia language

* The research is financed by NAWA — The Polish National Agency for Academic Exchange.

1 Introduction

Research on the analysis of networks has a long tradition, and have provided mathematics and computer scientists tools enabling the exploration, the study, and the comprehension of complex phenomena [63]. Since its birth in the Eighteenth century at the hands of the Swiss mathematician Leonhard Euler, *graph theory* — the branch of discrete mathematics dealing with the study of *networks* — has contributed to the resolution of many real-world problems [29,35]. In particular, over the last twenty years, the interests of research have focused on *complex networks*, namely networks whose structure is irregular, complex and dynamically evolving in time [26,42,67]. Complex networks naturally model many real-world scenarios, such as social interactions [31,55], biological [40,41,62] and economical [37,70] systems, Internet [36], and the World Wide Web [63], just to name a few examples. Traditionally, these networks are described using graphs, where nodes represent elements of the network, and edges represent relationships between some pairs of elements. However, in many practical applications, relationships between the elements of a network may not be dyadic but may involve more than two nodes. Examples of such scenarios include membership in groups on social platforms, co-authorships of scientific publications, or several parties participating in a crypto-currency transaction [30]. In such cases, nodes may be linked together based either on explicit information (e.g., inclusion in groups), or implicit information (e.g., whether online social network users share the same hashtag in a media post or review the same restaurant). Obviously, the resulting complexity of these networks is tremendous, as the relationships between vertices can involve an arbitrary number of elements. A challenging task arising in this context is providing scientists a tool to effortlessly model such scenarios. Here, *hypergraphs* come into play. A hypergraph is a generalization of a graph where the vertices are related not only by pair-wise connections (edges), but they can include an arbitrary number of nodes (hyperedges). In other words, hypergraphs can naturally model all the above scenarios.

The powerful expressiveness of hypergraphs has, however, few drawbacks: dealing with the complexity of such data structures and the lack of appropriate tools and algorithms for their study. For this reason, hypergraphs have been little used in literature in favor of their graph-counterpart. A traditional approach in network science to handle such scenarios is using the *two-section* graph representation of a hypergraph, which vertices are the vertices of the hypergraph and where two distinct vertices form an edge if and only if they are in the same hyperedge [57,69]. In other words, a complete graph (or a clique) of order k replaces each hyperedge of cardinality k . Another way to deal with a hypergraph is analyzing its *line-graph*, defined as the graph where the node-set is the set of the hyperedges and two nodes are connected by a link when the corresponding hyperedges share at least a node [48]. A third approach consists in using a bipartite graph, where the vertices and hyperedges of a hypergraph represent the two disjoint vertex sets. Recommender systems heavily manipulate such representation [23,65,53]. However, all these techniques share a weakness: as they do not exploit hypergraphs, their implementation requires a different and less

natural data structure to handle the same set of information. Additionally, both the two-section and line-graph transformations lose information encoded in a hypergraph that cannot be transferred to the corresponding graph. For a more clarifying example, we can consider the network built upon e-mail exchanges between some users. In this context, the object *e-mail* can be modeled as a relation involving a group of users. Thus, in this case, nodes of the network represent the persons, while the edges of the network incorporate a sub-set of them – i.e., all e-mail receivers. It is worth noting that if we represent this scenario with a graph, we lose the information about which users are receivers of the same e-mails. This approach, combined with grouping messages having the same title, can be used for anomaly and spam detection in electronic communication [64].

To illuminate this uncharted area, we delved into the study of hypergraphs, discussing how and to what extent this mathematical structure can model, analyze, and visualize complex networks characterized by many-to-many relations. In this paper, we propose `SimpleHypergraphs.jl`, a complete software tool written in the Julia language. Here, our aim is two-fold: i) improving the usability and efficiency of software libraries for hypergraphs manipulation by exploiting the efficiency provided by Julia and ii) developing a holistic set of functionalities ensuring the broad applicability of our library. The contributions of our work can be summarized as follows:

- We propose a software library for the analysis, exploration, and visualization of hypergraphs, exploiting the Julia language to ensure both efficiency and expressiveness. Julia is a new programming language developed at MIT [25], with a syntax similar to popular and easy-to-use scientific computing languages such as Python or R. This means that experience in those languages can be directly applied in Julia by computational scientists [33,60]. Although it keeps a math-oriented syntax, Julia compiles the code to a binary form. As a result, the observed performance of Julia programs is very similar to C++, but with around 4 times fewer lines of code. `SimpleHypergraphs.jl` is available on a GitHub public repository⁵, where it is possible to find the library documentation⁶, and several tutorials in the form of Jupyter Notebooks⁷. In this article, we describe the library functionalities available in the current version 0.1.7 of `SimpleHypergraphs.jl`. The library provides a set of analytical functionalities (modularity, connected components, random-walk), as well as a serialization mechanism to store hypergraph metadata. It also includes a visualization component that allows users to explore the network through two different hypergraph visualizations;
- We discuss two use cases where we use hypergraphs to analyze complex networks, and we compare their performance with the corresponding two-section graph. The first case study deals with business reviews from the platform *Yelp.com*, while the second application investigates the relationships between characters of the *Game-of-Thrones* TV Series. In order to

⁵ <https://github.com/pszufe/SimpleHypergraphs.jl>.

⁶ <https://pszufe.github.io/SimpleHypergraphs.jl/latest/reference/>.

⁷ <https://github.com/pszufe/SimpleHypergraphs.jl/tree/master/examples>.

perform these analyses we propose two new algorithms for analyzing the properties of hypergraphs: s -betweenness and modified label propagation.

The paper is structured as follows. In Section 2, we start by motivating the introduction of a novel software library to analyze and explore hypergraphs, and we provide a review of existing available (currently maintained) software tools. Section 3 defines the notation used in this work and introduces our Julia-based library for hypergraphs. Next, in Section 4, we present two use cases with the aim to show concrete applications and discuss the s -betweenness and modified label propagation algorithms we propose. Finally, in Section 5, we discuss some conclusions and future directions.

2 Motivation

Hypergraphs are a natural generalization of graphs, where a single (hyper)edge can connect more than two vertices. In several real-world applications, such representation is not only more general but also more natural than a standard graph representation, where a binary representation of relationships is sometimes not sufficient to correctly capture subtle interactions. Typical applications of hypergraphs include modeling paper co-authorship networks (different authors contribute to the same paper [39]), online reviews (the same good purchased by several users [72]), social network activities (the same post commented by multiple users, links in social networks [46]), disease contingency plans (groups of people locating in the same place [27]), bio-engineering systems (modeling cellular networks [45]). Even though hypergraphs are natural representations of many real-world systems, there currently are very few software frameworks suitable for modeling and mining these structures. In this section, we give a brief state-of-art overview of several software libraries, focusing on the availability of their code and their capability to model and analyze hypergraphs.

- **Chapel HyperGraph Library (CHGL)** [4] has been developed by the Pacific Northwest National Laboratory since 2018, and released under the MIT license. CHGL is a library for the emerging parallel language Chapel. It provides the `AdjListHyperGraph` module that allows storing hypergraphs on shared and distributed memory. The library is not well documented and does not support any easy mechanism for the two-section and bipartite analyses. Nonetheless, it is worth mentioning for its compatibility for parallel and distributed computing.
- **HyperX** [7] is a scalable framework for processing hypergraphs and learning algorithms 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 the Scala language. An interesting feature of this library is that it provides native support for the hypergraph elaboration. Directly processing hypergraph data, HyperX obtains significant speedup with respect using the bipartite or the two-section representation of a hypergraph and then exploiting the GraphX APIs.

- **Pygraph** [13] is a pure Python library for graph manipulation, released under the MIT license. It supports a hypergraph representation by exposing the class `hypergraph`, even though it does not provide any specific optimization or functionality for hypergraphs.
- **Multihypergraph** [10] is a Python package for graphs, released under the GPL license. The library emphasizes the mathematical understanding of graphs rather than the algorithmic efficiency, and it provides support for hyper-edges, multi-edges, and looped-edges. This library implements only the graph memory model definition and isomorphism functionalities, without defining any other functionality and algorithm for graphs and hypergraphs.
- **HyperNetX** [6] is a Python preliminary library released in 2018 under the Battelle Memorial Institute license. This library generalizes traditional graph metrics (such as vertex and edge degrees, diameter, distances) to hypergraphs, and it provides proper documentation and tutorials. HyperNetX supports the bipartite representation of a hypergraph, along with the possibility to load hypergraphs from their bipartite view. Furthermore, it exports some visualization functionalities for hypergraphs based on Euler-diagrams.
- **Halp** [3] is a Python software package providing both a directed and an undirected hypergraph implementation, as well as several major and classical algorithms. The library is developed by Murali’s Research Group at Virginia Tech, and it is released under the GPL license. The library provides several statistics on hypergraphs and model transformations in graphs, supported by the **NetworkX** Python library. Several algorithms for hypergraphs, such as k -shortest-hyperpaths and random walk, are also implemented.
- **HyperGraphLib** [5] is a C++ implementation of hypergraphs that exploits the Boost Library, also defining the library license. This library provides basic functionalities for hypergraphs and implements some simple metrics. Moreover, it provides isomorphism functionalities and path-finding algorithms. However, it does not implement any hypergraph representation into a graph (such as a bipartite or a 2-section graph) nor software integration with other graph libraries.
- **Iper** [8] is a JavaScript library for hypergraphs, released under the MIT license. The library defines a hypergraph and allows the user to define meta-information for vertices. However, it does not include any hypergraph transformation and integration with other graph libraries for classical statistics and algorithms.
- **NetworkR** [12] is an R package with a set of functions for analyzing social and economic networks, including hypergraphs. It incorporates analyses such as degree distribution, 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** [2] 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 weighted hyperedges, and vertices

with coordinates in the space. The hypergraph manipulation is obtained by representing the model as a graph. For this reason, although all graph functionalities are available, the library does not provide any specific solutions or optimization for hypergraphs.

Overall, all the considered libraries are a compromise between efficiency, which characterizes low-level languages such as C/C++, and the easy-of-use and expressiveness, distinguishing interpreted and scripting languages such as Python and R.

3 SimpleHypergraphs.jl

In this Section, we present the `SimpleHypergraphs.jl` library, which provides flexible functionalities for the analysis and modeling of hypergraphs. Being implemented in the Julia programming language, and released under the terms of the Open Source MIT License, it is currently part of the official Julia package repository. This section is organized as follows. We introduce the adopted formal notation and definitions, and we then move to describe the library design and memory model. Finally, we discuss its functionalities about hypergraph manipulation, analysis, and visualization.

3.1 Definitions and notation

Formally, a hypergraph [30] is an ordered pair $H = (V, E)$ where V is the set of nodes (often also called vertices) and E is the set of edges. Each edge is a non-empty subset of vertices; i.e., $E \subseteq 2^V \setminus \{\emptyset\}$, where 2^V is the power set of V . We use $n = |V|$ and $m = |E|$ to indicate the size of the vertex set and the edge set, respectively. A graph can be seen as a hypergraph where each hyperedge is a two element subset of V . In other words, a graph $G = (V, E)$ is a hypergraph, if $E \subseteq \binom{V}{2} \subseteq 2^V \setminus \{\emptyset\}$.

3.2 Library design

`SimpleHypergraphs.jl` provides APIs representing 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. In the APIs, 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 the vertex i within the hyperedge j . The library also provides several constructors for defining meta-information type and enables to attach meta-data values of arbitrary type to both vertices and hyperedges. To ensure flexible co-operability, `SimpleHypergraphs.jl` provides two-fold integration both with Julia standard matrix and `LightGraphs.jl` APIs.

- i. *Julia’s matrix APIs.* We achieved the Julia `Array` APIs integration by making the `Hypergraph` struct a subclass of `AbstractMatrix`, and providing a set of integration methods for manipulating matrices (i.e. querying the matrix size, fetching/updating elements). Internally, a hypergraph is stored as a sparse array. Hypergraph data are stored in a redundant format, using two separate hashmap structures for rows and columns to ensure good algorithmic performances. This design choice simultaneously provides high-grade performance across rows and columns. Furthermore, it avoids the circumstance where all data need to be rewritten when the adjacency matrix is updated (typical disadvantage of a compressed sparse row matrix). As a subclass of `AbstractMatrix`, the hypergraph adjacency matrix can be manipulated just like any other matrix in Julia. As a result, from the user’s point of view, a hypergraph $H = (V, E)$ can be seen as a $n \times k$ matrix representation, where n is the number of vertices and k is the number of hyperedges. Vertices and hyperedges are uniquely identified by progressive integers, corresponding to rows $(1, \dots, n)$ and columns $(1, \dots, k)$, respectively. Moreover, the library supports generic type metadata for both vertices and hyperedges.
- ii. *LightGraphs.jl.* We obtained the integration with this Julia library to manipulate graphs by creating hypergraph “view” classes providing a representation of a hypergraph as either a bipartite or a two-section graph. Those representations actually do *not* copy the data, but provide a view (`TwoSectionView` and `BipartiteView`) that allow to access the hypergraph data in a read-only mode. As we developed a full set of integration methods for the `LightGraphs.jl` library, the user can directly analyze a hypergraph structure with all functionality provided by `LightGraphs.jl`.

3.3 Memory model and functionalities

The latest release 0.1.7 of `SimpleHypergraphs.jl` provides a range of new functionalities and methods to build and explore hypergraphs. The following sections introduce the hypergraph representation, several basics operations, and transformations, the serialization mechanisms (raw and JSON formats), a set of analytical algorithms (Section 3.4), and two visualization strategies (Section 3.5).

Hypergraph constructors. The Julia hypergraph object is defined as:

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

where `T`, a subtype of `Real`, represents the type of the weights stored in the structure; `V` and `E` are the types of the meta-data values stored in the vertices and edges of the hypergraph, respectively; and `D`, a subtype of `AbstractDict{Int, T}`, is the type of the underlying dictionary used for storing the weight values. Note that when calling the constructor, the parameters `{T, V, E, D}` can be omitted (starting from the rightmost). The default value for the dictionary type `D` is a standard Julia dictionary `Dict{Int, T}` (where `T` is the type of the weights).

The default value for vertex and edge metadata types, `V` and `E`, is `Nothing` — i.e., by default, no metadata is stored. A new empty hypergraph can be built specifying the number of vertices (rows) and hyperedges (columns). Optionally, a hypergraph can be either materialized starting from a given matrix or a `LightGraphs.jl` graph object.

Querying and manipulating 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.
- `remove_vertex!` removes a vertex from a given hypergraph H .
- `set_vertex_meta!` sets a new meta-value *new_value* for vertex *id* in H .
- `get_vertex_meta` returns the meta-value stored at vertex *id* in H .
- `get_vertices` returns the vertices for a given hyperedge he_{id} in H .
- `nhv` returns the number of vertices in the hypergraph H .

We implemented analogous functionalities for the hyperedges.

Hypergraph transformations. The library provides two hypergraph transformations into the corresponding graph representations.

1. **BipartiteView.** It is the bipartite representation of a hypergraph H . As described in Bretto [30], this representation is the incidence graph of the 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) illustrates a simple example of bipartite view.
2. **TwoSectionView.** It is a two-section representation of a hypergraph H . As described in Bretto [30], this representation of a hypergraph $H = (V, E)$, denoted with $[H]_2$, 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) details a trivial example of two-section view.

Both `Views` are instances of `AbstractGraph`, the graph object defined by the `LightGraphs.jl` library. When the view is materialized, according to the package specifics, the generated graph does not include any meta information.

Hypergraph serialization. The library currently offers two mechanisms to load and save a hypergraph from or to a stream. Given a hypergraph H , it may be stored using either a *plain text* or *JSON* formats.

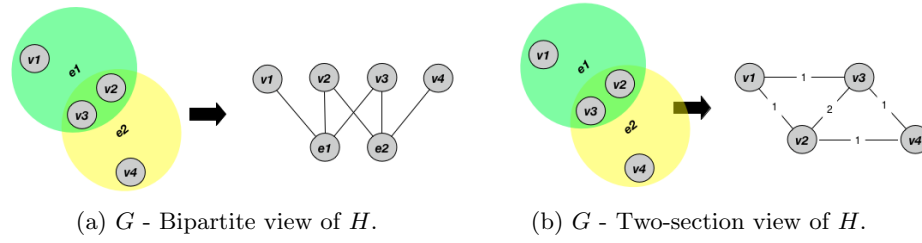


Fig. 1: Hypergraph transformations into the graph counterpart.

Plain text format, denoted by the `HGF_Format` storage type. The first line consists of two integers n and k , representing the number of vertices and the number of edges of H , respectively. The following k rows (lines in a text file) describe the actual structure of H : each line represents a hyperedge and contains a list of all vertex-weight pairs within that hyperedge.

JSON format, denoted by the `JSON_Format` storage type. The internal hypergraph structure is represented with a dictionary that is serialized into a plain JSON object. Each dictionary key represents a hypergraph field, while each dictionary value stores the corresponding hypergraph field value. Additionally, the matrix view of H is also stored. We used the Julia package `JSON3.jl` to handle the interaction between JSON and Julia types.

3.4 Analytical functionalities

Hypergraph modularity. Community detection is one of the most frequent tasks as it helps to find hidden interaction patterns in relational data. Newman and Girvan firstly proposed a hierarchical method to detect communities in complex systems introducing the concept of modularity [54]. The modularity value 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. Higher modularity values signify denser connections between the nodes within clusters but more sparse connections between nodes in different clusters. In `SimpleHypergraphs.jl`, we implemented a generalization of the modularity notion, recently proposed for hypergraphs [22]. We further provided an algorithm to calculate the modularity of a given vertex partition. This functionality is achieved via the `modularity` function which takes in input a hypergraph and its proposed partition given as a `Vector{Set{Int}}` object.

Connected components explorations. The function `get_connected_components` takes a hypergraph H as input and returns a vector of vectors. Each vector represents a set of vertices that are a connected component in H . We define two vertices a and b of a hypergraph H to be contained in a connected component if and only if a and b are connected by some sequences of hyperedges.

Random walk. Defining a random walk on a hypergraph is more complicated than defining the same notion on a graph, and there are a few natural ways it can be defined. One approach can be the following. Suppose the walk starts from some vertex i . Then it can randomly select a hyperedge i belongs to, and next choose a target vertex within that hyperedge, again at random. `SimpleHypergraphs.jl` provides the function `random_walk` that takes a hypergraph and a starting vertex id as input and returns a destination vertex id in one step of the walk. This design choice guarantees full flexibility in defining random walks on hypergraphs. The function also accepts two optional keyword arguments, both functions: `heselect` and `vselect`. The first function specifies the rule by which hyperedge is selected for a given starting vertex. The second parameter selects the destination vertex from the selected hyperedge. By default, `heselect` chooses a hyperedge containing the source vertex uniformly at random. Similarly, `vselect` selects a vertex from a given hyperedge uniformly at random.

3.5 Hypergraph visualization

`SimpleHypergraphs.jl` currently offers the possibility to draw a hypergraph by exploiting two kinds of visualization, through the function `draw`. The available plotting methods are either based on an interactive JavaScript (JS) or a static Python-based solution. Figure 2 illustrates the same hypergraph drawn using the two different strategies. A more detailed description follows.

A JS-based visualization. When dealing with complex objects that need to be visualized, it is of fundamental importance to have the possibility to easily catch the main information and, at the same time, to be able to retrieve more detail on demand [68]. For this reason, we decided to integrate a dynamic and interactive visualization within `SimpleHypergraphs.jl`. This visualization is a wrapper around an external JS package that exploits `D3.js`, a JS library for manipulating documents based on data, which combines powerful visualization components and a data-driven approach to DOM manipulation. This architectural stack provides the user with a way to generate a dynamic visualization embeddable into a web-based environment, such as a Jupyter Notebook. This method represents each hyperedge he of an hypergraph H as a new fictitious vertex fv to which each vertex $v \in he$ is connected (see Figure 2a). The appearance of vertices and hyperedges, whether displaying vertex weights and vertex and hyperedge metadata and labels are customizable.

A Python-based visualization. This visualization is a wrapper around the drawing functionalities offered by the Python library `HyperNetX` [6], built upon the Python package `matplotlib`. `HyperNetX` renders a Euler diagram of the hypergraph where vertices are black dots and hyperedges are convex shapes containing the vertices belonging to the edge set (see Figure 2b). As the authors note, it is not always possible to render the *correct* Euler diagram for an arbitrary hypergraph. For this reason, this technique may lead to cases where a hyperedge

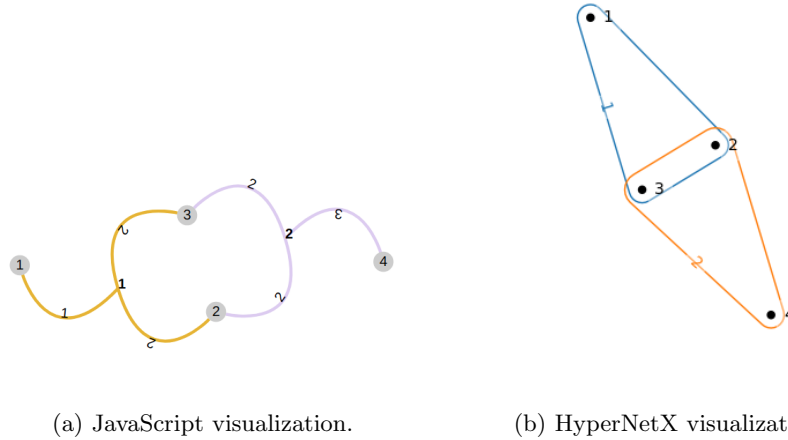


Fig. 2: SimpleHypergraphs.jl visualization methods.

incorrectly contains a vertex not belonging to its set. This library allows the user to manipulate the appearance of the resulting plot by letting the user defining the desired label, node, and edge options. SimpleHypergraphs.jl fully integrates the visualization potentiality of HyperNetX.

4 Use cases

In this Section, we discuss two use cases where we exploit hypergraphs to analyze complex data structures. The first case study deals with customers reviewing businesses on the social platform *Yelp.com*, while the second application investigates the relationships between characters of the *Game of Thrones* TV Series.

4.1 Exploring and analyzing user reviews: *Yelp.com*

In this first use case, we present a practical application of SimpleHypergraphs.jl applied to the analysis of business reviews from the online platform *Yelp.com* [14]. A hypergraph is an accurate representation of such data, where vertices symbolize *Yelp* businesses and hyperedges costumers who reviewed multiple businesses. An attractive property that is worth investigating in this context is the community structure [54], i.e., the division of the network 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 valuable insights into the organization and behavior of the system that the network models [34,35]. In this particular case, the topology of the so-built hypergraph suggests clusters of businesses that users commonly review together. As hypergraph clustering [74] is an example of an unsupervised learning technique, our

goal is to learn if such clusters are related to some natural characteristics of the underlying businesses. Such analysis allows us to understand which factors (ground-truth) influence the chance that a given user reviews any two businesses. To that end, we propose a methodology to measure and then to compare the results of hypergraph clustering against various possible ground-truth variables. In this context, the main challenge is to develop a measure comparable across different ground truths. Since the *Yelp* dataset serves only as an example, the proposed approach can be used to identify ground-truths in other datasets representable as a hypergraph. As a side effect of this use case, we point out that the hypergraph based approach conveys more information about the ground-truth properties of a network than a standard graph analysis. In particular, we compare the results obtained for hypergraphs with the results achieved for the corresponding two-section graphs and show that hypergraph clusters provide uniformly more information than their graph counterpart. In more detail, we analyzed five different sub-hypergraphs, each one containing only reviews with the same number of stars, from 1 to 5. This approach shed some light on how to review linkages form; in particular, we studied how the mechanism behind those linkages differs across different review classes.

To summarize, we were interested in the following two research questions.

- i) Does modeling the *Yelp* dataset with hypergraphs give more qualitative information than looking at the corresponding two-section graph representation?
- ii) Given the three hypergraphs consisting of positive, neutral, and negative reviews, are these similar and to what extent? To answer these two questions, we set up the two experiments explained below.

The *Yelp.com* dataset. *Yelp* is an online platform where customers can share their experiences with local businesses by posting reviews, tips, photos, and videos. It allows businesses and customers to engage and transact [14]. 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 [38,43,47,50]. As a use case, we analyzed the 2019 *Yelp* Challenge dataset [15], 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 [16]. 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 the top-20 most common categories. It is worth noting that as each *Yelp* restaurant may offer several types of cuisine (e.g., Indian, Chinese, Asian fusion), we labeled each business with a single *food category*, assigning to them the most frequent tag in the database. Namely, if a

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.

restaurant R had two genres A and B , but A was overall more frequent in the dataset, we labeled R with A .

Modeling *Yelp.com* using hypergraphs. We modeled the *Yelp* dataset using a hypergraph $H = (V, E)$, where V represents *Yelp* businesses, and E represents *Yelp* users. In more detail, each hyperedge representing a user u contains all businesses u has written at least one review for. Figure 4 shows a simple example hypergraph, defined by four businesses ($V = \{b_1, b_2, b_3, b_4\}$) and three users ($E = \{u_1, u_2, u_3\}$). Here, the hyperedge u_1 connects the businesses b_1, b_2 , and b_4 , as the corresponding user has written at least one review for each of the listed business.

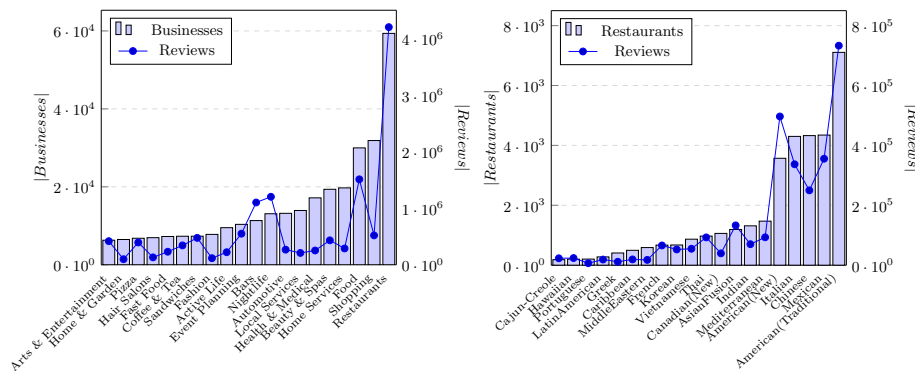


Fig. 3: Businesses (left) and Restaurants (right) distribution together with the number of reviews associated with each category.

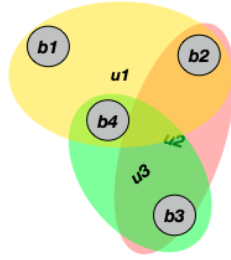


Fig. 4: The Yelp hypergraph defined by user reviews.

To accomplish our analysis, we explored only a subset of the Yelp data set, given the massive amount of available data (around 9 GB). We modeled the Yelp hypergraph according to the two following strategies.

1. **yelp-dataset1.** We collected 1 million of randomly chosen reviews, from which we selected the businesses and the users to build the hypergraph. It is worth mentioning that such selection defined the total number of businesses and users involved, i.e., the size of the hypergraph itself. We run our analysis on the largest connected component of the so-built hypergraph, removing isolated vertices and small components. More detailed information about the dimension of the network can be found in the following section.
2. **yelp-dataset2.** To generate this data set, we focused our attention only on the businesses belonging to the category “restaurant”. We attached to each restaurant the label (selected from its categories set) representing the type of cuisine it offers according to the methodology described above.

Forecasting the number of stars for a new business. This experiment focuses on the forecasting of the number of stars of a given business v , based on the information available in the local neighborhood of v . We developed two different strategies. We based one approach on the information provided by the hypergraph H defined in the previous section, the other on the information provided by the corresponding weighted two-section graph. Here, the weight of an edge (u, v) corresponds to the number of users that reviewed both u and v ; namely, the number of hyperedges containing both u and v .

In the *hypergraph-based* strategy, for each business u , we first computed the average number of stars for all hyperedges containing u (where, in each hyperedge e , the average was computed excluding u). This estimated value corresponded to the average rating given by the user associated with e . Then, we obtained the prediction about the number of stars of u as the average over the values computed in the previous step. In other words, the forecast of the number of stars of a given

user 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 *graph-based* strategy exploits the weighted two-section graph of H . In this case, the forecast of the number of stars of a user 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 .

To compare these two strategies, we computed the average error as follows:

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

We performed this experiment on several instances of the *yelp-dataset1*, varying the number of considered reviews. In particular, we selected five subsets of increasing size equal to 250k, 500k, 750k, and 1 million. For each subset, we computed the corresponding hypergraph. All hypergraphs resulted in having 209,393 nodes, while the number of hyperedges ranged from 175,022 to 432,381.

Figure 5 depicts the results for this experiment. As clearly visible from the plot, the error value err_2 using the weighted two-section graph is always greater than the error value err_1 obtained exploiting the hypergraph representation. We also run the stars forecasting experiment on the *yelp-dataset2*, which generated an hypergraph with 35,466 nodes and 1,133,890 hyperedges. Also in this case, the error for the two-section graph and the hypergraph was always close to 0.6 and 0.5, respectively. Interestingly, both experiment instances suggest that the information provided by a hypergraph model is more accurate than the information provided by the corresponding weighted two-section graph.

How much the rating of a review burden on a business? This second experiment examines the amount of information conveyed by different kinds of reviews, according to the number of stars associated with them. We used the **yelp-dataset2**; due to performance issues, we restricted the set of businesses only to the restaurant category, as described in Section 4.1. We ended up with 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. We built five hypergraphs, one for each set of reviews. Henceforth, we will denote with H_i , for $i = 1, 2, \dots, 5$, the

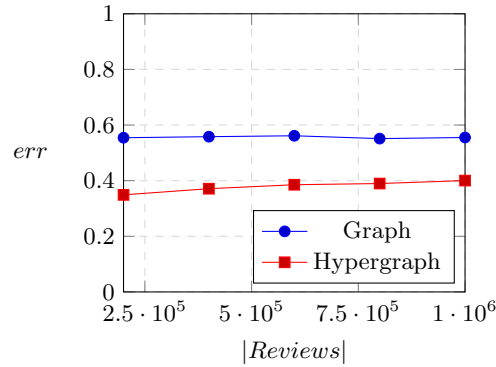


Fig. 5: Average error of the stars’ forecast experiment performed on yelp-dataset1, varying the number of reviews used to build the hypergraph and the corresponding two-section graph.

hypergraph generated using the set of reviews having i stars and by G_i the corresponding two-section graph. We first computed some statistics on these five hypergraphs and their corresponding two-section graphs. The collected information can be found in Table 2. This preliminary analysis highlighted that the so-built hypergraphs were quite different from their graph counterpart. Interestingly, what stands out in the columns corresponding to the two-section graphs is that the number of both edges and triangles exhibits a “bell-shaped” trend as a function of the number of stars.

In this second experiment, we focused our attention on understanding which factors influence the chance that a given user reviews any two businesses. To this aim, we tested the ability of these two different models to detect the community structure of the network. In other words, we intended to find the division of the vertices set into groups of restaurants that were similar among themselves but dissimilar from the rest of the network, and getting insights about the nature of their similarity. To have a first insight about the nature of the so-built network,

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: Some statistics on the yelp-dataset2 dataset.

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
ALL	(35856; 950488)	0.7500	0.8985	0.6162	0.5919	0.6013	0.5967

Table 3: Hypergraphs size for each H_i , $i = 1, 2, \dots, 5$. The modularity values have been computed on 6 different manually-evaluated ground-truth partitions, conditioned on some properties of the Yelp restaurants. Experiment have been run on yelp-dataset2.

we manually partitioned the five hypergraphs H_i , for $i = 1, 2, \dots, 5$, according to several restaurant properties available in the Yelp dataset. In particular, we considered the following attributes: the location of the restaurant (*city*, and *state*), whether it sells *alcohol*, its *noise level*, whether it offers a *take away* service, and its food *category*. Table 3 contains the modularity values for the 6 different partitions evaluated. To calculate the modularity, we used the approach presented in [44] and implemented in SimpleHypergraphs.jl (see Section 3.4). As shown in the table, the modularity is strongest when we used geographical information, as cities or states, to partition the hypergraph. These higher values mean that i) people doing reviews usually visit restaurants within the same city and that ii) if restaurants in different cities are reviewed by a single person, they are usually in the same state. It is worth noting that 1-star reviews have the strongest modularity values across all partitions. This result suggests 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.

Based on these outcomes, we then investigated to what extent the partition obtained from a community detection algorithm was able to mimic the communities output of a ground-truth partitioning. We compared the communities obtained on both models (hypergraph and two-section graph) against the given ground-truth partition to catch the best model in capturing that specific feature of the underlying network. Specifically, we focused on the communities evaluated on the “type of cuisine” (food *category*) of each restaurant. The ground-truth partition was made up of 55 categories, of which the largest (American Traditional) comprised 7,107 restaurants.

Several community detection algorithms have been proposed in the literature. A review of the various methods available can be found, for example, in [32,20]. We opted for the label propagation (LP) strategy proposed by Raghavan et al. [59] to find communities in a graph. It can be summarized as follows. Initially, each node has a unique label (initialization phase). At each iteration step, each node’s label is updated by choosing the most frequent label in its neighbors

(propagation rule); ties are broken with a random choice. The algorithm terminates either if it does not modify any label in two consecutive iterations, or it hits the predefined number of iterations (termination criteria). We exploited the LP implementation provided by the Julia `LightGraphs` library [9]. For hypergraphs, we propose to define a new LP strategy which generalizes the algorithm in [59]. The proposed algorithm shares both the initialization phase as well as the termination criteria with the standard graph label propagation algorithm. We modified the propagation rule, splitting it into two phases: hyperedge labeling and vertex labeling. During the hyperedge labeling phase, the labels of the hyperedges are updated according to the most frequent label among the vertices contained in that hyperedge. Similarly, during the vertex labeling phase, the label of each vertex is updated by choosing the most frequent label among the hyperedges it belongs to. Both algorithms have been executed setting the maximum number of iterations to 100.

To evaluate the correlation between two partitions, several measures have been borrowed from information theory, for instance, the *Normalized Mutual Information* (NMI) coefficient, which considers each partition as a probability distribution. Several variants of the NMI have been introduced (see, for example, [71] 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* (i.e., the shared information between the two distributions X and Y) and $H(X)$ denotes the Shannon Entropy (i.e., the information contained in the distribution) of X . The NMI coefficient holds several interesting properties. One of the most useful is that it is a *metric*, and it 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. Results appear in Figure 6. 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 five cases, the quality of partitioning provided by hypergraphs is always better than that provided by the corresponding two-section graphs. Moreover, also in this second experiment, 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.

Performance discussion. We performed our experiments on a Linux Ubuntu 18.04 machine equipped with an Intel i7 processor endowed of 8 cores, 16 GB of memory, and 256 GB SDD disk. We implemented the experiments using the 1.3.1 Julia language version. In the following, we present the average performance, in seconds, obtained executing 10 runs for each experiment. With this configuration, Julia required about 117.68 seconds to load the Yelp data (about 9GB) in memory.

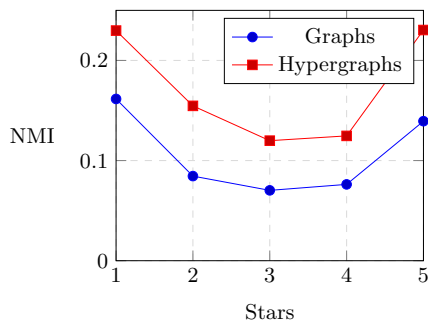


Fig. 6: NMI values between the “type of cuisine” ground-truth partition and the 10 partitions obtained running the label propagation algorithm on the five hypergraphs and on the corresponding 2-section graphs. Experiment have been run on *yelp-dataset2*.

The first case study concerned the execution of the forecasting algorithm on each hypergraph and the corresponding two-section graph. The completion time for the biggest experiment was about 42 seconds, while the smallest one required 0.92 seconds. In the second use case, the average completion time for all the experiments was around 20 seconds. Specifically, for all subsets of *yelp-dataset2* (see Section 4), we computed the network communities - exploiting the LP algorithm - on both the hypergraphs and the corresponding two-section graphs. Moreover, we compared the goodness of the computed sets with a ground-truth partition evaluating the NMI coefficient.

4.2 Mining and modeling social relationships: *Game of Thrones*

Another interesting line of inquiry is grasping the intricacies of a literary work or a movie to get insights into their narrative structure. Various works focused on finding common patterns across several plot lines [28], making sense of intricate character relationships [24], or just having fun trying to predict how the plot itself will evolve [66]. Usually, the character interaction network is modeled with a graph, where a vertex represents a storyline character, and an edge points out an interaction between two characters. Edges may also have different nature; for instance, they can express that the names (or aliases) of two characters appear within a certain number of words apart [17,28], that a character *A* has spoken right after a character *B*, or that a character *A* and character *B* appear in a scene together [24]. The output graph is, commonly, an indirect and weighted network.

A network built this way is an example of artificial collaboration networks, as usually most pairs of characters have cooperated or have been antagonist one of another [18]. A typical analysis carried on this kind of network is determining the most important characters, according to several centrality measures. Some nodes play a massive role in the network, either by having many connections or by being strategically positioned to help connect distant parts of the network.

A character may, indeed, be relevant or influential with different facets, and it is fundamental to interpret these quantities with respect to the underlying domain. Another question an exploratory analysis of networks of characters aims to answer is to capture which characters naturally belong together, forming coherent communities within the network. Investigating these behavioral patterns means looking for coherent sub-plotlines hidden in the network [11].

Considering a character interactions network that is built upon characters' co-occurrence in movie scenes, it is straightforward to see that this kind of network can be naturally modeled with a hypergraph, where vertices are still associated with characters and hyperedges are related to scenes. In this case, the topology itself of the hypergraph allows us to easily find clusters of characters that commonly appear together within a movie or a TV series episode or season. To verify if hypergraphs are able to convey more information than a standard graph analysis approach also in the case of collaboration networks, we have modeled and analyzed the Game of Thrones TV series characters co-occurrence. As for the Yelp use case (see Section 4.1), we compare the findings obtained by exploiting hypergraphs with the results achieved using the two-section graphs.

The Game of Thrones TV series dataset. Game of Thrones [1] (GoT) is an American fantasy drama TV series, created by D. Benioff and D.B. Weiss for the American television network HBO. It is the screen adaptation of the series of fantasy novels *A Song of Ice and Fire*, written by George R.R. Martin. The series premiered on HBO in the United States on April 17, 2011, and concluded on May 19, 2019, with 73 episodes broadcast over eight seasons. With its 12 million viewers during season 8 and a plethora of awards—according to Wikipedia⁸—Game of Thrones has attracted record viewership on HBO and has a broad, active, and international fan base. The intricate world narrated by George R.R. Martin and scripted by Benioff and Weiss extend well beyond the boundaries of the traditional TV medium to create a deeply immersive entertainment experience [21]. This allows both the academic community and industries to study not only complex dynamics within the GoT storyline [24], but also how viewers engage with the GoT world on social media [19,58,61], or how the novel itself is a portrait of real-world dynamics [49,56,51,52,73].

This study is based on the dataset at the GitHub repository *Game of Thrones Datasets and Visualizations*⁹. Specifically, we made use of the data describing season episodes and containing meta-information about each of them, such as title, identification number, season, and description. Information about each scene within an episode is also reported. For each scene, start, end, location and a list of characters performing in it are listed. Table 4 reports some basic information about the number of episodes, scenes, and characters per GoT season. A more detailed description of the dataset can be found on the GitHub repository.

⁸ https://en.wikipedia.org/wiki/Game_of_Thrones

⁹ Game of Thrones Datasets and Visualizations <https://github.com/jeffreylancaster/game-of-thrones> by Jeffrey Lancaster

Season	Episodes	Scenes	Characters
1	10	286	125
2	10	468	137
3	10	470	137
4	10	517	152
5	10	508	175
6	10	577	208
7	7	468	75
8	6	871	66

Table 4: Some GoT dataset numbers.

Modeling Game of Thrones using hypergraphs. We studied GoT characters’ co-occurrences with different levels of granularity. We modeled the GoT dataset building three different types of hypergraphs, each one reporting whether the GoT characters have appeared in the same season, in the same episode, or in the same scene together. In the following, we describe the hypergraphs considered in our study:

- **Seasons.** A coarse-grained model is represented by the seasons hypergraph $H_{seasons} = (V, E_{seasons})$, where V represents GoT characters, and $E_{seasons}$ represents, for each GoT season s , the set of characters appearing in s . The hypergraph $H_{seasons}$ is shown in Figure 7, using an Euler-based visualization (see Section 3.5).
- **Episodes×Season.** An intermediate-grained model is obtained by considering co-occurrence within Episodes. In this case, we considered 8 different hypergraphs, $H_{episodes}^s = (V^s, E_{episodes}^s)$, $s \in [1, 8]$, where s indicates a GoT season, V^s represents the GoT characters appearing in season s , and $E_{episodes}^s$ represents, for each GoT episode e of season s , the set of characters appearing in e . The 8 hypergraphs $H_{episodes}^s$ are shown in Figure 8.
- **Scenes×Season.** A fine-grained model is obtained by considering co-occurrence within scenes. In this case, we considered 8 different hypergraphs, $H_{scenes}^s = (V^s, E_{scenes}^s)$, $s \in [1, 8]$, where s indicates a GoT season, V^s represents the GoT characters appearing in season s , and E_{scenes}^s represents, for each GoT scene f of season s , the set of characters appearing in f .

The collaboration structure of Game of Thrones. We performed a community detection task on the Scenes×Season hypergraphs and their corresponding two-section graphs. Running the community detection algorithm on such networks allows us to find coherent plotlines and, therefore, groups of characters frequently appearing in a scene together. In this experiment, our goal is to figure out whether and to what extent hypergraphs are able to capture characters’ collaboration (and, generally speaking, any type of user-defined collaboration)

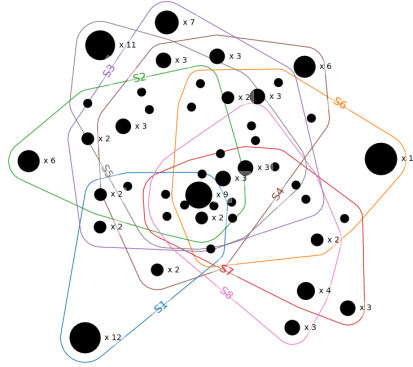


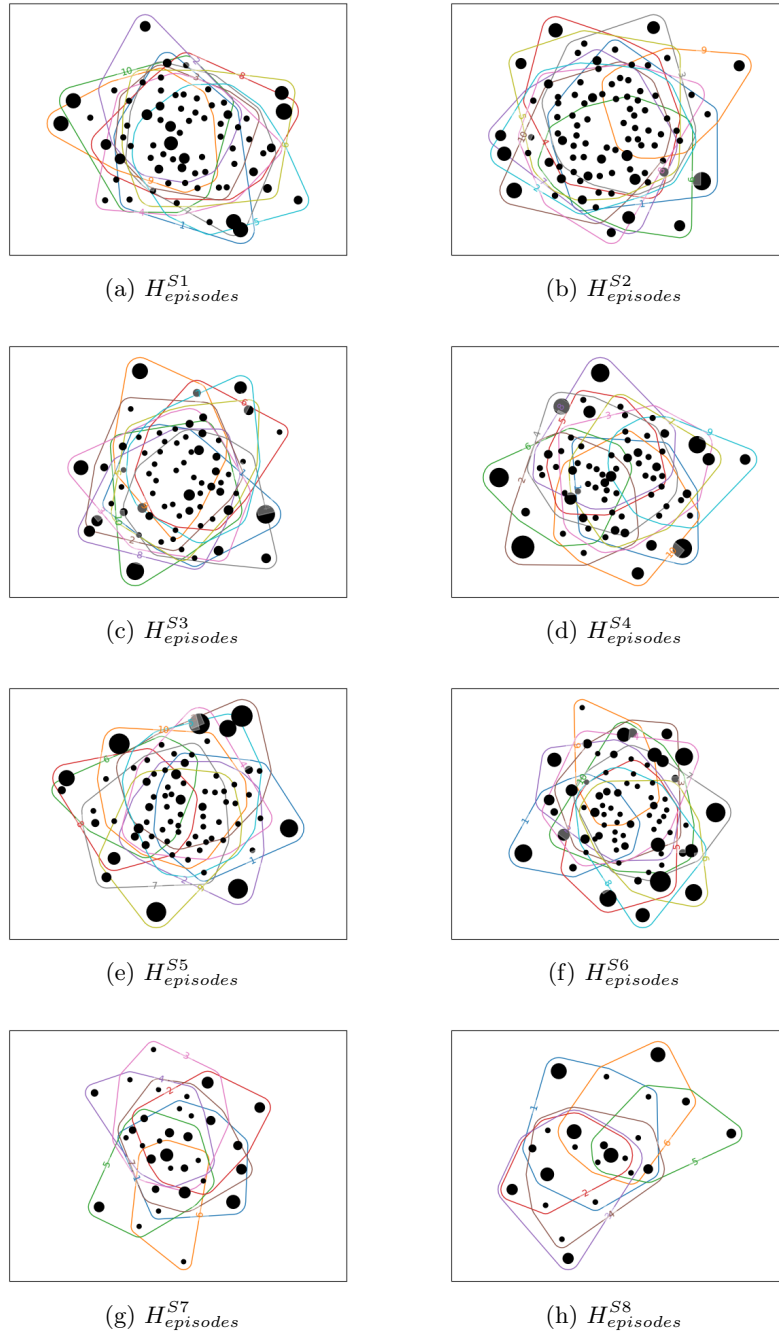
Fig. 7: The GoT season hypergraph $H_{seasons}$ defined by characters overlapping through seasons.

with respect to graphs. Specifically, we executed the label propagation algorithms defined in Section 4.1 and measured the quality of the solution obtained by computing the *modularity*. Results, described in Table 5, show that the solutions obtained for hypergraphs provide a higher number of communities. This pattern emerges particularly in the graphs describing the last two seasons, which are characterized by a smaller number of characters.

To measure the difference between the two approaches, we computed the NMI between the obtained partitions, shown in Table 6. Considering the NMI values, the partitions are strongly related, except for the last two seasons (the seasons with the lower number of characters). It is important to notice that the last two seasons exhibit the worst results. Nonetheless, we believe that it is well justified to the nature of their screenplay: fewer characters and high related plotlines. The equivalent hypergraphs are characterized by a high degree for both nodes and hyperedges. As a consequence, the corresponding two-section views result in *quasi-complete* graphs. In this particular case, the label propagation algorithm is not able to find out distinct communities.

	H_{scenes}^{S1}	H_{scenes}^{S2}	H_{scenes}^{S3}	H_{scenes}^{S4}	H_{scenes}^{S5}	H_{scenes}^{S6}	H_{scenes}^{S7}	H_{scenes}^{S8}
	(C , m)	(C , m)	(C , m)	(C , m)	(C , m)	(C , m)	(C , m)	(C , m)
H	(12, .5359)	(15, .4511)	(15, .7028)	(18, .5527)	(22, .5794)	(19, .5781)	(11, .2159)	(8, .1536)
G	(8, .3399)	(9, .6970)	(11, .7652)	(11, .6218)	(15, .7300)	(16, .7342)	(4, 2163)	(1, .0)

Table 5: A comparison between hypergraph and graph capability on discovering characters' communities. For each hypergraph (H) and its two-section graph (G), the table provides the number of communities ($|C|$) and the corresponding modularity values (m).

Fig. 8: GoT characters overlap through Episodes \times Season.

	<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>S4</i>	<i>S5</i>	<i>S6</i>	<i>S7</i>	<i>S8</i>
<i>NMI</i>	.82085	.83354	.92340	.87596	.88143	.88143	.69506	.0

Table 6: NMI values evaluated between the partitions obtained computing the communities on the Scenes×Season hypergraphs and their corresponding two-section graphs.

Discussion on season 8. It is worth discussing the interesting facts revealed by the results of the 8th season. The label propagation algorithm, in the case of the two-section graph, reveals only one big community — the whole graph itself. However, in the case of hypergraphs, it can determine eight different communities. In the following, we discuss the conflicting results obtained by providing a possible interpretation for the 8 discovered communities by the label propagation algorithm run on the hypergraph model.

In more detail, three minor communities of characters, appearing only in few scenes in the whole season, emerged from the (hyper)network structure. These communities are made up by: i) the *Winterfell* boy — appearing only in the first episode; ii) *Dirah*, *Craya*, *Marei* — seen only in the first episode; and iii) *Eleanor* and her daughter — occurring only in the fifth episode. Exploiting a hypergraph approach, the algorithm correctly identifies background characters that do not contribute to the main storyline and that are not strictly related to any main character. Other two communities pinpoint key characters belonging to the two central alliances: the *north* versus the *south*. The south-alliance is made up of *Cersei Lannister*, her counselor *Qyburn*, her guard *Gregor Clegane*, and her husband *Euron Greyjoy*; while the north-alliance is forged by Jon Snow and *Daenerys Targaryen*, with her dragons. In particular, in the north-alliance community also appear two enemies that have been faced by *Jon* and *Daenerys*: the *Night King* (and his white walker soldiers), and Harry Strickland, captain of the sellsword *Golden Company*. Two more communities can be labeled as north-allied: they contain a group of characters that consistently have interacted and/or fought together. Indeed, one group contains *Bran Stark* and *Theon Greyjoy* (who stands guard for him in the battle for *Winterfell*), *Lyanna Mormont* and *Wun Wun* (they fought against in the battle for *Winterfell*), *Lord Varys*, *Davos Seaworth*, *Grey Worm* and *Jorah Mormont*. The other community includes *Arya* and *Sansa Stark*, *Samwell Tarly* and his partner *Gilly*, *Brienne of Tarth* and *Jamie Lannister*, *Tyrion*, *Tormund*, *Missandei*, *Melisandre*, and *Sandor Clegane*, among few others. The algorithm also discovers a community related to the subplotline of *Yara Greyjoy* and some lords loyal to her: after having being saved by her brother *Theon*, she leaves the stage to claim her land. She reappears only in the last episode of the season, together with the other main characters. In this group, we can also find two royal background characters - *Edmure Tully* and *Robin Arryn* - that do not contribute to the development of the main plotline and only appear in the last episode.

Which are the most important characters? Identifying truly important and influential characters in a vast narrative like GoT may not be a trivial task, as it depends on the considered level of granularity. In these cases, the main character(s) in each plotline is referred with the term *fractal protagonist(s)*, to indicate that the answer to “who is the protagonist” depends on the specific plotline [11]. Following the same methodology of previous experiments, in this section, we focus on evaluating GoT characters according to both the degree and betweenness centrality metrics, exploiting Seasons×Scenes hypergraphs and the corresponding two-section graphs.

Degree centrality. Generally speaking, this metric gives information about the number of interactions of a node. If we consider a hypergraph H_{scenes}^s , the degree centrality of a node is the number of scenes in a given season s a character v appears in. In other words, we are enumerating the number of hyperedges where the vertex is contained. Formally,

$$C^H(v) = |E(v)|$$

Analogously, the degree centrality of a character v , on the associated two-section graph $G = [H_{scenes}^s]_2$, represents the number of characters he/she played with during season s . Formally,

$$CG(v) = deg(v)$$

Figure 9 shows that the information provided by the hypergraph analysis is much better to distinguish the centrality of characters. Indeed, each figure depicts the centrality values of the 10 most important characters of the corresponding season. Only $C^H(v)$ exhibits a clear trend, which is also more coherent among seasons. For instance, *Jon Snow* is the higher degree central character in the last 4 seasons, while *Tyrion Lannister* in the 2, 3, and 4 seasons.

Betweenness centrality. We investigated the importance of the characters also evaluating the betweenness centrality (BC) metric of hypergraph nodes. BC measures the centrality of a node by computing the number of times that a node acts as a bridge between the other two nodes, considering the shortest paths between them. Along the same line of *HyperNetX*[7], we define the *s-node-shortest-path* between two different nodes u and v is the shortest *s-node-walk* between them. A *s-node-walk* is a sequence of nodes (characters) such that they share at least s hyperedges. We notice that a 1-node-walk corresponds to a walk on a graph (or in on the two-section graph), and consequently, a 1-node-shortest-path is a classical shortest path. Moreover, using the *s-node-shortest-path* definition, we are able to compute the BC considering a path made using more robust connections (or interactions), which we suppose to be more precise to evaluate the importance of the characters in each season. Formally, the *s*-betweenness centrality is defined as

$$C_B^s = \sum_{x \neq v \neq y \in V} \frac{\sigma_{xy}^s(v)}{\sigma_{xy}^s},$$

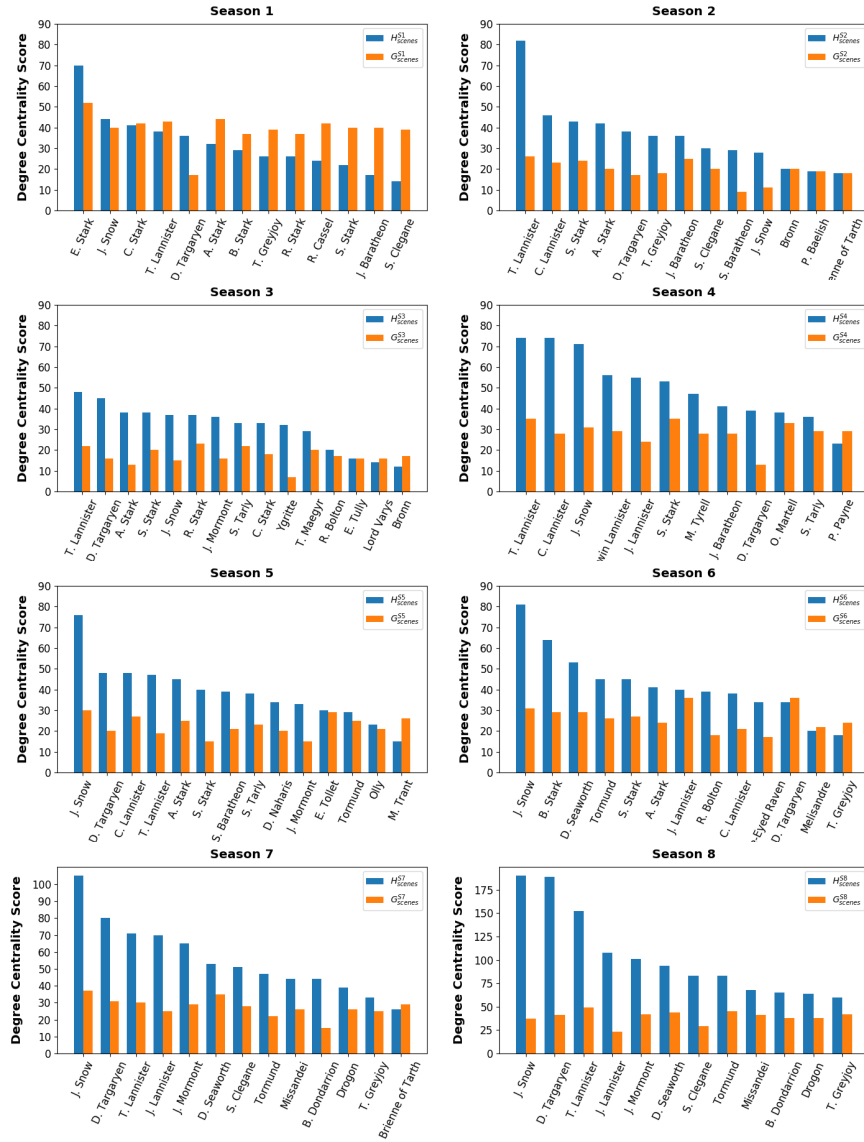


Fig. 9: GoT characters degree centrality scores per season.

where $\sigma_{xy}^s(v)$ is the number of the s -node-shortest-paths between two vertices x and y that pass through v , while σ_{xy}^s is the total number of s -node-shortest-paths between x and y . We notice that C_B^s generalizes the definition of betweenness centrality. Using $s = 1$, C_B^1 is the betweenness centrality of the two-section view of H . The rationale behind this generalized definition is that it allows measuring the centrality of the nodes according to a robust shortest path definition. We compared the obtained results varying the value of s from 1 to 3, measuring

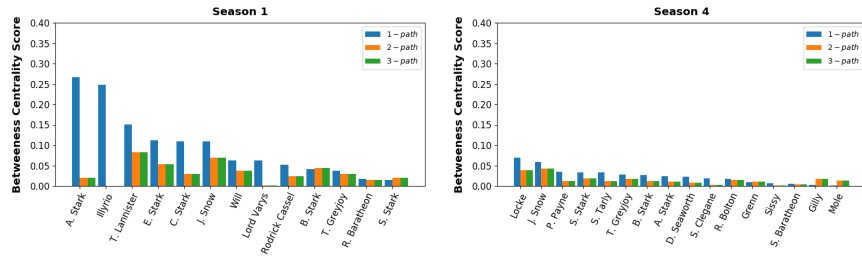


Fig. 10: GoT characters s -betweenness centrality scores per Season 1 and 4.

their correlation to understand whether they provide different information. In Table 7, we report the correlation scores for the season with minimum (season 1) and maximum (season 4) correlation. Furthermore, Figure 10 depicts the corresponding s -betweenness centrality values. It is worth mentioning, especially in season 1, that each character rank varies according to the s -value.

	$\rho(1, 2)$	$\rho(1, 3)$	$\rho(2, 3)$
Season 1 - MIN	-0.0078	0.1174	0.8366
Season 4 - MAX	0.8004	0.7962	0.7999

Table 7: Pearson correlation values for s -betweenness centrality using $\{1, 2, 3\}$ -path on the seasons with minimum and maximum correlation.

5 Conclusions

In this work, we have presented a new library for analyzing, exploring, and visualizing of hypergraph structures through an optimized set of hypergraph manipulations. The library, named `SimpleHypergraphs.jl`, provides Hypergraph views built exploiting the popular Julia library `LightGraphs.jl` for manipulating graphs. Functionalities for the I/O, manipulation, transformation, and visualization of hypergraphs have been developed and are available on a public GitHub repository. The library enables the user defining meta-information type as well as attaching meta-data values of arbitrary nature to both vertices and hyperedges. This approach allows programmers to efficiently analyze the structural properties of the network, combined with the possibility to perform semantic analysis based on the attached meta-data. To the best of our knowledge, `SimpleHypergraphs.jl` is the only hypergraph software tool providing this functionality.

Hypergraphs are a natural generalization of graphs and, at least theoretically, they provide a much richer structure than their well-known graph counterparts.

More importantly, they seem to be more suitable than graphs to model many natural phenomena that involve group-based interactions, such as collaborative activities. However, it is not still clear if the advantage of preserving a more detailed relationship structure justifies a more complicated data structure and, as a consequence, more complex underlying algorithms. In our work, we investigated this aspect of the research on hypergraphs by providing two case studies belonging to different domains. In the first case study, we explored the application of data analysis using hypergraphs for understanding users reviewing activities on the social network *Yelp.com*. In the second case study, we proposed the application of hypergraph analysis for discovering the community structure of a network society based on the interactions of characters in the *Game of Thrones* TV series. Results suggest that the hypergraph structure seems to improve the analysis of networks defined by *many-to-many* relationships, as they convey more information compared to the alternative view using the graph structure.

Future investigations are necessary to validate the conclusions drawn from this study. We plan to explore other types of data by widening the application domain. Furthermore, we are currently working on improving the library functionalities by implementing more algorithms and different accurate and engaging visualizations mechanisms.

References

1. Game of Thrones. <https://www.hbo.com/game-of-thrones>, 2019. [Online; 2019].
2. gspbox, MatLab. <https://github.com/epfl-lts2/gspbox>, 2019. [Online; 2019].
3. halp, Python. <https://github.com/Murali-group/halp>, 2019. [Online; 2019].
4. Hypergraph, Chapel. <https://github.com/pnnl/chgl>, 2019. [Online; 2019].
5. HyperGraphLib, C++ . <https://github.com/alex-87/HyperGraphLib>, 2019. [Online; 2019].
6. HyperNetX, Python. <https://github.com/pnnl/HyperNetX>, 2019. [Online; 2019].
7. HyperX, Scala. <https://github.com/jinhuang/hyperx>, 2019. [Online; 2019].
8. iper, JavaScript. <https://github.com/fibo/iper>, 2019. [Online; 2019].
9. LightGraphs.jl, Julia. <https://github.com/JuliaGraphs/LightGraphs.jl>, 2019. [Online; 2019].
10. multihypergraph, Python. <https://github.com/vaibhavkarve/multihypergraph>, 2019. [Online; 2019].
11. Network of Thrones. <https://networkofthrones.wordpress.com>, 2019. [Online; 2019].
12. networkR, R. <https://github.com/01sims/networkR>, 2019. [Online; 2019].
13. pygraph, Python. <https://github.com/jciskey/pygraph>, 2019. [Online; 2019].
14. yelp. <https://www.reuters.com/finance/stocks/company-profile/YELP.N>, 2019. [Online; 2019].
15. yelp-dataset. <https://www.yelp.com/dataset/challenge>, 2019. [Online; 2019].
16. yelp-dataset-docs. <https://www.yelp.com/dataset/documentation/main>, 2019. [Online; 2019].
17. AGARWAL, A., CORVALAN, A., JENSEN, J., AND RAMBOW, O. Social network analysis of alice in wonderland. In *Proceedings of the NAACL-HLT 2012 Workshop on Computational Linguistics for Literature* (Montréal, Canada, June 2012), Association for Computational Linguistics, pp. 88–96.
18. ALBERICH, R., MIRO-JULIA, J., AND ROSSELLO, F. Marvel universe looks almost like a real social network, 2002.
19. ANTELMÍ, A., BRESLIN, J., AND YOUNG, K. Understanding user engagement with entertainment media: A case study of the twitter behaviour of game of thrones (got) fans. In *2018 IEEE Games, Entertainment, Media Conference (GEM)* (Aug 2018), pp. 1–9.
20. ANTELMÍ, A., CORDASCO, G., SPAGNUOLO, C., AND VICIDOMINI, L. On evaluating graph partitioning algorithms for distributed agent based models on networks. In *Euro-Par 2015: Parallel Processing Workshops* (Cham, 2015), Springer International Publishing, pp. 367–378.
21. ASKWITH, IVAN D. Television 2.0 : reconceptualizing TV as an engagement medium. <https://dspace.mit.edu/handle/1721.1/41243>, 2007. [Online; 2019].
22. B., K., V., P., P., P., S., AND F., T. Clustering via hypergraph modularity. *PLoS ONE* 11, 11 (2019), e0224307.
23. BEEL, J., GIPP, B., LANGER, S., AND ET AL. Research-paper recommender systems: a literature survey. *Int J Digit Libr*, 17 (June 2016), 305–338.
24. BEVERIDGE, A., AND SHAN, J. Network of thrones. *Math Horizons* 23, 4 (2016), 18–22.
25. BEZANSON, J., EDELMAN, A., KARPINSKI, S., AND SHAH, V. B. Julia: A fresh approach to numerical computing. *SIAM review* 59, 1 (2017), 65–98.
26. BOCCALETTI, S., LATORA, V., MORENO, Y., CHAVEZ, M., AND HWANG, D.-U. Complex networks: Structure and dynamics. *Physics Reports* 424, 4 (2006), 175 – 308.

27. BODÓ, Á., KATONA, G. Y., AND SIMON, P. L. Sis epidemic propagation on hypergraphs. *Bulletin of mathematical biology* 78, 4 (2016), 713–735.
28. BONATO, A., D’ANGELO, D. R., ELENBERG, E. R., GLEICH, D. F., AND HOU, Y. Mining and modeling character networks. In *Algorithms and Models for the Web Graph* (Cham, 2016), A. Bonato, F. C. Graham, and P. Pralat, Eds., Springer International Publishing, pp. 100–114.
29. BONDY, J. A., AND MURTY, U. S. R. *Graph Theory with Applications*. Elsevier Science Publishing Company, Incorporated, 1976.
30. BRETTO, A. *Hypergraph Theory: An Introduction*. Springer Publishing Company, Incorporated, 2013.
31. CHEN, D.-B., SUN, H.-L., TANG, Q., TIAN, S.-Z., AND XIE, M. Identifying influential spreaders in complex networks by propagation probability dynamics. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 29, 3 (mar 2019), 033120.
32. DANON, L., DÍAZ-GUILERA, A., AND DUCH, J. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment* (2005).
33. EDELMAN, A. Julia: A fresh approach to technical computing and data processing. Tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE CAMBRIDGE, 2019.
34. EL-MOUSSAOUI, M., AGOUTI, T., TIKNIOUINE, A., AND ADNANI, M. E. A comprehensive literature review on community detection: Approaches and applications. *Procedia Computer Science* 151 (2019), 295–302.
35. FORTUNATO, S. Community detection in graphs. *Physics Reports* 486, 3-5 (feb 2010), 75–174.
36. GAN, C., YANG, X., LIU, W., ZHU, Q., JIN, J., AND HE, L. Propagation of computer virus both across the internet and external computers: A complex-network approach. *Communications in Nonlinear Science and Numerical Simulation* 19, 8 (aug 2014), 2785–2792.
37. GÖBEL, M., AND ARAÚJO, T. A network structure analysis of economic crises. In *Complex Net. and Their App. VIII* (2020), pp. 547–560.
38. GULATI, A., AND EIRINAKI, M. Influence propagation for social graph-based recommendations. In *2018 IEEE International Conference on Big Data (Big Data)* (2018), pp. 2180–2189.
39. HAN, Y., ZHOU, B., PEI, J., AND JIA, Y. Understanding importance of collaborations in co-authorship networks: A supportiveness analysis approach. In *Proceedings of the 2009 SIAM International Conference on Data Mining* (2009), SIAM, pp. 1112–1123.
40. HASENJAGER, M. J., HOPPITT, W., AND LEADBEATER, E. Network-based diffusion analysis reveals context-specific dominance of dance communication in foraging honeybees. *Nature Communications* 11, 1 (jan 2020).
41. HOSSAIN, M., KHAN, A., AND UDDIN, S. Understanding the progression of congestive heart failure of type 2 diabetes patient using disease network and hospital claim data. In *Complex Net. and Their App. VIII* (2020).
42. INTERDONATO, R., ATZMUELLER, M., GAITO, S., KANAWATI, R., LARGERON, C., AND SALA, A. Feature-rich networks: going beyond complex network topologies. *Applied Network Science* 4, 1 (jan 2019).
43. JI, Z., PI, H., WEI, W., XIONG, B., WOŹNIAK, M., AND DAMASEVICIUS, R. Recommendation based on review texts and social communities: A hybrid model. *IEEE Access* 7 (2019), 40416–40427.
44. KAMIŃSKI, B., POULIN, V., PRALAT, P., SZUFEL, P., AND THÉBERGE, F. Clustering via hypergraph modularity. *PloS one* 14, 11 (2019).

45. KLAMT, S., HAUS, U.-U., AND THEIS, F. Hypergraphs and cellular networks. *PLoS computational biology* 5, 5 (2009), e1000385.
46. LI, D., XU, Z., LI, S., AND SUN, X. Link prediction in social networks based on hypergraph. In *Proceedings of the 22nd International Conference on World Wide Web* (2013), ACM, pp. 41–42.
47. LI, R., JIANG, J.-Y., JU, C. J.-T., AND 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* (2019), WSDM ’19, pp. 69–77.
48. LIU, D., BLENN, N., AND MIEGHEM, P. V. Characterising and modelling social networks with overlapping communities. *Int. J. Web Based Communities* 9, 3 (June 2013), 371–391.
49. LOVRIC, B., AND HERNÁNDEZ, M. The house of black and white: Identities of color and power relations in the game of thrones. *Revista Nós* 4, 2 (9 2019), 161–182.
50. LU, X., QU, J., JIANG, Y., AND ZHAO, Y. Should i invest it?: Predicting future success of yelp restaurants. In *Proceedings of the Practice and Experience on Advanced Research Computing* (2018), PEARC ’18, pp. 64:1–64:6.
51. MILKOREIT, M. Pop-cultural mobilization: Deploying game of thrones to shift us climate change politics. *International Journal of Politics, Culture, and Society* 32, 1 (Mar 2019), 61–82.
52. MUNO, W. *“Winter Is Coming?” Game of Thrones and Realist Thinking*. Springer International Publishing, Cham, 2019, pp. 135–149.
53. MUSTO, C., BASILE, P., LOPS, P., DE GEMMIS, M., AND SEMERARO, G. Introducing linked open data in graph-based recommender systems. *Information Processing & Management* 53, 2 (2017), 405 – 435.
54. NEWMAN, M. E., AND GIRVAN, M. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.
55. NGUYEN, V. X., XIAO, G., XU, X.-J., WU, Q., AND XIA, C.-Y. Dynamics of opinion formation under majority rules on complex social networks. *Scientific Reports* 10, 1 (jan 2020).
56. OLESKER, R. Chaos is a ladder: A study of identity, norms, and power transition in the game of thrones universe. *The British Journal of Politics and International Relations* 0, 0 (2019), 1369148119885065.
57. PENG, Y., SHI, J., FANTINATO, M., AND CHEN, J. A study on the author collaboration network in big data. *Information Systems Frontiers* 19, 6 (jun 2017), 1329–1342.
58. PÉREZ, H. J., AND REISENZEIN, R. On jon snow’s death: Plot twist and global fandom in game of thrones. *Culture & Psychology* 0, 0 (2019), 1354067X19845062.
59. RAGHAVAN, U. N., ALBERT, R., AND 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).
60. 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).
61. RHODES, R. E., AND ZEHR, E. P. Fight, flight or finished: forced fitness behaviours in game of thrones. *British Journal of Sports Medicine* 53, 9 (2019), 576–580.
62. ROMERO, M., FINKE, J., QUIMBAYA, M., AND ROCHA, C. In-silico gene annotation prediction using the co-expression network structure. In *Complex Net. and Their App. VIII* (2020).

63. SCHARNHORST, A. Complex networks and the web: Insights from nonlinear physics. *Journal of Computer-Mediated Communication* 8, 4 (jun 2006), 0–0.
64. SILVA, J., AND WILLETT, R. Hypergraph-based anomaly detection of high-dimensional co-occurrences. *IEEE transactions on pattern analysis and machine intelligence* 31, 3 (2008), 563–569.
65. SILVEIRA, T., Z.-M. L. X., AND ET AL. How good your recommender system is? a survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics* 10, 5 (May 2019), 813–831.
66. STAVANJA, J., AND KLEMEN, M. Predicting kills in game of thrones using network properties, 2019.
67. STROGATZ, S. H. Exploring complex networks. *Nature* 410 (2001).
68. TUFTE, E. *The Visual Display of Quantitative Information*. Graphics Pr, 1983.
69. VARGAS, D. L., BRIDGEMAN, A. M., SCHMIDT, D. R., KOHL, P. B., WILCOX, B. R., AND CARR, L. D. Correlation between student collaboration network centrality and academic performance. *Physical Review Physics Education Research* 14, 2 (oct 2018).
70. VERBA, M. A. “learning hubs” on the global innovation network. In *Complex Net. and Their App. VIII* (2020), pp. 620–632.
71. VINH, N. X., EPPS, J., AND BAILEY, J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.* 11 (2010), 2837–2854.
72. YUAN, X., SUN, M., CHEN, Z., GAO, J., AND LI, P. Semantic clustering-based deep hypergraph model for online reviews semantic classification in cyber-physical-social systems. *IEEE Access* 6 (2018), 17942–17951.
73. ZARE, Z. Representation of gender roles in child and young characters in game of thrones series. *IAU International Journal of Social Sciences* 9, 2 (2019), 83–91.
74. ZHOU, D., HUANG, J., AND SCHÖLKOPF, B. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in neural information processing systems* (2007), pp. 1601–1608.