Humanities & Social Sciences Communications

ARTICLE

https://doi.org/10.1057/s41599-025-05180-5 OPEN

EasyHypergraph: an open-source software for fast and memory-saving analysis and learning of higher-order networks

Bodian Ye[1], Min Gao[1], Xiu-Xiu Zhan[2], Xinlei He[3], Zi-Ke Zhang[4], Qingyuan Gong[5]✉, Xin Wang[1] & Yang Chen[1]✉

Higher-order relationships exist widely across different disciplines. In the realm of real-world systems, significant interactions involving multiple entities are common. The traditional pairwise modeling approach leads to the loss of important higher-order structures, while hypergraph is one of the most typical representations of higher-order relationships. To deeply explore the higher-order relationships, researchers and practitioners use hypergraph analysis to model the higher-order relationships and describe the important topological features in higher-order networks. At the same time, they carry out hypergraph learning studies to learn better node representations by designing hypergraph neural network models. However, existing hypergraph libraries still have the following research gaps. The first is that most of them are not able to support both hypergraph analysis and hypergraph learning, which negatively impacts the user experience. The second is that the existing libraries exhibit insufficient computational performance, which causes researchers and practitioners to spend more time and incur expensive resource costs. To fill these research gaps, we present EasyHypergraph, a comprehensive, computationally efficient, and storage-saving hypergraph computational library. To ensure comprehensiveness, EasyHypergraph designs data structures to support both hypergraph analysis and hypergraph learning. To ensure fast computation and efficient memory utilization, EasyHypergraph designs the computational workflow and demonstrates its effectiveness. Through experiments on five typical hypergraph datasets, EasyHypergraph saves at most 8470 s and 935 s over two baseline libraries in terms of analyzing node distance on a dataset with more than one hundred thousand nodes. For hypergraph learning, EasyHypergraph reduces HGNN training time by approximately 70.37% in a similar scenario. Finally, by conducting case studies for hypergraph analysis and learning, EasyHypergraph exhibits its usefulness in social science research.

[1] College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China. [2] Research Center for Complexity Sciences, Hangzhou Normal University, Hangzhou, China. [3] Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China. [4] Center for Digital Communication Studies, Zhejiang University, Hangzhou, China. [5] Research Institute of Intelligent Complex Systems, Fudan University, Shanghai, China.
✉email: gongqingyuan@fudan.edu.cn; chenyang@fudan.edu.cn

Introduction

Hypergraph is a generalization of simple networks in which a hyperedge can connect any number of nodes (Agarwal et al. 2006; Feng et al. 2019), and also a critical methodology for research in complex systems (Fic and Gokhale, 2024; Lerman et al. 2024; Wang et al. 2024). In contrast to simple networks that only model pairwise relationships, hypergraphs can represent more complex relationships among entities, thereby avoiding the loss of higher-order information (Varley, 2024; Wang and Kleinberg, 2024). It enables a more precise representation of real-world systems and enhances network-based analysis, thereby advancing data mining and knowledge discovery in the fields of sociology (Fic and Gokhale, 2024; Rodríguez-Casañ et al. 2024; Schwartz, 2021), development studies (Kim et al. 2024a), economics (Li et al. 2024a; Xi et al. 2024), and education (Cheng et al. 2024; Landaeta-Torres et al. 2024). Specifically, in the academic collaboration scenario between more than two researchers (Newman, 2001), hypergraphs offer more precise representations that help us deduce both collaborative links among authors and the specific papers they have published together. However, with a simple network, we can only represent a collaboration between two authors, and might even incorrectly project the original higher-order relationship to a pairwise link.

Therefore, more researchers and practitioners explore the potential of higher-order structure using hypergraphs, and propose a number of theoretical studies (Lung et al. 2018; Ramadan et al. 2004). This paper concerns two research areas in the field of hypergraphs. Here, we refer to the quantization and characterization of hypergraph structures at multiple scales as hypergraph analysis and node representation learning through hypergraph neural networks (HNNs) as hypergraph learning. Hypergraph analysis enables researchers and practitioners to explore the patterns of target networks, which prompted the emergence of research topics such as node ranking, hyperedge ranking, and hypergraph connectivity. Hypergraph learning exhibits a superior capability to capture the higher-order relationships and increase the precision of downstream tasks in target networks by designing specific learning components (Feng et al. 2019; Gao et al. 2022a).

However, existing theoretical studies are implemented in various forms and dispersed across different platforms, which increases the difficulty and cost for researchers and practitioners to carry out related studies. To address the above problems, several hypergraph computation libraries are presented, which are shown in Table 1. Functionally, most existing libraries support hypergraph analysis and structure visualization, while only one library focuses on hypergraph learning. In terms of development, some libraries partially depend on a graph library called NetworkX (Hagberg et al. 2008) for metrics computation, which might be limited by the performance of NetworkX and constrain their scalability. Besides, several libraries are in a state of discontinued development.

Although existing hypergraph computation libraries partially meet the needs of multidisciplinary researchers and practitioners, several critical issues remain unresolved. The primary limitation lies in their data structure design, which lacks a unified framework for hypergraph computation. This architectural constraint prevents researchers and practitioners from conducting hypergraph analysis and learning tasks conveniently. As shown in Table 1, existing libraries are limited to providing only a restricted range of metrics or concentrating solely on implementing HNNs. These issues drive researchers and practitioners to shift between libraries and lead to a time-consuming process. Secondly, existing libraries face challenges with increased time and memory consumption when handling large-scale datasets. This inefficiency might stem from their external dependencies, sub-optimized metric implementations, and ineffective handling of higher-order relationships.

To fill the research gap of existing hypergraph libraries, the first goal of EasyHypergraph is to design a unified hypergraph computation framework that supports both hypergraph analysis and hypergraph learning. Then, we should extend this framework to provide researchers and practitioners with two key capabilities: 1) rich hypergraph analysis functions and hypergraph learning models, and 2) high-performance computing for large-scale data scenarios by efficiently utilizing computational resources. Based on the research goals, we summarize the contributions of this paper as follows:

1. We design and build EasyHypergraph, a comprehensive software library that unifies hypergraph analysis and learning. Our design enables hypergraph analysis and hypergraph learning to be implemented within a unified framework, which helps users deal with different hypergraph-relevant functions efficiently without relying on external libraries.
2. EasyHypergraph incorporates efficient computational workflows to enable optimization of hypergraph analysis and learning, achieving superior computation speed and memory efficiency in processing large-scale data compared with existing libraries. The computational workflow firstly divides hypergraph properties into basic and advanced properties. Then, EasyHypergraph achieves a computation acceleration by designing loop fusion, preloading, and caching mechanisms to optimize both types of properties. Furthermore, EasyHypergraph is built with a sparse storage format selection strategy for memory efficiency by characterizing the hypergraph incidence matrix. Experiments for hypergraph analysis demonstrate significant performance improvements. The average speedup ratio on all compared metrics calculations could reach 11,155. In scenarios with tens of thousands of nodes, EasyHypergraph reduces memory usage by up to 54.28% and 44.17% for the incidence matrix compared with HNX and XGI, respectively. Meanwhile, our learning model implementation can achieve a faster training speed than DHG. Specifically, our implementation of the HGNN model achieves up to a 70.37% reduction in training time compared with DHG when the data size exceeds hundreds of thousands.
3. EasyHypergraph demonstrates its availability and flexibility in hypergraph analysis and hypergraph learning by serving

Table 1 Comparison between traditional hypergraph computation libraries.

Library	Hypergraph analysis	Hypergraph learning	Structure visualization	NetworkX dependency	Continuous development
XGI	✓	-	✓	✓	✓
HNX	✓	-	✓	✓	✓
HGX	✓	-	✓	✓	✓
Reticula	✓	-	-	-	✓
Halp	✓	-	-	-	-
DHG	-	✓	✓	-	✓
EasyHypergraph	✓	✓	✓	-	✓

two case studies. For hypergraph analysis, EasyHypergraph is able to characterize the structural properties through a suite of hypergraph metrics and algorithms for two congressional committee datasets. Besides, EasyHypergraph exhibits its potential in multi-class node classification tasks. By utilizing two widely adopted co-citation datasets to classify scientific publications into distinct categories, HNNs implemented by EasyHypergraph outperform traditional graph convolutional networks in prediction accuracy, precision, and recall.

Literature review

The concept of hypergraph and relevant libraries. Hypergraph is an extended form of a simple graph, first proposed by Berge (Berge, 1985) in 1973. In a hypergraph, hyperedges can connect any number of nodes, enabling more flexible modeling of relationships among nodes. This flexibility allows hypergraphs to better represent complex relationships among multiple nodes, such as higher-order interactions. Given a hypergraph $\mathcal{H}$, its node set is denoted as $V = \{v_1, v_2, \ldots, v_N\}$, and its hyperedge set is denoted as $E = \{e_1, e_2, \ldots, e_M\}$, and the definitions can be formalized as follows:

$$\mathcal{H} = (V, E) \tag{1}$$

However, modeling hypergraphs faces the challenge of how to represent nodes and hyperedges. The original idea is to project (Yadati et al. 2019; Yan et al. 2024; Yang et al. 2022) the hypergraph into a simple graph and then analyze it by a graph analysis library such as EasyGraph (Gao et al. 2023). EasyGraph is one of the representative multifunctional and effective Python libraries for interdisciplinary graph analysis. Although EasyGraph can facilitate the analysis of projected hypergraphs, existing studies (Wang and Kleinberg, 2024) have also found that the current hypergraph projection algorithms might cause higher-order information loss. Therefore, researchers have to make trade-offs on whether or not to perform a hypergraph projection based on the target scenario. In summary, it is necessary to develop a hypergraph computation library with flexibility in modeling hypergraphs in the form of pair-wise or higher-order structures. Currently, several end-to-end hypergraph computation libraries have been developed, which are listed below:

- HNX (HyperNetX) (Laboratory, 2023) is a Python-based hypergraph library that covers a wide range of algorithms and metrics for hypergraph analysis, such as centrality measures (Estrada and Rodríguez-Velázquez, 2006a), identification of motifs (Lotito et al. 2022).
- XGI (CompleX Group Interactions) (Landry et al. 2023) is a hypergraph library that is implemented in Python, which partly depends on NetworkX. It contains a comprehensive hypergraph analysis pipeline, including algorithms for hypergraph structure analysis and dynamic hypergraph simulation.
- DHG (DeepHypergraph) (Dai and Gao, 2023) is a representative hypergraph deep learning library, and it is inherited and extended from the THU-HyperG (Gao et al. 2022b). DHG can not only implement the basic structure of hypergraph for hypergraph learning, but also provide many basic operations during the learning process. Besides, the library contains modules for loading hypergraph datasets, visualizing hypergraphs, and metrics for evaluation.

In addition to the aforementioned hypergraph computation libraries in the community, several other open-source hypergraph libraries are noteworthy. For instance, HGX (Hypergraphx) (Lotito et al. 2023) is a versatile hypergraph analysis library that offers a comprehensive suite of hypergraph analysis metrics and algorithms, including centrality measures (Estrada and Rodríguez-Velázquez, 2006b), motif identification (Lotito et al. 2022), and community detection (Tudisco and Higham, 2023). Another notable library, Reticula (Badie-Modiri and Kivelä, 2023), specializes in the analysis of temporal hypergraphs-those incorporating time-dependent attributes. It provides methods for understanding temporal correlations within hypergraphs, offering unique insights into dynamic higher-order interactions. Most of these tools enable users to conduct in-depth analysis of hypergraphs, uncovering their structural features and patterns.

A review of hypergraph analysis. Hypergraph analysis could quantitatively characterize the higher-order networks based on the hypergraph structure. It usually derives from the following properties. The structural representation of the hypergraph can be described by the incidence matrix H, which is defined as

$$H(v_i, e_j) = \begin{cases} 1, v_i \in e_j \\ 0, else. \end{cases} . \tag{2}$$

The connectivity of the nodes in the hypergraph can be calculated based on the hypergraph adjacency matrix, where the definition of the adjacency matrix A is described as

$$A = HH^\top. \tag{3}$$

In addition, hyperdegree and hyperedge degree are also important properties in hypergraph analysis, which can describe the connectivity of nodes/hyperedges. Their definitions are

$$D(v) = \sum_{e \in E} W(e) \cdot H(v, e), \tag{4}$$

and the hyperedge degree is defined as

$$D(e) = \sum_{v \in V} H^\top(e, v). \tag{5}$$

where $W(e)$ represents the weight of each hyperedge, which defaults to 1 if not specified. To achieve hypergraph analysis, a range of theoretical studies is proposed, which are introduced below.

Node ranking and hyperedge ranking. Node ranking quantifies the importance of nodes in the hypergraph, and different metrics quantify node importance from different perspectives. Apart from hyperdegree, there are also other important metrics. Kovalenko et al. (2022) proposed vector centrality, which quantifies the importance of nodes by first constructing a line graph where nodes correspond to hyperedges in the original hypergraph. They first calculated the eigenvector centrality (Ruhnau, 2000) of nodes in the line graph. Then, they defined the vector centrality for each node in the hypergraph based on the centrality of the hyperedges it belongs to. Mancastroppa et al. (2023) proposed hypercoreness to describe the role of a node in the dissemination and reception of information within a network. Its idea is to perform k-core decomposition on the hypergraph to obtain the maximum number of layers, so as to calculate the proximity of each node to the hypergraph core. Nodes with higher core values play a more important role in the network. Additionally, Fan et al. (2021) introduced a metric called cycle ratio. This metric involves calculating the number of shortest cycle structures. The greater the number of cycle structures a node is part of, the more influential it is within the network.

Apart from node ranking, hyperedge ranking evaluates the importance of hyperedges. Vasilyeva et al. (2023) proposed s-closeness centrality. This metric reflects the role of each hyperedge in the network by calculating the shortest path between a hyperedge and all other hyperedges. Besides, Lee et al. (2021) extended the betweenness centrality in simple graphs to hypergraphs by projecting a hypergraph to a line graph.

Connectivity. Researchers also studied the connectivity of the hypergraph by calculating the connected components. From the perspective of global structure, Dewar et al. (2018) ported the concept of connected components on simple graphs to hypergraphs. The connected components in hypergraphs are essentially maximal connected weak sub-hypergraphs. Besides, the distance (Vasilyeva et al. 2023) between nodes within the hypergraph is used to indicate inter-node reachability. From the perspective of local structure, Gallagher and Goldberg (2013) measured the clustering coefficient of node neighborhoods by considering independent and dependent connections between node neighbors, as well as the overlap of adjacent hyperedges.

Null model. Null models generate random hypergraphs with specific properties, serving as benchmarks for statistical significance testing. The simplest random hypergraph null model is similar to the pair-wise network generation algorithm proposed by Erdös et al. (1960). One needs to specify the number of nodes, the number of hyperedges, and the probability of hyperedge generation. Then, this null model will generate a uniform hypergraph with the same hyperdegree for all nodes. Besides, Aksoy et al. (2017) proposed a Chung-Lu random model to generate random hypergraphs that satisfy a specified degree distribution. Kook et al. (2020) proposed a hypergraph null model that can reproduce five structural patterns from previous studies using only two parameters. Juul et al. (2024) introduced m-patterns, which are simple hypergraphs with m nodes, and they employed the hypergraph null model to understand the prevalence of m-patterns to formalize relationships between collaborators.

Applications. Various researchers apply hypergraph analysis to fulfill their studies. For example, hypergraph analysis can help researchers better understand how different social network structures influence group synchronization and decision-making behaviors through higher-order interactions, providing a crucial research perspective for uncovering the collective dynamics of complex systems (Chen et al. 2025; Lucas et al. 2020; Zhang et al. 2021). Besides, hypergraph analyses are also adopted to identify critical and evolving patterns of residents' post-disaster needs over time (Zhao et al. 2024). At the same time, hypergraphs are capable of transforming building geometries into computable graphical structures for solving problems related to space utilization and carbon emissions in the field of architecture (Weber et al. 2024). In the field of cybersecurity (Jia et al. 2025), hypergraphs can readily integrate strategies and techniques for cyber attack analysis directly into the network structure, which enables researchers to increase the compactness of modeling (Guzzo et al. 2017) and extract diverse relationships among security entities more efficiently.

A review of hypergraph learning. There is a wide range of higher-order relationships in real life, and it covers structural patterns that are usually invisible in simple networks (Kim et al. 2024b). Moreover, the existing GNNs (Graph Neural Networks) do not consider higher-order interactions. Therefore, HNNs are proposed to solve these challenges. To achieve hypergraph learning, users should prepare node/hyperedge features and encode them into a high-dimensional vector representation (e.g., node representation). Different from the GNNs, which only pass messages between pairwise nodes, HNNs can pass messages from nodes to hyperedges, hyperedges to nodes, or use even more complicated mechanisms. Finally, the node representation is learned by crafting the above mechanisms after rounds of training. The node representation can be utilized to improve the precision of node classification and hyperlink prediction. In summary, hypergraph learning refers to representation learning using HNNs (Kim et al. 2024b).

Existing HNNs can be broadly categorized into spectral-based approaches (Feng et al. 2019; Yadati et al. 2019) and spatial-based approaches (Dong et al. 2020; Gao et al. 2022a; Huang and Yang, 2021; Yan et al. 2024). The representative work spectral-based method is HGNN (Feng et al. 2019). The basic principle of HGNN is to extend the traditional graph Laplacian operator and graph Fourier transform to the domain of hypergraph. Then, to achieve the effect of node-to-edge and edge-to-node message passing and integration, the hypergraph signals are further converted from the spatial domain to the frequency domain. The representative spatial-based work is HGNN$^+$. This model defines a two-stage convolutional paradigm that inputs the node features of the hypergraph with the hyperedge features into the corresponding HNN. For each round of training, two separate steps of message passing and information integration are required. Firstly, each node receives the messages from hyperedges corresponding to its neighbors. Subsequently, a node integrates received messages and updates its representation. Next, the direction of message passing is reversed, and each hyperedge can update its representation. In addition to the classical message-passing mechanism in the above methods, Chien et al. (2022) proposed a new hypergraph neural network paradigm called AllSet. The key idea of AllSet is to dynamically learn multiset functions, transforming the existing message-passing mechanism into a combination of two multiset functions to achieve generalization of hypergraph learning.

Applications. Hypergraph learning has also found widespread applications across diverse domains. In social network recommendation, researchers can integrate a variety of factors such as user ratings, user profiles, features of the recommended targets, and contextual information into the hypergraph modeling process. Then, the hypergraph could be used to model multi-dimensional higher-order interaction information for recommendation (Guo et al. 2025; Ma et al. 2022; Xia et al. 2021; Yu et al. 2021) and improve the reliability of the recommendation. In the field of malicious detection, Xu et al. (2024) used hypergraphs to fuse malicious triggers in textual communication, such as sentiment context, topic context, user profile context, and interaction context. For fake news detection, Su et al. (2025) abstracted fake news detection as hyperedge classification and captured the higher-order correlation between news and users using HNNs. To address the hidden nature of trajectory interactions and achieve a better trajectory prediction, Xu et al. (2022) constructed multi-scale hypergraphs to model complex social influences based on the correlation of action trajectories.

Methods

The goal of this paper is to present a hypergraph computation library that seamlessly adapts to hypergraph analysis and hypergraph learning scenarios. EasyHypergraph should primarily provide users with a unified class for hypergraph modeling, a comprehensive set of hypergraph analysis functions, and hypergraph learning models. Meanwhile, we also need to design an efficient computational workflow to ensure that the library achieves a superior computational performance and resource efficiency, thereby reducing the cost and effort for multidisciplinary researchers and practitioners.

Unified framework for hypergraph analysis and hypergraph learning. EasyHypergraph not only provides a unified data

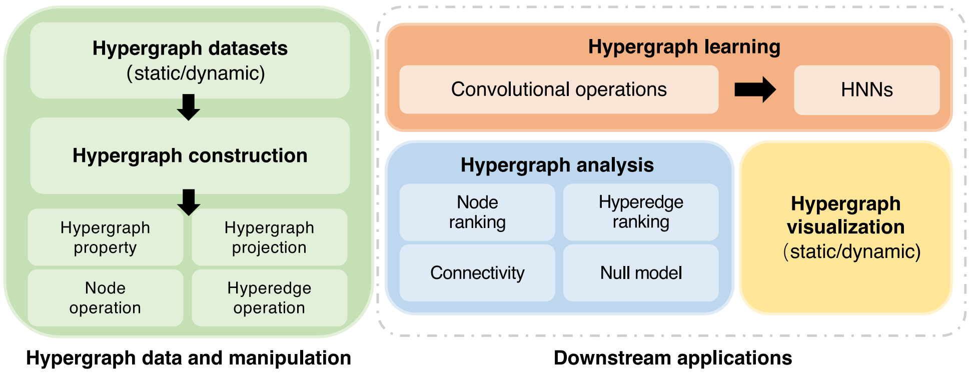


Fig. 1 General functions of EasyHypergraph. Users can load various hypergraph datasets and manipulate hypergraphs by node/hyperedge operation, accessing properties, or projecting hypergraphs into simple graphs. Furthermore, modules in the right part are combined and augment users' ability to apply hypergraph analysis, learning, and visualization.

structure for hypergraph but also encapsulates a wide range of mainstream functions for hypergraph analysis and hypergraph learning. The general framework is described in Fig. 1, and we follow the design logic of building before applying. Therefore, the module on the left (building) of Fig. 1 supports the three modules (applying) on the right. Below, we will introduce it from left to right.

Firstly, the core functionality of EasyHypergraph is built upon the hypergraph data and manipulations module, which is designed to support the following submodules. 1) Hypergraph datasets. It provides a collection of typical hypergraph datasets, particularly those related to social networks (e.g., the datasets listed in Table 3. 2) Hypergraph construction. Users can load hypergraph datasets and construct hypergraphs by instantiating the hypergraph class, which facilitates manipulations at various levels of granularity. 3) Hypergraph manipulations. It contains four parts in the left part of Fig. 1. They support a range of operations, including accessing hypergraph properties, performing hypergraph projection to derive a corresponding simple graph, and modifying the hypergraph structure through node/hyperedge operations (e.g., add and remove).

Then, based on the constructed hypergraph class, users can utilize the following three modules for downstream applications.

1. *Hypergraph analysis module*. This module contains various features and corresponding functions, which are shown in Table 2. EasyHypergraph covers a rich set of node ranking algorithms (Estrada and Rodríguez-Velázquez, 2006b; Fan et al. 2021; Kovalenko et al. 2022; Mancastroppa et al. 2023), hyperedge ranking algorithms (Aksoy, Sinan G. et al. 2020), connectivity metrics, and hypergraph null models for describing common hypergraph topologies (Gallagher and Goldberg, 2013; Klimm et al. 2021; Zhou and Nakhleh, 2011).
2. *Hypergraph learning module*. This module consists of HNNs based on a series of convolutional operations. As shown in Table 2, EasyHypergraph provides a typical hypergraph neural network model. Users are only required to define the input feature dimensions, hidden sizes, and output feature dimensions. Then, they can input the corresponding hypergraph structure during each round of training, which can quickly build a hypergraph learning pipeline.
3. *Hypergraph visualization module*. Users are able to observe the hypergraph topology information more intuitively through the static visualization function that this module offers. To meet the users' needs to observe the evolution of dynamic hypergraphs, EasyHypergraph enables users to visualize dynamic hypergraphs by providing timestamp parameters on top of the static hypergraph visualization function. Each hypergraph structure is visualized according to the order of timestamps. An example of dynamic hypergraph visualization is shown in Fig. 2. Here, it should be noted that the focus of this paper is not the design of hypergraph layout algorithms, which will not be discussed here.

Finally, EasyHypergraph bridges the gap between EasyGraph and higher–order relationships. EasyHypergraph is developed as an integrated library within the EasyGraph framework, maintaining full compatibility with its core architecture. Specifically, one hypergraph modeled by EasyHypergraph can be projected into a simple graph in EasyGraph by hypergraph projection algorithms (Antelmi et al. 2023). Then, users can utilize the algorithms associated with simple graphs in EasyGraph for further exploration without relying on external libraries.

Computational workflow for accelerating hypergraph analysis and hypergraph learning. Researchers and practitioners have realized that hypergraph has the advantage of modeling higher–order interactions in different domains (Çatalyürek et al., 2007), which attracts more emergent research, and the scale of hypergraph datasets available for the study could reach ten thousand magnitudes. More importantly, it might take an unacceptable time to obtain results. To address the challenges of large-scale hypergraph datasets with limited computational resources, we design a computationally efficient and storage-saving computational workflow. The overview of our computational workflow is shown in Fig. 3, along with its relation to hypergraph downstream applications.

Firstly, the computational workflow is designed based on the hypergraph class. The hypergraph class is used to model the hypergraph programmatically, which requires the user to input only two parts of the necessary information, i.e., the number of nodes contained in the hypergraph and the list of hyperedges. The number of nodes is determined to ensure that the nodes can be discretized into a sequence of node indices, e.g., if the number of nodes is 5, EasyHypergraph generates a sequence of nodes [0, 1, 2, 3, 4] by default. EasyHypergraph does not initialize the nodes provided by the user as Python objects, avoiding the node mapping management problem introduced by Python objects and the relatively expensive memory consumption problem. Furthermore, it shows better computation speed and memory efficiency in processing large-scale hypergraph datasets. Secondly, this design can make full use of the vectorization technique to improve the computational efficiency. It not only ensures efficient access to structural information during hypergraph analysis but

Table 2 Function comparison of various hypergraph libraries.

Modules	Features	Functions	EasyHypergraph	HNX	XGI	HGX	DHG
Hypergraph manipulation	Hypergraph property	Incidence matrix	✓	✓	✓	✓	✓
		Adjacency matrix	✓	✓	✓	✓	✓
		Neighbor	✓	✓	✓	✓	✓
		Density	✓	-	✓	-	-
		Diameter	✓	✓	-	-	-
	Hypergraph projection	Line graph expansion	✓	✓	-	✓	-
		Clique expansion	✓	✓	-	✓	✓
		Star expansion	✓	-	-	-	✓
Hypergraph analysis	Node ranking	Hyperdegree	✓	✓	✓	-	✓
		Vector centrality	✓	-	✓	✓	-
		Hypercoreness	✓	-	-	-	-
		Cycle ratio	✓	-	-	-	-
	Hyperedge ranking	S-closeness centrality	✓	✓	-	-	-
		S-betweenness centrality	✓	✓	-	-	-
	Connectivity	Connected components	✓	✓	✓	✓	-
		Distance	✓	✓	✓	-	-
		Cluster coefficient	✓	✓	✓	-	-
	Null model	K-uniform model	✓	✓	✓	✓	-
		Chung-Lu model	✓	✓	✓	-	-
Hypergraph learning	HNNs	HGNN	✓	-	-	-	✓
		HGNN$^+$	✓	-	-	-	✓
		HNHN	✓	-	-	-	✓
		HyperGCN	✓	-	-	-	✓
		UniGNN	✓	-	-	-	✓
		DHNE	✓	-	-	-	-
		AllDeepSet	✓	-	-	-	-
		AllSetTransformer	✓	-	-	-	-

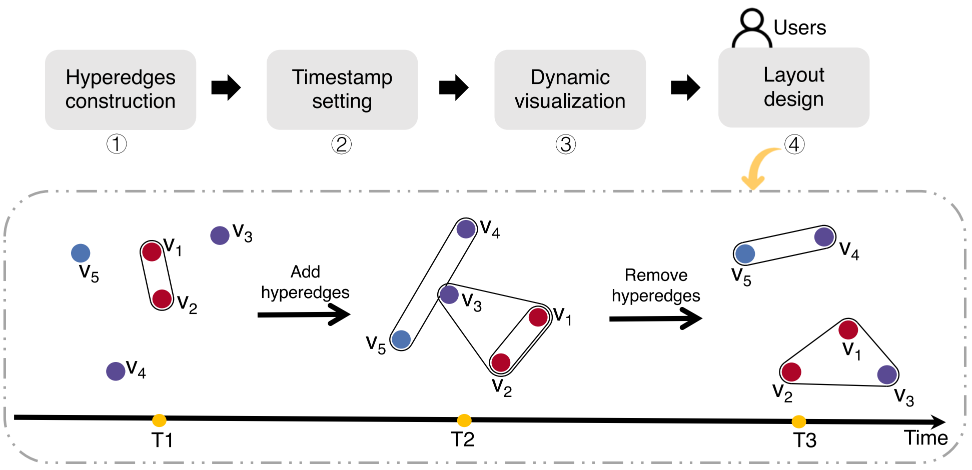


Fig. 2 The use case and an example of dynamic visualization. EasyHypergraph allows users to set timestamps for each hyperedge and design the layout of figures after applying the dynamic visualization function. In the visualization, nodes with the same color represent nodes of the same type, which can be configured by users.

also provides a more natural support for hypergraph learning without extra data conversion. As shown in Fig. 3(a), a user could construct a hypergraph by indicating the number of nodes and a list of hyperedge tuples. Moreover, there are over ten integrated hypergraph datasets for users to adopt. EasyHypergraph also offers hypergraph save/load functions, and thus a user could save a temporary hypergraph for future studies.

The core design of the computational workflow is depicted in Fig. 3(b–d). In the first stage, we select one hypergraph dataset as the initial input. In the second stage, the construction of hyperedges occupied most of the processing time, and it could be a time-consuming process if we need to handle a large scale of hypergraph data. Aiming at a better computation efficiency and fully maximizing this time, we enable the hyperdegree calculation stage to be fused into the hypergraph construction. This optimization results in a significant enhancement in computational efficiency, approximately halving the execution time from a theoretical standpoint. Then, based on the previous stage results, the structural information, including *hyperedge ids*, *row offset*, and *row pointers*, are preloaded into memory for higher-speed access. All of them are used for sparse matrix construction in the following stage. In the fourth stage, all of the structural information is organized to construct the incidence matrix. Here, we focus on the incidence matrix because it is a crucial representation of the hypergraph, and we regard it as a basic property that prioritizes optimization. Then, the compressed sparse row (CSR) is employed to store the incidence matrix, which is a more efficient sparse format to deal with most of the

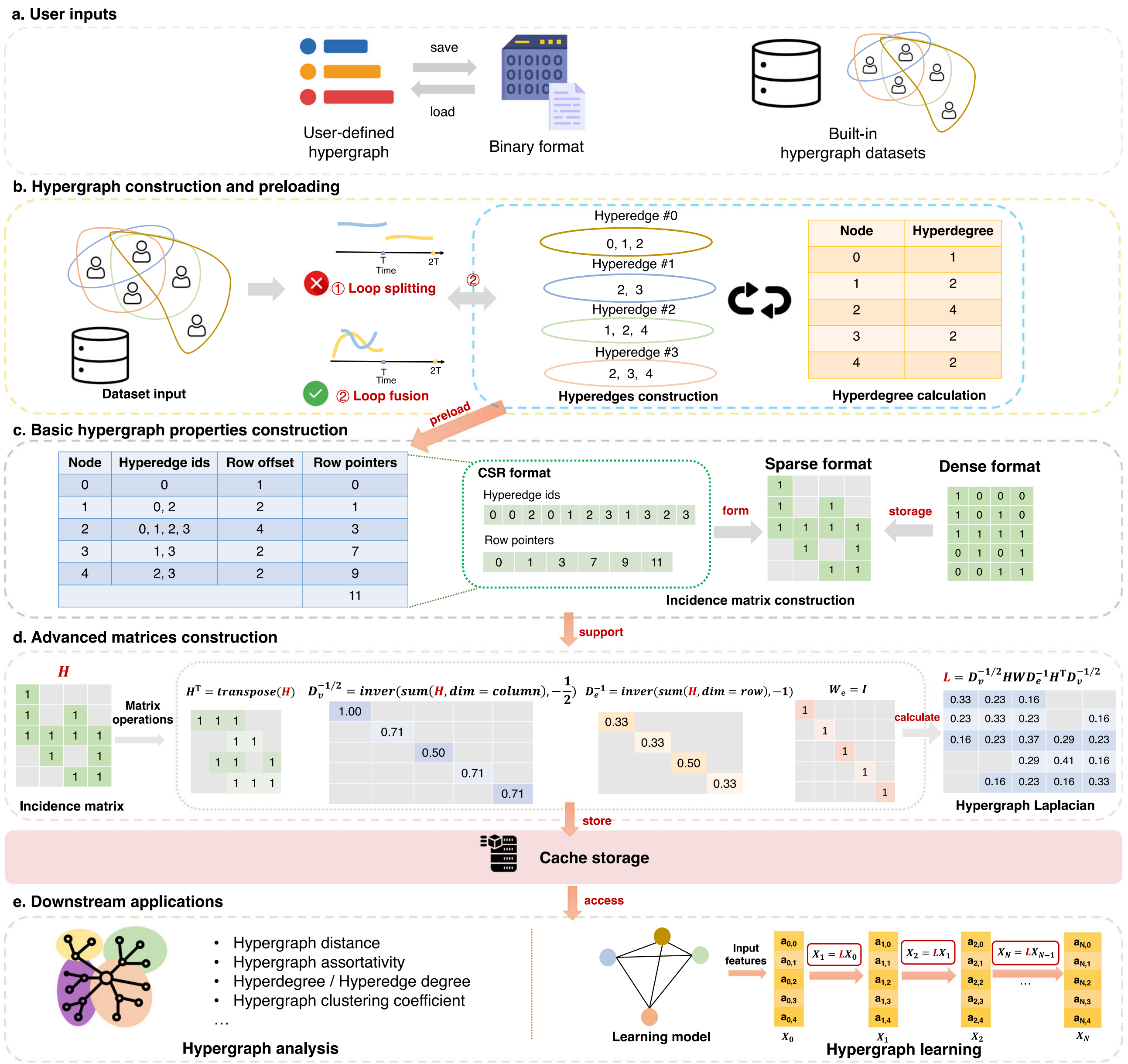


Fig. 3 The computational workflow of EasyHypergraph. a EasyHypergraph allows users to model hypergraphs from the following formats: (i) user-defined nodes and hyperedges, which can be saved to binary format, and recovered into the original hypergraph by the provided load function. (ii) Built-in hypergraph datasets in EasyHypergraph. **b** As soon as the users input a hypergraph dataset, the first stage is to construct hyperedges in sequence, and here the hyperdegree calculation is fused simultaneously during the loop process. **c** Then, the structural information is preloaded into memory for incidence matrix construction. The dense incidence matrix is transformed to CSR sparse format for the following computational optimization. **d** After building up the sparse incidence matrix, several advanced matrices can be inferred from the incidence matrix and used for hypergraph Laplacian calculation. All of them are stored in cache memory for efficient access. **e** Based on the computational results, users can serve two major downstream applications, e.g., in hypergraph analysis, the above matrix can be used for further hypergraph metrics and algorithms. In hypergraph learning, the hypergraph Laplacian matrix is frequently used for multiplication operations.

hypergraphs and will be further explored in the following part. In Fig. 3(d), it can be observed that the incidence matrix serves as a critical common subexpression, facilitating the construction of various advanced properties. Here, the hypergraph Laplacian matrix, a mathematical representation of the hypergraph structure for describing higher-order relationships, is composed of these advanced properties. In addition to this, we store these matrices in the cache storage because they might be frequently used in the downstream tasks, and they can hardly be modified during the computational workflow. Implementing this computational workflow can contribute to the downstream applications shown in Fig. 3(d), such as metrics computation in hypergraph analysis, and Sparse Matrix-Vector Multiplication (SpMV) (Williams et al. 2007; Yesil et al. 2023) in hypergraph learning.

Implementation of the computational workflow. A detailed and technical implementation of the designed computational workflow in Fig. 3 is introduced below.

① Categorization. We categorize properties based on their frequency of usage. We consider properties containing structural information reusable in hypergraph-related computations, and here we call them basic properties. These basic properties (generally in the form of a matrix) are low-level routines for further hypergraph-related computation. Then, we call properties that need further computation based on basic properties as advanced properties. The most common advanced properties include the transposing of the matrix, the inverse matrix, and the generation of a new weight matrix using existing structural information.

② Preloading. We preload the invariant structural information of the basic/advanced properties. At this point, we preload the essential structural details required for constructing the matrix, such as column IDs, row pointers, and hyperdegree, which will be further used in the next step. Specifically, the column IDs represent an entry in the values and store the column index of that element, and the row pointers array stores the index of the first non-zero element of each row.

③ Matrix construction. We construct the corresponding matrix using the structural information preloaded in the previous step. The format of sparse storage has a significant impact on performance by addressing the problem of excess memory and computational resource consumption caused by continuous zeros in the matrix. Therefore, choosing the right sparse storage format would be helpful for the performance optimization of hypergraph analysis or hypergraph learning. While there are various sparse storage methods like the coordinate format (COO), row-compressed format (CSR), and column-compressed format (CSC) (Paszke et al. 2019; Virtanen et al. 2020), in this work, we focus on the COO and CSR formats. The COO format is based on triples where only non-zero elements and their corresponding indices are saved, whereas the CSR format includes compressed row indices. The CSR format typically provides improved storage efficiency and much faster calculations in contrast to the COO format in the scenario of higher row sparsity (PyTorch, 2024; Zhao et al. 2018). As stated in the PyTorch documentation (PyTorch, 2024), employing the CSR format for a 10,000 x 10,000 tensor containing 100,000 non-zero 32-bit floating point values leads to memory savings of 1.6 times compared to the usage of the COO format. Compared with dense formats, the CSR format can be reduced by 310 times in memory usage. Inspired by the inherent sparsity of hypergraphs, we primarily employ the CSR format while maintaining the flexibility to adopt the COO format when more suitable. For specific operations like matrix element access, we prioritize COO utilization. This format selection enables efficient generation of advanced hypergraph properties, including adjacency matrices, weight matrices, and hypergraph Laplacian matrices.

④ Cache storage. The created matrix is stored in a cache for future use. EasyHypergraph utilizes a state container to store properties that are infrequently altered to quickly access the basic/advanced properties mentioned above, thus avoiding unnecessary recalculations.

The primary advantage of our computational workflow is its full utilization of the underlying structural information of the hypergraph. It does not rely on a specific algorithm design but rather optimizes the underlying structure required for most hypergraph computations. When more hypergraph metrics and HNNs need to be implemented in the future, developers are only required to extract the basic structural information within metrics or HNNs and preload or cache them for possible needs or more complicated operations. Therefore, this computational workflow is generic across hypergraph analysis and hypergraph learning, facilitating their comprehensive improvement.

Results

In this section, we demonstrate the effectiveness of EasyHypergraph by applying a quantitative analysis. Secondly, we conduct several experiments for hypergraph analysis and learning on five typical hypergraph datasets to evaluate the computational performance of EasyHypergraph. Meanwhile, the efficiency of the key components in the computational workflow is evaluated during hypergraph analysis and learning. The hyperparameters adaptivity during hypergraph learning is also assessed. Finally, two case studies are presented to showcase the potential of Easyhypergraph in hypergraph analysis and hypergraph learning tasks.

Description of empirical hypergraph datasets. The hypergraph datasets utilized in our experiments are sourced from real-world data, with sizes varying from thousands to hundreds of thousands. The detailed statistics of these datasets can be found in Table 3, with their descriptions provided below:

- PubMed (co-citation) (Yadati et al. 2019) is a dataset derived from the PubMed database, which is widely used to capture the higher-order co-citation relationships between scientific papers. It consists of scientific papers with their co-citation relationships and contains 19,717 nodes and 7963 hyperedges. A node represents a paper, and a hyperedge represents a paper set that has cited the same articles.
- DBLP (co-authorship) (Yadati et al. 2019) is a dataset derived from the DBLP computer science bibliography, which is widely used to study the patterns of scientific collaboration and evolution of the scientific networks. It consists of researchers with their co-author relationships and contains 41,302 nodes and 22,363 hyperedges. A node represents a researcher, and a hyperedge consists of researchers who collaborate on one paper.
- Yelp (Chien et al. 2022) is a restaurant review network dataset. Yelp is an American merchant review site where users can rate and review merchants. This dataset is often used in studies such as recommender systems and merchant analysis. It contains 50,758 nodes and 679,302 hyperedges, where a node represents a restaurant, and a hyperedge is a collection of restaurants visited by the same user.
- Walmart-trips (Jpernci et al. 2015) is a retail industry-related dataset that records consumer transaction data during shopping at Walmart. This dataset is suitable for

Table 3 Properties of the hypergraph datasets, and we show the number of nodes (No. nodes), the number of hyperedges (No. hyperedges), the average hyperedge size (Avg. hyperedge size), and the average hyperdegree of the nodes (Avg. hyperdegree), respectively.

Networks	No. nodes	No. hyperedges	Avg. hyperedge size	Avg. hyperdegree
PubMed (co-citation)	19,717	7963	4.47	8.79
DBLP (co-authorship)	41,302	22,363	4.61	2.33
Yelp	50,758	679,302	6.74	88.65
Walmart-trips	88,860	65,979	6.70	5.09
Trivago-clicks	172,738	220,971	3.18	4.06

analyzing the shopping behavior of customers and the correlation between products. It contains 88,860 nodes and 65,979 hyperedges. A node represents a product, and a hyperedge consists of a group of products purchased by a user at the same time.

- Trivago-clicks (Chodrow et al. 2021) is a hypergraph dataset derived from Trivago, a popular hotel booking platform. This dataset records the click behavior of users when searching for hotels. It typically covers detailed information about user interactions with the platform and can be used for user behavior analysis and ad click prediction. It contains 172,738 nodes and 220,971 hyperedges. A node stands for a lodging option, while a hyperedge indicates a group of lodgings that were clicked on in the same browsing session.

Experiment settings. All comparison experiments are conducted on two Linux servers, each with an Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz and 31 GB of RAM for the metrics computation. Also, we used one server with an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz and 187GB of RAM for HNNs training. The version of HNX is 2.1.3, XGI is 0.8.2, and DHG is 0.9.4.

We select three typical state-of-the-art baseline libraries for comparison. For hypergraph analysis, we choose HNX and XGI. Also, DHG is used for hypergraph learning. They cover a number of functions which are widely used, and consistent functions for more fair efficiency comparison of hypergraph analysis or hypergraph learning (Gao et al. 2022a; Mancastroppa et al. 2023).

Effectiveness analysis. To further demonstrate the effectiveness of the design of computational workflow, we investigated the implementations of EasyHypergraph, HNX, and XGI. Firstly, all libraries realize the significance of vectorization, and fundamental implementation relies on the NumPy library (Harris et al. 2020). However, compared with HNX and XGI, EasyHypergraph does not initialize the node as a hashable object since it is memory inefficient. Alternatively, Easyhypergraph discretizes the node into a sequence of node indices, which enables us to conduct vector operations and adapt to most scenarios of hypergraph analysis. Secondly, since we identified the incidence matrix as a basic property, its optimization benefits the calculation of advanced properties. Thirdly, the incidence matrix is normally stored in a sparse format because it can alleviate memory pressure to a great extent by only storing non-zero elements. CSR (Compressed Sparse Row) and COO (Coordinate Format) are two common storage formats for sparse matrices. CSR stores non-zero values along with their column indices and row pointers, making it memory-efficient for row operations, while COO stores non-zero values combined with their row and column indices as triples, which is intuitive but less efficient for computations. Here, EasyHypergraph chooses the CSR format as the first option based on the structural characteristics of the incidence matrix, while HNX and XGI overlook it and only consider COO.

Although sparse storage formats can reduce the space required for matrix storage, no work has been done to discuss the optimal storage format for hypergraph computing scenarios. Therefore, we take a data-driven approach to analyze the relevant features of the properties of hypergraphs in order to show convincing evidence for a suitable format and as a complementary explanation of our choice of CSR format in the computational workflow. Precisely, we select five typical hypergraph datasets in Table 3 to analyze the Gini index of row/column sparsity of the incidence matrix shown in Fig. 4. The Gini index is generally used as an indicator of the income disparity of the population, with a value range between 0 and 1. When it is greater than or equal to 0.4, it is generally considered to be a country with a large income disparity between the population of that country. The Gini index on the sparsity of each row/column can reflect the degree of unevenness in the sparsity of the matrix. For instance, the larger Gini index on the sparsity of the rows means that some rows of the matrix contain a large number of non-zero elements while other rows have almost none, which suggests that there is a significant inhomogeneity in the data distribution. In particular, we construct the corresponding hypergraph and compute the incidence matrix. Subsequently, two arrays are calculated separately, the first containing the ratio of non-zero elements in all rows of the incidence matrix, which we refer to here as the row sparsity. The second contains the ratio of non-zero elements in all columns, which we refer to here as the column sparsity. Finally, the Gini index is calculated for each of these two sparsity arrays separately.

From the left of Fig. 4(a–e), we can observe that the Gini index values of row sparsity of the incidence matrices corresponding to the four datasets, i.e., PubMed (co-citation), Yelp, Walmart-trips, and Trivago-clicks, are all over 0.4. It indicates that the differences in the distributions of the number of row elements are large, whereas the Gini index of column sparsity is over 0.4, suggesting that the differences in the distribution of the number of row elements are large. Although the general column sparsity Gini index value is lower, the column sparsity Gini index value of the Yelp dataset exceeds 0.5. It can reflect that the variance of the hyperedge sizes of this dataset is larger, and so is the difference in the number of nonzero elements in each column. On the contrary, the row/column sparsity of the DBLP (co-authorship) dataset does not show obvious inhomogeneity. The Gini index value of column sparsity in this dataset is larger than the Gini index value of row sparsity, which indicates that the number of non-zero elements in the columns of this hypergraph incidence matrix is more uneven.

According to the above analysis, most of the datasets exhibit uneven row sparsity. Considering that hypergraph learning involves a large number of matrix multiplication operations, i.e., there will be more demand for row operations. Thus, the CSR format is more suitable for such row operation-intensive scenarios than the COO format. Not only that, the use of CSR format as a storage format facilitates the full use of the locality principle in computers to optimize the memory access pattern (Li et al. 2013). This enables fast access to the non-zero elements of a row when performing row-based operations in hypergraph learning, reducing the extra overhead caused by cache misses.

Moreover, given the essential information required by the CSR format (Zhao et al. 2018), it is crucial to consider basic storage budgets. According to the definition of CSR format, m represents the number of nodes, n represents the number of hyperedges, and nnz indicates the number of non-zero elements (Kanakagiri and Solomonik, 2024; Ramamoorthy et al. 2024). The CSR's capabilities can be maximized when the inequality (6) holds:

$$\begin{aligned} nnz + nnz + m + 1 &< m \times n, \\ \Rightarrow 2 \times nnz &< m \times n - m - 1, \\ \Rightarrow 2 \times nnz &< m \times (n-1) - 1, \\ \Rightarrow nnz &< \tfrac{m \times (n-1) - 1}{2}. \end{aligned} \tag{6}$$

The smaller the nnz is in the inequality, the more storage memory the CSR format can save when storing the original matrix.

Then, inspired by the aforementioned results, we further select the same five empirical hypergraph datasets and depict their distribution of hyperdegree. The results are shown in the right part of Fig. 4(a–e). The y-axis indicates the frequency of the occurrence of each hyperdegree value. These observations imply that most nodes have a relatively low degree, resulting in relatively few non-zero elements in the incidence matrix.

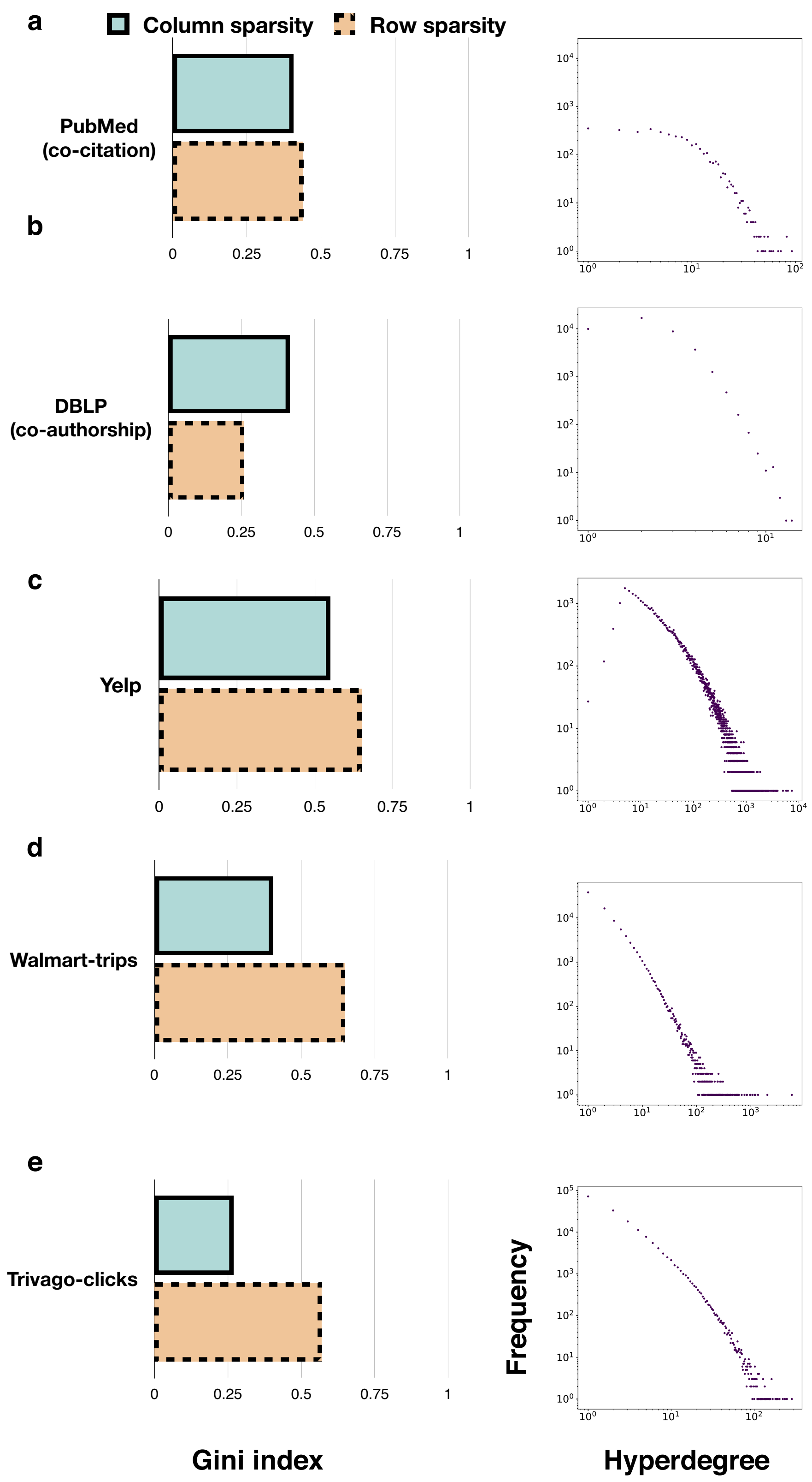


Fig. 4 The Gini index of row/column sparsity and the hyperdegree distribution. The row/column sparsity is calculated by the number of non-zero elements in the row/column divided by the row/column size. Each purple point represents the frequency at which the corresponding hyperdegree value occurs. **a–e** The left part indicates the Gini index of row sparsity and column sparsity for the incidence matrix of each hypergraph dataset. The right part shows the hyperdegree distribution of five empirical hypergraph datasets.

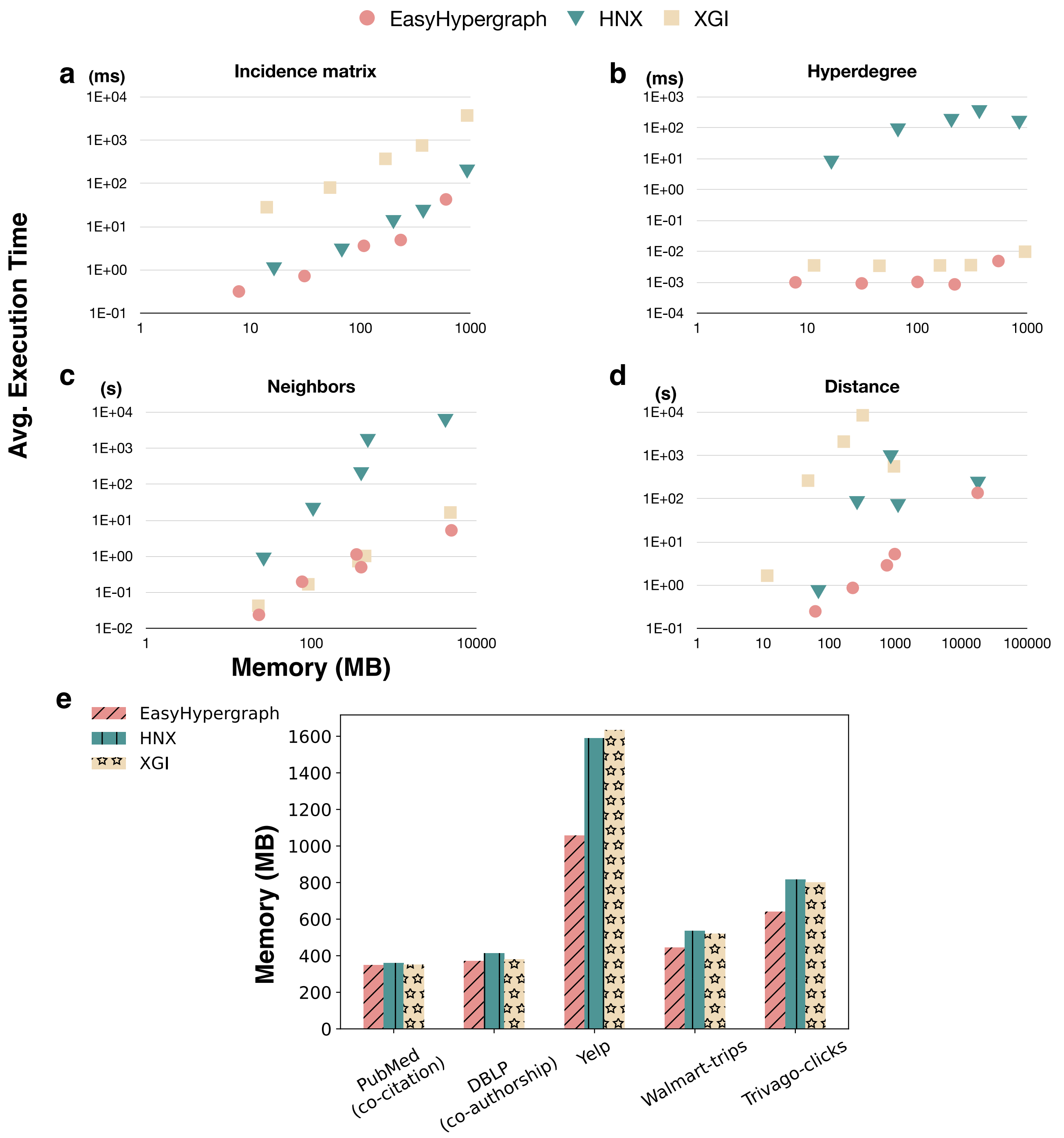


Fig. 5 Comparisons for metrics computational efficiency and memory utilization on four hypergraph analysis metrics. a–d Shows the comparison for metrics computational efficiency and memory utilization on incidence matrix, hyperdegree, neighbor, and distance, respectively. **e** Demonstrates the comparison for maximal memory consumption of each library for computing the incidence matrix.

However, we also notice that the hyperdegree distribution of the Yelp dataset is denser in the middle range, which might degrade the performance brought from the CSR format. To overcome this problem, we adopt a heuristic method by fusing CSR and COO formats into our implementation of hypergraph learning, which enables us to effectively combine the flexibility of COO with the computational efficiency of CSR.

Performance evaluation for hypergraph analysis. The compared metrics include the incidence matrix, hyperdegree, node neighbors, and distance, respectively. Figure 5(a–d) shows the comparison of both time and memory cost of metrics implemented by EasyHypergraph and other libraries among different datasets. The results demonstrate that EasyHypergraph always outperforms HNX and XGI in computational efficiency and memory utilization when calculating the four metrics.

In terms of computational efficiency, the speedup ratios for the incidence matrix range from 2.64 to 153.16. Furthermore, it significantly outperforms HNX and XGI when calculating hyperdegree and distance, with speedup ratios reaching up to several orders of magnitude (e.g., tens of thousands) in optimal cases. This performance advantage becomes even more pronounced for large-scale hypergraphs. This performance gap can be attributed to the metrics calculation algorithms designed by XGI and HNX. The execution time of their designed algorithms is linearly proportional to the hypergraph size. There, when the scale of the input hypergraph exceeds ten thousand, the performance of the algorithms will be significantly reduced. However, the proposed preloading and caching mechanisms in our optimization workflow effectively address this issue. The only exception is the computation of neighbors, where XGI outperforms our library by 0.03s and 0.11s for DBLP (co-authorship) and Trivago-clicks, respectively. A potential reason for this is that since EasyHypergraph computes neighbors through the hypergraph adjacency matrix, it will take more time to compute some nodes that have more neighbors.

In terms of memory utilization, as shown in Fig. 5(a–d), EasyHypergraph reduces memory usage by 46.1% in the best case when calculating the incidence matrix, and by 50.8% in the best case when calculating the hyperdegree. Since the memory usage during the metrics computation is dynamic, here we also take the incidence matrix as an example. Figure 5(e) shows the maximum memory consumption during the process of the incidence matrix computation between EasyHypergraph and the other baseline libraries. The results show that compared to the other libraries, EasyHypergraph saves 25.3% of memory usage in the best scenario while saving an average of 13.5% of memory space across all datasets. Generally, HNX and XGI consume more memory than EasyHypergraph, as EasyHypergraph discretizes nodes into integers, while HNX and XGI store them as more complex Python objects. Besides, we also notice that EasyHypergraph's memory consumption rises gently with the increase of the dataset, which indicates that it has a more obvious advantage in terms of scalability. Finally, we discover that EasyHypergraph costs more memory than XGI when calculating distances in Fig. 5(d). When calculating distances, EasyHypergraph will conduct clique expansion on the hypergraph and transform it into a simple graph to accelerate the calculation process, which will not affect the results. In the worst case, this algorithm will consume 17.5% more memory space than XGI. However, in the process of calculating distance metrics, XGI retrieves the nearest node with a linear time complexity. In contrast, we employ a more efficient min-heap structure to reduce the time complexity to the logarithmic level, with acceleration as fast as 4.09 times.

Finally, we further demonstrate the efficiency of each part in our computational workflow for hypergraph analysis. The demonstration is carried out by taking the calculation of the incidence matrix as an example in Fig. 7(a). We can see that if we remove the cache part or the preloading part, the performance degrades. The reason behind this occurrence is that we preload the edge index while constructing the hypergraph, and also the index of the initial non-zero element in every row. Therefore, the time spent in constructing the incidence matrix has notably decreased. Compared with the traditional approach of calculating the incidence matrix by repeating double nested loops, our computational workflow minimizes the repetitive computation of the sparse matrix by designing cache storage.

Performance evaluation for hypergraph learning. To evaluate the benefits of the computational workflow in hypergraph learning, we have chosen six representative HNNs, i.e., HGNN, HGNN$^+$, HNHN, HyperGCN, UniGCN, and UniGAT, that have demonstrated their effectiveness in previous studies. The results in Fig. 6 show that we are better than the DHG with a notable increase in most hypergraphs. As the scale of datasets increases, we could achieve a greater enhancement. The time required for the HGNN model training can save up to about 70.37% of computation time compared with DHG on the Trivago-clicks dataset. More data can be found in Supplementary Table S1 and Supplementary Table S2. The main deficiency of DHG ignores the features of the intermediate matrices during hypergraph computation, only considering the COO sparse storage format. Nevertheless, EasyHypergraph not only explores these imperceptible features but also proposes to prioritize the CSR format and fuse the flexibility of the COO format.

Subsequently, we further investigate the COO and demonstrate the superiority of CSR over COO. Firstly, we construct the incidence matrix in different storage formats. Figure 7(b) illustrates the improvement in maximal memory savings. Specifically, compared with the dense format, the sparse format mitigates out-of-memory issues and maximizes the utilization of limited memory resources. Compared with the COO format, the CSR format can achieve a better memory saving, about 2.8% at most. Then, we conduct a similar comparison during the hypergraph learning task. The experimental results are shown in Fig. 7(c–h). All of these HNNs gain acceleration on training time, and the superiority of CSR becomes evident during the training of these HNNs. In addition, we observe that HGNN exhibits a more notable improvement than other HNNs, because HGNN depends on more SpMV operations, which means it is more sensitive to the design of the matrix sparse storage strategy. More data can be found in Supplementary Table S3 and Supplementary Table S4. Furthermore, we would like to discuss the impact of hyperparameters on HGNN training. We can see in Fig. 7(i–n) that our computational workflow on HGNN does not introduce additional computational fluctuation and maintains a stable performance under different hyperparameter settings. However, DHG shows a notable growth trend as the size of the hidden layer increases. The main challenge lies in the positive correlation between the hidden size and the number of matrix rows in the neural network, combined with the fact that the COO format used by DHG does not support efficient random row access.

Case studies. To demonstrate the usefulness of EasyHypergraph in social science, we conduct two case studies combined with hypergraph analysis and learning, which is shown in Fig. 8. For hypergraph analysis, we employ two congressional committee datasets, the House-committees dataset (1290 nodes, 341 hyperedges) and Senate-committees dataset (282 nodes, 315 hyperedges), which can be used to study political collaboration patterns (Chodrow et al. 2021; Mancastroppa et al. 2023). Each member in these datasets comes from the US House of Representatives or the US Senate and is represented as a node. At the same time, hyperedges are formed by members who jointly serve on the same congressional committee during the relevant congressional term (103rd–114th Congresses, 1993–2017) (Chodrow et al. 2021). Besides, all members have labels indicating the party to which they belong. The ratio of Democrats to Republicans in House-committees is 620:670, while the ratio in Senate-committees is 140:142.

In Fig. 8(a–f), we adopt metrics or algorithms from EasyHypergraph's hypergraph analysis module, including hyperdegree, hyperedge degree, cycle ratio, hypercoreness, s-closeness, s-betweenness, and the Chung-Lu generative model. By analyzing the results, we have the following observations:

- In Fig. 8(a, b), we discover that nodes belong to more committees in the Senate of Representatives than in the House of Representatives by observing the cumulative frequency of hyperdegree. Meanwhile, compared with the Senate committee, the number of members in each House committee is larger. By combining these two indicators, we

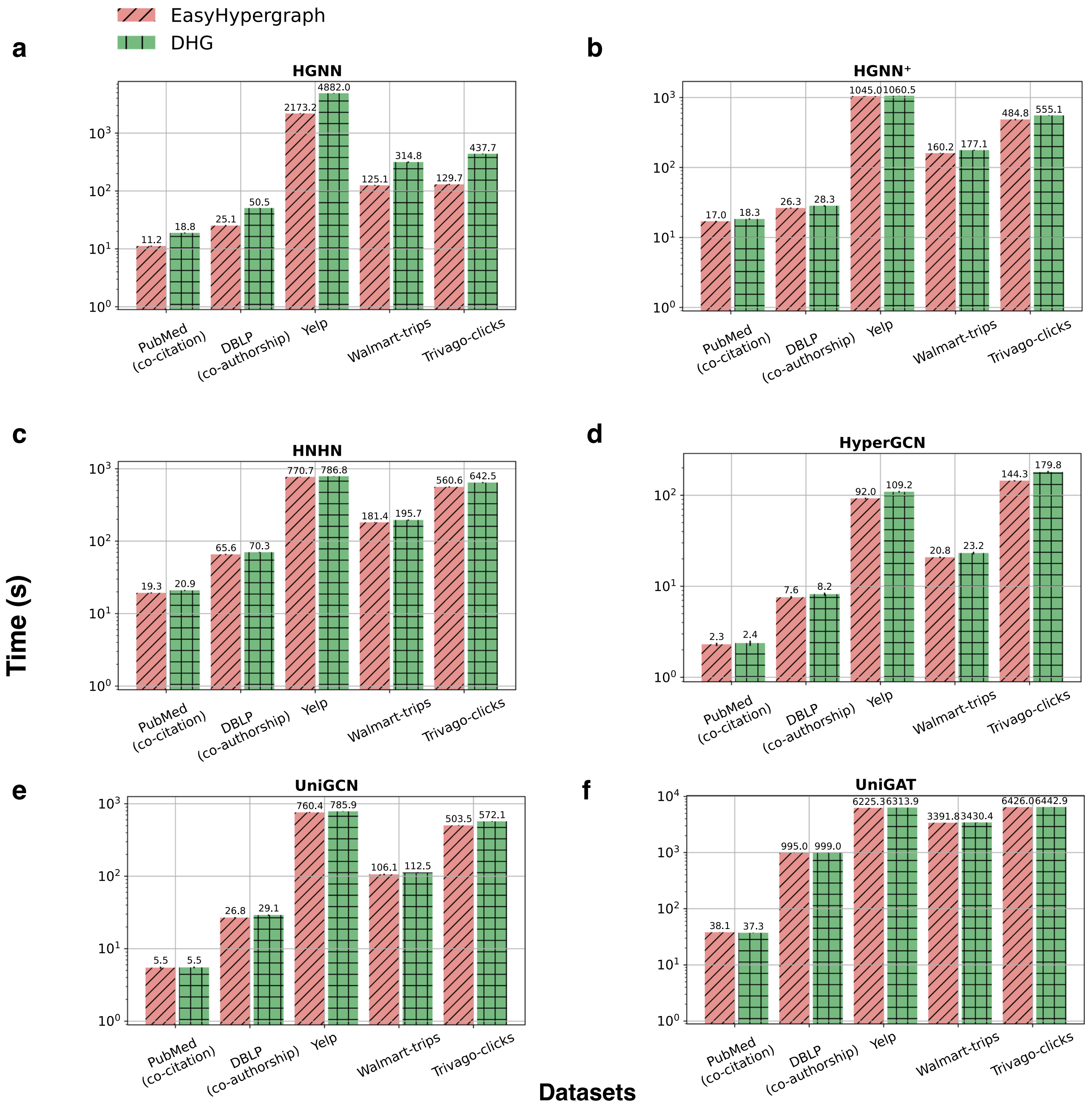


Fig. 6 Comparisons of hypergraph learning efficiency on different hypergraph datasets. The average training time (s) of 100 epochs of **a** HGNN, **b** HGNN$^+$, **c** HNHN, **d** HyperGCN, **e** UniGCN, **f** UniGAT.

can deduce that the participation of members in Senate committees may be more diverse, and each committee has smaller gaps in the number of people, while the House committees show the opposite phenomenon. The potential reasons might be due to the operation mechanism and missions of the two kinds of committees.

- In Fig. 8(c, d), we apply two node ranking metrics (including cycle ratio and hypercoreness) to quantify the node influence in the two datasets. First, the two datasets show similar trends in cycle ratios, which implies that their committee compositions are structurally similar to some extent. Then, we calculate the percentage of members of both parties by filtering out nodes with cycle ratios below the average. The results show that at this point, members of both parties account for a similar percentage of the two datasets, but Republican members have a higher chance of occupying a more important position in the House committees. Second, using a scatterplot, we examine the distribution of node hypercoreness values and observe that Republican legislators' hypercoreness values cluster in two distinct intervals: [0.9, 0.7] and [0.6, 0.2]. Although the nodes of the Democratic Party show similar clusters, their nodes below 0.2 are more than the Republican Party. This phenomenon also confirms the discovery in cycle ratio comparison, that more influential Republican members are in each committee.
- In Fig. 8(e), we preserve different interaction sizes (hyperedges with at least a certain number of nodes) to show the Spearman correlation coefficient of s-closeness and s-betweenness of two datasets. For the sake of brevity,

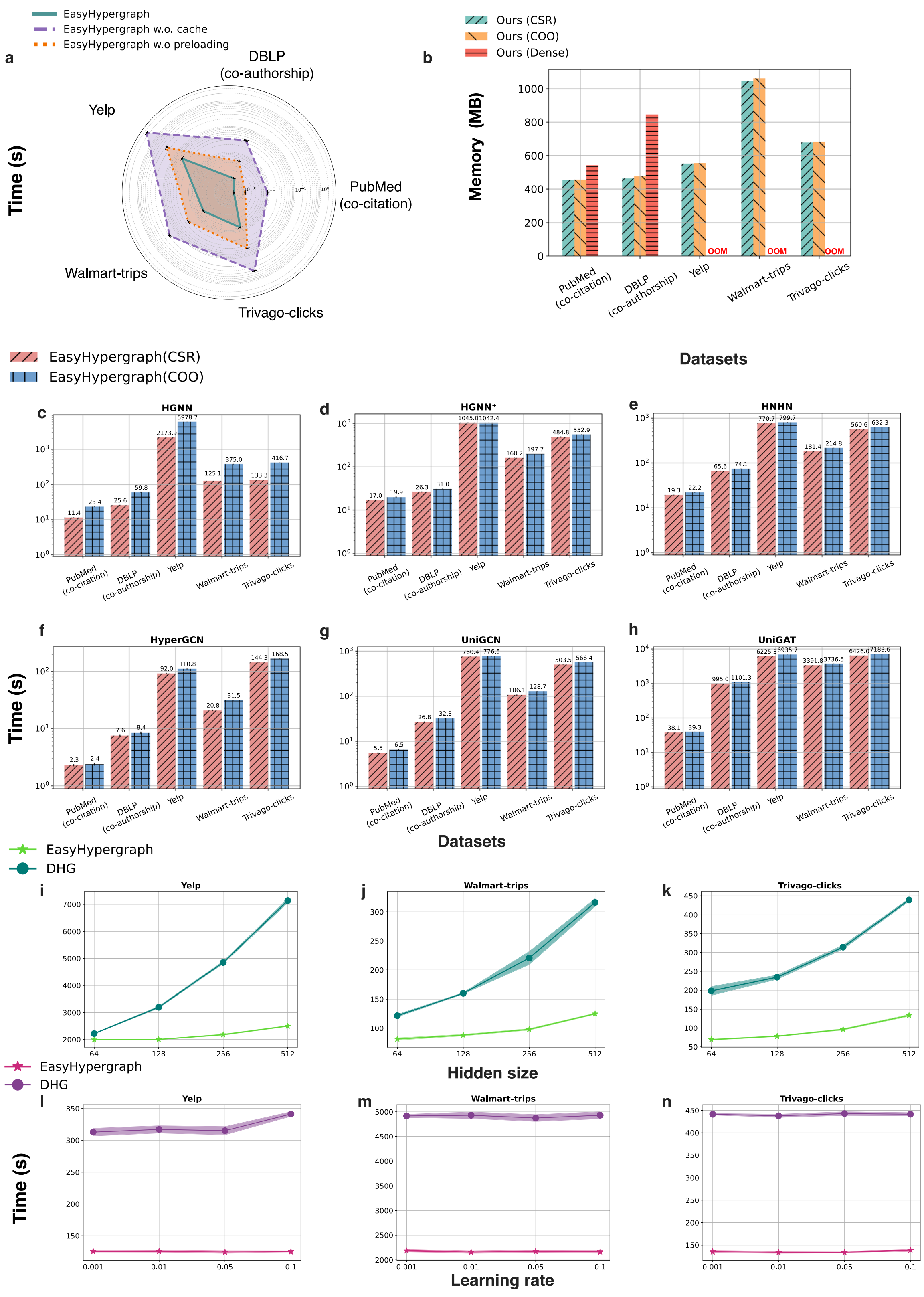
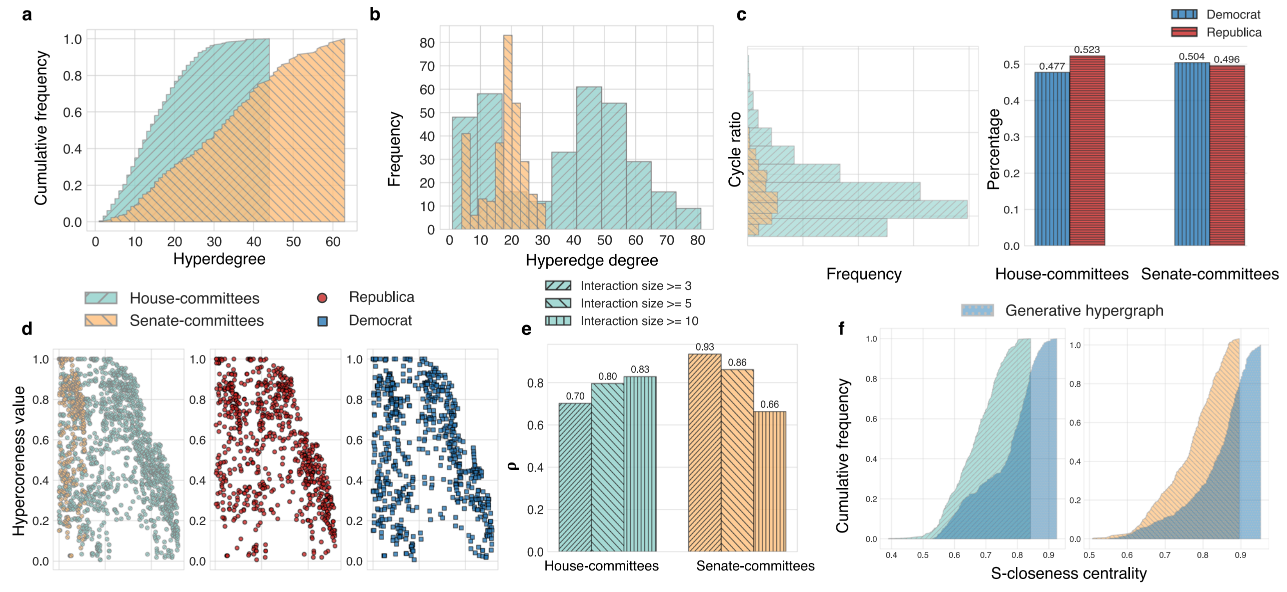
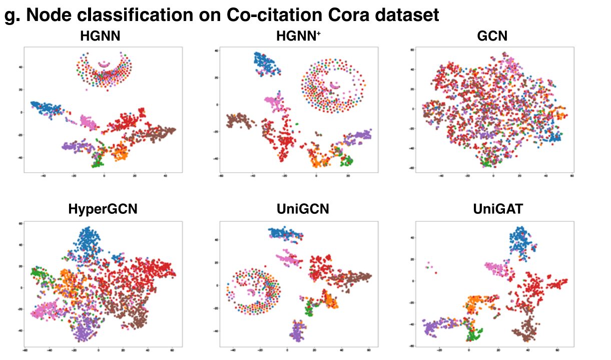
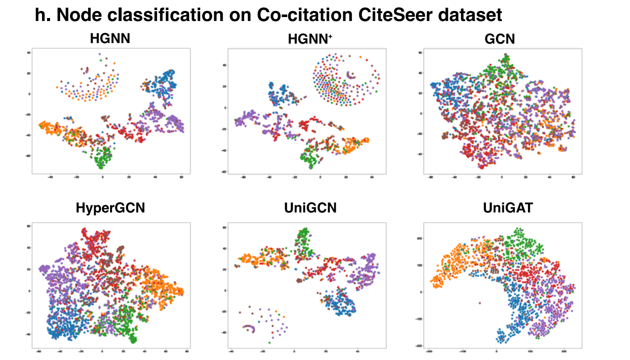


Fig. 7 Ablation study and hyperparameters adaptability experiments for the computational workflow. a Radar chart for comparing time consumption when excluding the cache and preloading, respectively. **b** Comparison of maximal memory usage with different storage formats for computing the incidence matrix. “OOM” indicates out-of-memory situations. **c–h** Comparison of average time with two sparse storage options on different HNNs: HGNN, HGNN$^+$, HNHN, HyperGCN, UniGCN, and UniGAT. **i–n** Hidden size and learning rate are selected as two representative hyperparameters for comparison. Here we train HGNN on various hyperparameter settings. **i–k** The hidden size is tuned from 64 to 512 on corresponding datasets. For fair comparisons, we use the same learning rate of 0.001 for all the models. **l–n** The learning rate is tuned from 0.001 to 0.1 on the corresponding datasets. We use the same hidden sizes, which are also used in hypergraph learning efficiency comparisons for all the models.





Models	GCN	HGNN	HGNN+	HyperGCN	UniGCN	UniGAT
Accuracy	0.249	0.465	0.462	0.649	0.458	0.466
Precision	0.336	0.695	0.684	0.680	0.674	0.686
Recall	0.263	0.465	0.462	0.649	0.458	0.466



Models	GCN	HGNN	HGNN+	HyperGCN	UniGCN	UniGAT
Accuracy	0.403	0.424	0.421	0.615	0.411	0.416
Precision	0.463	0.572	0.563	0.637	0.543	0.561
Recall	0.420	0.424	0.421	0.615	0.411	0.416

Fig. 8 Two case studies for hypergraph analysis and hypergraph learning on real-world datasets. a–f Hypergraph analysis on House-committees and Senate-committees datasets. **a** The cumulative frequency of hyperdegree. **b** The frequency of hyperedge degree. **c** The frequency of cycle ratio and the percentage of members from Democrat or Republican beyond the average cycle ratio. **d** The distributions of hypercoreness value and corresponding political party affiliation of members. **e** The Spearman correlation coefficient between s-closeness and s-betweenness across different interaction sizes. **f** The cumulative frequency of s-closeness ($s = 1$) of original hypergraph and generative hypergraph. The generative hypergraph is generated by the Chung-Lu model. **g**, **h** Node classification on co-citation Cora and co-citation CiteSeer datasets by training hypergraph learning models.

we will refer to this as ρ. Although s-closeness and s-betweenness are both used to rank hyperedges, they focus on global importance and local importance, respectively. In particular, s-closeness reflects the distance between the target hyperedge and all other hyperedges, while s-betweenness calculates how many times that target hyperedge serves as the bridge. We notice that ρ shows an opposite trend on the two datasets. This phenomenon indicates that as the scale of interaction size increases, the hypergraph of House committees may exhibit a structure with more hub hyperedges and bridge hyperedges, while the hypergraph of Senate committees may separate into more connected communities or become more compact.

- As shown in Fig. 8(f), by specifying the node degrees and hyperedge degrees, the Chung-Lu model we implemented can generate a hypergraph that shows a similar s-closeness centrality distribution to the original hypergraph. Researchers are facilitated to characterize topological statistics by treating them as a reference (Zeng et al. 2023). In the future, researchers can also consider it as a benchmark and evaluate their proposed hypergraph null model.

For hypergraph learning, we employ two widely used co-citation datasets, including Cora (2708 nodes, 1579 hyperedges) (Yadati et al. 2019) and CiteSeer (3312 nodes, 1079 hyperedges) (Sen et al. 2008). Cora consists of 2708 scientific publications classified into one of seven fields, while CiteSeer consists of 3312 scientific publications classified into one of six. The co-citation hyperedge is formed between several papers when another article cites them simultaneously. Here, the primary task is to conduct multi-class node classification. We can further compare their learning performance by training a graph convolution network (GCN) and our implemented HNNs. We set the hidden size as 64, the learning rate as 0.01, and the weight decay as 0.0005. After 200 rounds of training, we visualize the low-dimensional node embedding by the t-SNE algorithm (Van der Maaten and Hinton, 2008) and show their training performance in Fig. 8(g, h). As shown in the results, the node representations of HNNs exhibit more distinct classification and clearer class boundaries after visualization. Meanwhile, most HNNs surpass GCN in accuracy, precision, and recall, with precision showing the most significant improvement of up to 2%. Such improvement also demonstrates the effectiveness of HNNs when performing node classification tasks, and users might unlock their potential in the broader social sciences.

In summary, EasyHypergraph is able to complement the deficiency of simple graphs in modeling higher-order relationships. Moreover, EasyHypergraph could help social science researchers propose new research questions and explore the deeper triggers of certain patterns in higher-order networks. Gaining insight from the target network is also critical for researchers and practitioners to predict objective properties or behaviors by HNNs, such as predicting market behavior (Batrancea et al. 2024; Ma et al. 2022; Sawhney et al. 2020) and user behavior in social networks (Guo et al. 2025; Han et al. 2023).

Discussion

This paper studies the design and optimization of a hypergraph computation library for researchers and practitioners from multiple disciplines. Although several libraries have tried to meet the users' need for hypergraph analysis or learning, these libraries failed to support them in a unified framework and lack optimization for computation speed or memory consumption. More importantly, the existing research gaps cause unnecessary costs in terms of time, resources, and development. To fill existing research gaps, this paper proposes a unified hypergraph computation library named EasyHypergraph for both of them, as well as an efficient computational workflow.

Although the existing libraries have a certain degree of user-friendliness in the construction of hypergraphs, they also have the problems of tedious design and insufficient performance. First, in the existing hypergraph analysis libraries, the code design at the node level or the hyperedge level involves more complex nesting within the hypergraph class. It reduces the readability of the code, making developers difficult to track the flow of data in the process of hypergraph construction or metrics computation. Second, the process of constructing a hypergraph generates more intermediate results or Python objects in memory, which generates memory overheads that will be overloaded by the general computational resources as the size of the hypergraph increases. Finally, existing hypergraph analysis libraries do not fully utilize the structural information of the hypergraph, which generates a large number of redundant computations in the process of hypergraph metrics calculation and reduces the efficiency of hypergraph-related computation. EasyHypergraph eliminates the management of original node mapping by discretizing the node into a sequence of continuous node indexes. Therefore, Easyhypergraph not only ensures the availability of hypergraph analysis but also meets the need of hypergraph learning since it is compatible with vector operations. Based on the hypergraph class, users can manipulate hypergraph structure with different granularity and utilize the functions in modules, including hypergraph analysis, hypergraph learning, and hypergraph visualization.

To accelerate the hypergraph analysis and hypergraph learning and increase their memory utilization, this paper designs and implements an efficient computational workflow. Firstly, we divide the hypergraph properties into basic properties (e.g., incidence matrix and hyperdegree) and advanced properties (e.g., adjacency matrix and hypergraph Laplacian matrix). They are interconnecting properties and strongly related to the computation of the underlying hypergraph structure, which indicates that optimization on them will bring a general improvement for more complicated metrics computation or model training. To implement the optimization, we develop three key mechanisms: loop fusion and preloading for basic properties and cache for both basic properties and advanced properties, each contributing to the enhancement of computational efficiency. Additionally, this paper utilizes five representative hypergraph datasets and analyzes them from two perspectives, namely, sparsity and power law characteristics. Firstly, we define the sparsity of each row and column of the incidence matrix and calculate the Gini index of the two indicators. We observe that the row sparsity of most of the incidence matrices is uneven, while the Gini index of the column sparsity is lower, which reflects that most of the incidence matrices have larger differences in the distributions of the rows. Therefore, combining the above analysis with the background that the process of hypergraph learning involves a large number of matrix multiplication operations, we believe that it is suitable for the preferred CSR sparse storage format. However, the CSR format might be affected by the layout of non-zero elements, which inspires us to develop a heuristic sparse storage format selection strategy. This strategy combines the effectiveness of the CSR format and the flexibility of the COO format, improving the overall memory efficiency while maintaining computational performance. By conducting experiments on the above five typical hypergraph datasets and comparing them with three state-of-the-art hypergraph computation libraries, EasyHypergraph outperforms its baselines in terms of the compared metrics. The average speedup ratio on all calculations could reach 11,155. For memory usage, EasyHypergraph saves memory by up to 54.28% and 44.17% for the incidence matrix. Meanwhile, our implementation of the HGNN model achieves up to a 70.37% reduction in training time compared with DHG when the data size exceeds hundreds of thousands.

Limitations and future works. Despite offering a wider range of functions and being more efficient than existing hypergraph libraries, EasyHypergraph still has some limitations. Firstly, motif detection algorithms on simple graphs are mature, but there is no unified definition of motif detection on hypergraphs in academia. Currently, EasyHypergraph supports projecting hypergraphs into simple graphs and utilizing the algorithms in EasyGraph for detection. However, this approach might be a barrier to capturing higher-order interaction patterns in the hypergraph, limiting the depth of hypergraph analysis. A similar situation occurs in the process of using community discovery algorithms, indicating that our support for some important hypergraph analysis algorithms still needs to be improved. Secondly, although real-world networks (e.g., social networks) are mostly highly sparse, the need for users who might deal with dense matrices cannot be ignored. Currently, EasyHypergraph has not fully considered the management of dense matrices. In dense matrices, due to the large number of non-zero elements in each row, the index storing and searching overhead of the CSR format increases significantly, resulting in performance degradation and affecting the user experience. Thirdly, the current developer-dependent division of basic and advanced properties in computational workflows may result in sub-optimal hypergraph property computations.

To overcome existing limitations, the first potential future direction is to investigate a method that can minimize the error when we have to project the hypergraph to a graph for target algorithms. Besides, EasyHypergraph will integrate representative motif detection algorithms in the future. Secondly, EasyHypergraph will not only consider dense matrix management but also set a goal to create a machine learning model that can dynamically adopt the best storage format based on the structural features of hypergraphs to further improve the efficiency of hypergraph analysis and hypergraph learning. Thirdly, designing an automated method for identifying basic and advanced properties is one of the feasible ways in future work. For example, a static analysis tool can identify high-frequency code segments in the hypergraph computation, which can help developers make more reasonable optimization decisions.

The future development of EasyHypergraph is aimed at evolving EasyHypergraph into a cross-disciplinary research tool that bridges gaps between hypergraph theory and practice. The first is functional versatility. Given a large number of open problems in the field of higher-order networks, emerging hypergraph algorithms will be integrated into EasyHypergraph in the future, giving users a richer perspective on hypergraph analysis. In the meantime, EasyHypergraph will continue to expand access to hypergraph datasets to include not only social networks but also higher-order networks in other areas such as chemistry and biology. From the perspective of technological innovation, the training time of different HNNs might vary on different types of hardware. To minimize the performance uncertainty introduced by hardware, in the future, deep learning compilers could be considered to convert these models from high-level representations to executable code that can run efficiently on specific hardware. Finally, the documentation and tutorials of EasyHypergraph will be further improved to enhance the user experience and reduce the learning costs.

Conclusion

Higher-order networks are increasingly being studied for its ability to represent relationships among numerous entities (Li et al. 2024b). Hypergraph, as a representative kind of higher-order network, has been used as a powerful tool to reveal higher-order relationships in different disciplines. While the existing hypergraph computation libraries like HNX, XGI, and DHG have facilitated hypergraph-related analysis and learning tasks, they are deficient in functional comprehensiveness and computational efficiency. To this end, this paper designs and implements EasyHypergraph, a unified library that supports hypergraph analysis and hypergraph learning simultaneously and achieves fast speed and efficient memory utilization. By conducting computational experiments, EasyHypergraph demonstrates its fast speed that could reduce the computation time from XGI and HNX by a large margin for hypergraph analysis, with average speedup ratios on all compared metrics calculations reaching 11,155. In addition, for hypergraph learning, EasyHypergraph can save about 70.37% of computation time for HGNN training on datasets with hundreds of thousands of nodes. Finally, EasyHypergraph shows its capability by conducting case studies on four real-world datasets related to political science and co-citation relationships for hypergraph analysis and hypergraph learning, respectively.

Data availability

The datasets used for the current study are available via https://github.com/easy-graph/EasyHypergraph_benchmark/. Source code and package installation instructions can be found in the https://easy-graph.github.io/docs/hypergraph.html.

Received: 15 January 2025; Accepted: 30 May 2025;
Published online: 09 August 2025

References

Agarwal S, Branson K, Belongie S (2006) Higher order learning with graphs. In International Conference on Machine Learning, volume 148, pages 17–24
Aksoy SG, Kolda TG, Pinar A (2017) Measuring and modeling bipartite graphs with community structure. J Complex Netw 5(4):581–603
Aksoy SG, Joslyn C, Ortiz Marrero C, Praggastis B, Purvine E (2020) Hypernetwork science via high-order hypergraph walks. EPJ Data Sci 9(1):16
Antelmi A, Cordasco G, Polato M, Scarano V, Spagnuolo C, Yang D (2023) A survey on hypergraph representation learning. ACM Comput Surveys 56(1):24:1–24:38
Badie-Modiri A, Kivelä M (2023) Reticula: a temporal network and hypergraph analysis software package. SoftwareX 21:101301
Batrancea LM, Akgüller Ö, Balcí MA, Nichita A (2024) Financial network communities and methodological insights: a case study for Borsa Istanbul sustainability index. Humanit Soc Sci Commun 11(1):1046
Berge C (1985) Graphs and Hypergraphs. Elsevier Science Ltd
Çatalyürek ÜV, Boman EG, Devine KD, Bozdag D, Heaphy RT, Riesen LA (2007) Hypergraph-based dynamic load balancing for adaptive scientific computations. In IEEE International Parallel & Distributed Processing Symposium, pages 1–11
Chen L, Xia Y, Zhang Y (2025) Group synchronization of heterogeneous multiagent system with hierarchical structure and beyond pairwise interactions. IEEE Trans Syst Man Cybern Syst 55(3):1578–1590
Cheng K, Cheng X, Wang W (2024) The determinants influencing bilingual instruction in chinese higher education: a complex network analysis. Humanit Soc Sci Commun 11(1):1029
Chien E, Pan C, Peng J, Milenkovic O (2022) You are allset: a multiset function framework for hypergraph neural networks. In International Conference on Learning Representations
Chodrow PS, Veldt N, Benson AR (2021) Generative hypergraph clustering: from blockmodels to modularity. CoRR, abs/2101.09611
Dai Q, Gao Y (2023) Hypergraph Computation. Springer Nature
Dewar M, Pike D, Proos J (2018) Connectivity in hypergraphs. Can Math Bull 61(2):252–271
Dong Y, Sawin W, Bengio Y (2020) HNHN: Hypergraph networks with hyperedge neurons. ICML Graph Representation Learning and Beyond Workshop
Erdös P, Rényi A (1960) On the evolution of random graphs. Publ Math Inst Hung Acad Sci 5(1):17–60
Estrada E, Rodríguez-Velázquez JA (2006a) Subgraph centrality and clustering in complex hyper-networks. Phys A: Stat Mech Appl 364:581–594
Estrada E, Rodríguez-Velázquez JA (2006b) Subgraph centrality and clustering in complex hyper-networks. Phys A: Stat Mech Appl 364:581–594
Fan T, Lü L, Shi D, Zhou T (2021) Characterizing cycle structure in complex networks. Commun Phys 4(1):272
Feng Y, You H, Zhang Z, Ji R, Gao Y (2019) Hypergraph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, pages 3558–3565
Fic M, Gokhale CS (2024) Catalysing cooperation: the power of collective beliefs in structured populations. npj Complex 1(1):6
Gallagher SR, Goldberg DS (2013) Clustering coefficients in protein interaction hypernetworks. In Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics, pages 552–560
Gao M, Li Z, Li R, Cui C, Chen X, Ye B (2023) EasyGraph: a multifunctional, cross-platform, and effective library for interdisciplinary network analysis. Patterns 4(10):100839
Gao Y, Feng Y, Ji S, Ji R (2022a) HGNN+: General hypergraph neural networks. IEEE Trans Pattern Anal Mach Intell 45(3):3181–3199
Gao Y, Zhang Z, Lin H, Zhao X, Du S, Zou C (2022b) Hypergraph learning: Methods and practices. IEEE Trans Pattern Anal Mach Intell 44(5):2548–2566
Guo L, Zhou S, Tang H, Zheng X, Luo Y (2025) Multi-behavior hypergraph contrastive learning for session-based recommendation. IEEE Trans Knowl Data Eng 37(3):1325–1338
Guzzo A, Pugliese A, Rullo A, Saccà D, Piccolo A (2017) Malevolent activity detection with hypergraph-based models. IEEE Trans Knowl Data Eng 29(5):1115–1128
Hagberg A, Swart PJ, Schult DA (2008) Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States)
Han Y, Huang EW, Zheng W, Rao N, Wang Z, Subbian K (2023) Search behavior prediction: a hypergraph perspective. In ACM International Conference on Web Search and Data Mining, page 697–705
Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D (2020) Array programming with numpy. Nature 585(7825):357–362
Huang J, Yang J (2021) UniGNN: a unified framework for graph and hypergraph neural networks. In International Joint Conference on Artificial Intelligence, pages 2563–2569
Jia J, Yang L, Wang Y, Sang A (2025) Hyper attack graph: Constructing a hypergraph for cyber threat intelligence analysis. Comput Security 149:104194
Jpreci Admin WC, Cukierski W (2015) Walmart recruiting: Trip type classification. https://kaggle.com/competitions/walmart-recruiting-trip-type-classification. Kaggle

Juul JL, Benson AR, Kleinberg J (2024) Hypergraph patterns and collaboration structure. Front Phys 11:1301994
Kanakagiri R, Solomonik E (2024) Minimum cost loop nests for contraction of a sparse tensor with a tensor network. In ACM Symposium on Parallelism in Algorithms and Architectures, pages 169–181
Kim K, Kogler DF, Maliphol S(2024a) Identifying interdisciplinary emergence in the science of science: combination of network analysis and bertopic Humanit Soc Sci Commun 11(1):603
Kim S, Lee SY, Gao Y, Antelmi A, Polato M, Shin K (2024b) A survey on hypergraph neural networks: An in-depth and step-by-step guide. In ACM Knowledge Discovery and Data Mining, pages 6534–6544
Klimm F, Deane CM, Reinert G (2021) Hypergraphs for predicting essential genes using multiprotein complex data. J Complex Netw 9(2):cnaa028
Kook Y, Ko J, Shin K (2020) Evolution of real-world hypergraphs: patterns and models without oracles. In IEEE International Conference on Data Mining, pages 272–281
Kovalenko K, Romance M, Vasilyeva E, Aleja D, Criado R, Musatov D (2022) Vector centrality in hypergraphs. Chaos Solitons Fractals 162:112397
Laboratory PNN (2023) HyperNetX. Online
Landaeta-Torres V, Candia C, Pulgar J, Fábrega J, Varela JJ, Yaikin T (2024) Game theory in the classroom: low cooperative relationships identify bullying patterns in elementary schools. Humanit Soc Sci Commun 11(1):1101
Landry NW, Lucas M, Iacopini I, Petri G, Schwarze A, Patania A (2023) XGI: a Python package for higher-order interaction networks. J Open Source Softw 8(85):5162
Lee J, Lee Y, Oh SM, Kahng B (2021) Betweenness centrality of teams in social networks. Chaos: Interdiscip J Nonlinear Sci 31(6):061108
Lerman K, Feldman D, He Z, Rao A (2024) Affective polarization and dynamics of information spread in online networks. npj Complex 1(1):8
Li J, Tan G, Chen M, Sun N (2013) Smat: an input adaptive auto-tuner for sparse matrix-vector multiplication. In ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, pages 117–126
Li P, Zhang P, Guo Y, Li J(2024a) How has the relationship between major financial markets changed during the Russia–Ukraine conflict? Humanit Soc Sci Commun 11(1):1731
Li X, Zhu Q, Zhao C, Duan X, Zhao B, Zhang X(2024b) Higher-order Granger reservoir computing: simultaneously achieving scalable complex structures inference and accurate dynamics prediction Nat Commun 15(1):2506
Lotito QF, Musciotto F, Montresor A, Battiston F (2022) Higher-order motif analysis in hypergraphs. Commun Phys 5(1):79
Lotito QF, Contisciani M, De Bacco C, Di Gaetano L, Gallo L, Montresor A (2023) Hypergraphx: a library for higher-order network analysis. J Complex Netw 11(3):cnad019
Lucas M, Cencetti G, Battiston F (2020) Multiorder Laplacian for synchronization in higher-order networks. Phys Rev Res 2:033410
Lung RI, Gaskó N, Suciu MA (2018) A hypergraph model for representing scientific output. Scientometrics 117:1361–1379
Ma X, Zhao T, Guo Q, Li X, Zhang C (2022) Fuzzy hypergraph network for recommending top-K profitable stocks. Inf Sci 613:239–255
Mancastroppa M, Iacopini I, Petri G, Barrat A (2023) Hyper-cores promote localization and efficient seeding in higher-order processes. Nat Commun 14(1):6223
Newman MEJ (2001) The structure of scientific collaboration networks. Proc Natl Acad Sci 98(2):404–409
Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al (2019) PyTorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems, volume 32
PyTorch (2024) torch.sparse - PyTorch 2.4 documentation. https://pytorch.org/docs/stable/sparse.html. (Accessed on 8/31/2024)
Ramadan EY, Tarafdar A, Pothen A (2004) A hypergraph model for the yeast protein complex network. In IEEE International Parallel & Distributed Processing Symposium, pages 26–30
Ramamoorthy A, Meng R, Girimaji VS (2024) Leveraging partial stragglers within gradient coding. In Advances in Neural Information Processing Systems, volume 37, pages 60382–60402
Rodríguez-Casañ R, Carbó-Catalan E, Solé-Ribalta A, Roig-Sanz D, Borge-Holthoefer J, Cardillo A (2024) Analysing inter-state communication dynamics and roles in the networks of the International Institute of Intellectual Cooperation. Humanit Soc Sci Commun 11(1):1408
Ruhnau B (2000) Eigenvector-centrality-a node-centrality? Soc Netw 22(4):357–365
Sawhney R, Agarwal S, Wadhwa A, Shah RR (2020) Spatiotemporal hypergraph convolution network for stock movement forecasting. In IEEE International Conference on Data Mining, pages 482–491
Schwartz GA (2021) Complex networks reveal emergent interdisciplinary knowledge in Wikipedia. Humanit Soc Sci Commun 8(1):127
Sen P, Namata GM, Bilgic M, Getoor L, Gallagher B, Eliassi-Rad T (2008) Collective classification in network data. AI Mag 29(3):93–106
Su X, Yang J, Wu J, Qiu Z (2025) Hy-defake: Hypergraph neural networks for detecting fake news in online social networks. Neural Netw 187:107302
Tudisco F, Higham DJ (2023) Core-periphery detection in hypergraphs. SIAM J Math Data Sci 5(1):1–21
Van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. Journal of Machine Learning Research, 9(11)
Varley TF (2024) A scalable synergy-first backbone decomposition of higher-order structures in complex systems. npj Complex 1(1):9
Vasilyeva E, Romance M, Samoylenko I, Kovalenko K, Musatov D, Raigorodskii AM (2023) Distances in higher-order networks and the metric structure of hypergraphs. Entropy 25(6):923
Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat Methods 17(3):261–272
Wang S, Bette HM, Schreckenberg M, Guhr T (2024) How much longer do you have to drive than the crow has to fly? npj Complex 1(1):22
Wang Y, Kleinberg J (2024) From graphs to hypergraphs: Hypergraph projection and its reconstruction. In International Conference on Learning Representations
Weber RE, Mueller C, Reinhart C (2024) A hypergraph model shows the carbon reduction potential of effective space use in housing. Nat Commun 15(1):8327
Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, and Demmel J (2007) Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In International Conference for High Performance Computing, Networking, Storage, and Analysis, page 38
Xi X, Gao X, Sun X, Zheng H, Wu C (2024) Dynamic analysis and application of network structure control in risk conduction in the industrial chain. Humanities Soc Sci Commun 11(1):1468
Xia X, Yin H, Yu J, Wang Q, Cui L, Zhang X (2021) Self-supervised hypergraph convolutional networks for session-based recommendation. In Proceedings of the AAAI Conference on Artificial Intelligence, pages 4503–4511
Xu, B., Qiao, X., Lin, H., and Zong, L. (2024). MPHDetect: Multi-view prompting and hypergraph fusion for malevolence detection in dialogues. In ACM International Conference on Information and Knowledge Management, page 4133-4137
Xu C, Li M, Ni Z, Zhang Y, Chen S (2022) GroupNet: Multiscale hypergraph neural networks for trajectory prediction with relational reasoning. In IEEE/CVF Computer Vision and Pattern Recognition Conference, pages 6498–6507
Yadati N, Nimishakavi M, Yadav P, Nitin V, Louis A, Talukdar P (2019) HyperGCN: A new method for training graph convolutional networks on hypergraphs. In Conference on Neural Information Processing Systems, volume 32
Yan Y, Chen Y, Wang S, Wu H, Cai R (2024) Hypergraph joint representation learning for hypervertices and hyperedges via cross expansion. In Proceedings of the AAAI Conference on Artificial Intelligence, number 8, pages 9232–9240
Yang C, Wang R, Yao S, Abdelzaher T (2022) Semi-supervised hypergraph node classification on hypergraph line expansion. In ACM International Conference on Information and Knowledge Management, page 2352-2361
Yesil S, Heidarshenas A, Morrison A, Torrellas J (2023) WISE: Predicting the performance of sparse matrix vector multiplication with machine learning. In ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, page 329–341
Yu J, Yin H, Li J, Wang Q, Hung NQV, Zhang X (2021) Self-supervised multi-channel hypergraph convolutional network for social recommendation. In International World Wide Web Conferences, page 413-424
Zeng Y, Liu B, Zhou F, Lü L (2023) Hyper-null models and their applications. Entropy 25(10):1390
Zhang Y, Latora V, Motter AE (2021) Unified treatment of synchronization patterns in generalized networks with higher-order, multilayer, and temporal interactions. Commun Phys 4(1):195
Zhao Y, Li J, Liao C, Shen X (2018) Bridging the gap between deep learning and sparse matrix format selection. In ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, page 94-108
Zhao Z, Zhou X, Zheng Y, Meng T, Fang D (2024) Enhancing infrastructural dynamic responses to critical residents' needs for urban resilience through machine learning and hypernetwork analysis. Sustain Cities Soc 106:105366
Zhou W, Nakhleh L (2011) Properties of metabolic graphs: biological organization or representation artifacts? BMC Bioinforma 12(1):132

Acknowledgements
This work is sponsored by the National Natural Science Foundation of China (No. 62072115, No. 62102094, No. 72371224), the Shanghai Science and Technology Innovation Action Plan Project (No. 22510713600, No. 24692102500), and the Shanghai Municipal Education Commission Artificial Intelligence Project (No.24RGZN13).

Author contributions

Conceptualization, BY, MG, and YC; software package, BY, and MG; dataset collection and benchmarking, BY and MG; writing–review and editing, BY, MG, XZ, XH, ZZ, QG, XW, and YC; visualization, BY; supervision, YC. All authors read and approved the final manuscript.

Competing interests

The authors declare no competing interests.

Ethical approval

Ethical approval was not necessary for this study since it did not involve human participants. This study only adopts publicly available datasets as its data sources, and they adhere to relevant ethical guidelines and standards. Meanwhile, all data used in this study adhere to the relevant ethical guidelines and standards for the use of publicly available information.

Informed consent

Since no human subjects are involved in this study, informed consent was not required. This study used publicly available datasets only for performance evaluation experiments, which do not require consent.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1057/s41599-025-05180-5.

Correspondence and requests for materials should be addressed to Qingyuan Gong or Yang Chen.

Reprints and permission information is available at http://www.nature.com/reprints