Modularity-based Hypergraph Clustering: Random Hypergraph Model, Hyperedge-cluster Relation, and Computation

ZIJIN FENG, Department of Systems Engineering and Engineering Management, and Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong, Hong Kong
MIAO QIAO, University of Auckland, New Zealand
HONG CHENG, Department of Systems Engineering and Engineering Management, and Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong, Hong Kong

A graph models the connections among objects. One important graph analytical task is clustering which partitions a data graph into clusters with dense innercluster connections. A line of clustering maximizes a function called *modularity*. Modularity-based clustering is widely adopted on dyadic graphs due to its scalability and clustering quality which depends highly on its selection of a *random graph model*. The random graph model decides not only which clustering is preferred – modularity measures the quality of a clustering based on its alignment to the edges of a random graph, but also the cost of computing such an alignment. Existing random hypergraph models either measure the hyperedge-cluster alignment in an All-Or-Nothing (AON) manner, losing important group-wise information, or introduce expensive alignment computation, refraining the clustering from scaling up. This paper proposes a new random hypergraph model called Hyperedge Expansion Model (HEM), a non-AON hypergraph modularity function called Partial Innerclusteredge modularity (PI) based on HEM, a clustering algorithm called Partial Innerclusteredge Clustering (PIC) that optimizes PI, and novel computation optimizations. PIC is a scalable modularity-based hypergraph clustering that can effectively capture the non-AON hyperedge-cluster relation. Our experiments show that PIC outperforms eight state-of-the-art methods on real-world hypergraphs in terms of both clustering quality and scalability and is up to five orders of magnitude faster than the baseline methods.

CCS Concepts: • **Information systems** → **Clustering**; • **Mathematics of computing** → **Hypergraphs**; • **Computing methodologies** → *Cluster analysis*.

Additional Key Words and Phrases: Modularity; Cardinality; Random graph

ACM Reference Format:
Zijin Feng, Miao Qiao, and Hong Cheng. 2023. Modularity-based Hypergraph Clustering: Random Hypergraph Model, Hyperedge-cluster Relation, and Computation. *Proc. ACM Manag. Data* 1, 3 (SIGMOD), Article 215 (September 2023), 25 pages. https://doi.org/10.1145/3617335

1 INTRODUCTION

A graph models the interconnections among real-world objects using a set of edges on a set of nodes. Typically, a graph captures *pairwise* relations among objects, i.e., each edge connects exactly

Authors' addresses: Zijin Feng, Department of Systems Engineering and Engineering Management, and Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong, Hong Kong, zjfeng@se.cuhk.edu.hk; Miao Qiao, University of Auckland, New Zealand, miao.qiao@auckland.ac.nz; Hong Cheng, Department of Systems Engineering and Engineering Management, and Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong, Hong Kong, hcheng@se.cuhk.edu.hk.

2836-6573/2023/9-ART215 $15.00
https://doi.org/10.1145/3617335

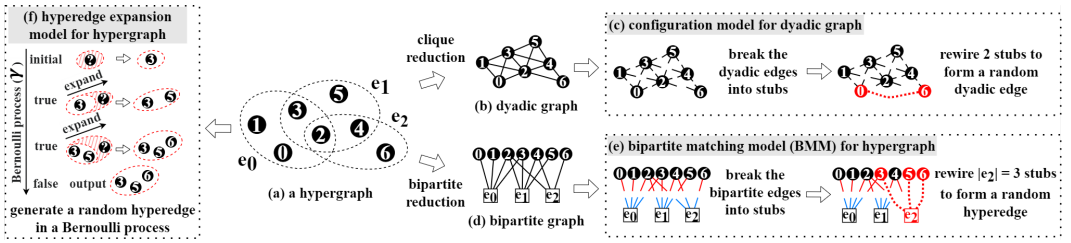


Fig. 1. A running example to illustrate the concepts of hypergraph, clique reduction (details in Section 2.2.1), bipartite reduction (details in Section 2.2.2), and random graph models (Section 2.1, Section 2.2, and Section 3).

two nodes; a more general graph models *groupwise* relations among objects, where each edge connects a group of two or more nodes. The former type of graph is often called a *dyadic graph* where each edge is a dyadic edge; the latter is called a *hypergraph* where each edge is a hyperedge. Extensive high-order connections in graph-based applications [13] have triggered the study and analysis of hypergraphs. For example, in a coauthor network, a node represents an author and a hyperedge represents the author team who published a paper in a joint effort. In contrast, a dyadic graph can only capture the pairwise co-authorship, losing the information of the author team of each paper. In a protein complex network [53], a node denotes a protein and a hyperedge represents a group of proteins that form a multi-protein complex. In contrast, a dyadic graph can only indicate if two proteins co-exist in forming a complex but cannot represent the complex itself (such a dyadic network is known as a protein-protein interaction network). A hypergraph preserves high-order connections of the applications, thus allowing finer graph analysis. Figure 1(a) is a hypergraph with 3 hyperedges e_0, e_1, e_2. As an example, hyperedge $e_0 = \{v_0, v_1, v_2, v_3\}$ has 4 nodes. $|e_0| = 4$ is called the hyperedge cardinality of e_0.

One important graph analytical task is graph clustering. It aims at partitioning the nodes of a graph into a collection of node sets called clusters. Nodes in each cluster should be more closely connected to the other nodes in the same cluster than to the nodes in other clusters. Graph clustering is widely applied in various commercial and scientific scenarios such as community detection [26], link prediction [57], group-oriented marketing [63], brain parcellations [59], etc., and thus has been extensively studied on dyadic graphs. A highly scalable line of clustering is modularity-based which, as the name suggests, maximizes a function called modularity. Modularity-based clustering such as Louvain [14] has been widely used in industry applications due to its scalability and clustering quality [65]. The key to its success is the selection of a *random graph model*. Specifically, the modularity function evaluates a clustering $\mathscr{C}$ based on the difference in its "alignment" to the data graph G and its expected "alignment" to a random graph G'. Here G' is generated by the chosen *random graph model* while the "alignment" indicates the portion of edges that fall in the same cluster in $\mathscr{C}$. The selection of the random graph model not only decides which clustering will be preferred, but also the cost required in computing the alignment. An ideal random graph model should preserve *essential* statistics of the data graph G – enough for quality clustering without overburdening the computation. Such a model is found on dyadic graphs: the *configuration model* [10] (depicted in Figure 1(c)) generates random graphs that preserve the degree distribution of G and allow the efficient computation of the expected alignment.

When it comes to hypergraphs, *Modularity-based Hypergraph Clustering* (MBHC) is worth studying for three reasons. Firstly, hypergraph clustering has drawn an increasing attention recently [20, 34, 38, 39, 44, 58, 60] due to its wide applications in social community detection [20], VLSI placement [47], metabolic reactions analysis [37], image segmentation [36], multi-relational

data analysis including clustering [11, 29, 60] and recommendation [18, 43]. Secondly, as discussed in [48, 58], existing hypergraph clustering with objective functions often struggles with scalability and efficiency. Hypergraph clustering methods based on measures such as normalized cut [30], conductance [58], and correlation [61] were tested only on small graphs with approximately 10^2 to 10^4 edges. Modularity-based clustering which allows the clustering of dyadic graphs of large size [65] is more likely to be scaled up on hypergraphs. Thirdly, existing MBHC methods [7, 20, 34, 38, 39] lack a *random hypergraph model* that can capture the *hyperedge-cluster alignment* effectively in their modularity without overloading the computation. To speed up the computation, they lose either entire or partial groupwise information, jeopardizing the cluster quality.

A line of existing MBHC [7, 38, 39] transforms the hypergraph to a dyadic graph using *clique reduction* which converts a hyperedge e into a clique of dyadic edges among all nodes in e. Figure 1(b) is a dyadic graph by clique reduction from the hypergraph in Figure 1(a). It then applies existing dyadic graph clustering. The main downside is the **loss of the groupwise information** in the hypergraph. Another line [19, 34, 41] transforms the hypergraph into a bipartite graph using *bipartite reduction* and then generates a random hypergraph. Figure 1(d) is a bipartite graph by bipartite reduction from the hypergraph in Figure 1(a). We thus call their random hypergraph model Bipartite Matching Model (BMM) (depicted in Figure 1(e)). BMM generates a random hypergraph that preserves not only the degree distribution, but also the *cardinality distribution*: each hyperedge of the given hypergraph has a counterpart in the random hypergraph with the same cardinality. The downside is that **the computation of the expected alignment between the hyperedges and the clustering on the random hypergraph generated by** BMM **is expensive**. Specifically, for a hyperedge e, the expected alignment of its counterpart hyperedge in the random graph to a cluster C is computed in $O(|e|)$ time. This needs to be calculated between every hyperedge and every cluster, as long as they have nodes in common, leading to at least quadratic complexity for clustering computation (see analysis in Section 2.2.2). To avoid such a heavy computation, most existing MBHC [20, 34] consider only the hyperedge e and cluster C in the computation of modularity when all the nodes in e fall in C. Otherwise, e is ignored. This All-Or-Nothing (AON) constraint loses partial groupwise information and thus leads to degenerated clustering quality: a hyperedge cannot be reflected in the clustering result at all even if most of its nodes are aligned with a cluster.

The drawbacks of existing random hypergraph models in the context of modularity-based clustering motivate us to explore real-world hypergraphs, which leads to an observation that their cardinality distributions can be well-captured by an *exponential function*. Based on this observation, the paper proposes a suite of new techniques including Hyperedge Expansion Model (HEM), a random hypergraph model, Partial Innerclusteredge (PI) modularity, a modularity function under HEM, and a clustering algorithm PI clustering (PIC). The merits of our solution are summarized below.

(1) HEM simplifies the existing random hypergraph model while preserving the essential features of real hypergraphs including the hyperedge number, the degree sequence, and a highly fitted cardinality distribution. The simplification paves the way to an efficient modularity computation for quality clustering.
(2) PI modularity function collects partial (non-AON) contributions from hyperedges to clusters which can be efficiently computed.
(3) PIC optimizes the PI modularity iteratively; to further scale PIC to massive graphs, we propose two non-trivial optimization techniques to prune the search space in the iterative adjustment of clustering for maximizing the PI modularity.

(4) Empirically, PIC outperforms the state-of-the-art methods in both effectiveness and scalability. When averaged over all hypergraphs we tested, PIC obtained 15 times higher F-measure [49], 4 times higher purity [49], 10 times higher ARI [33], and 75% higher NMI [35]. PIC is also quite scalable to large hypergraphs and faster than all baseline methods by an average of four orders of magnitude and up to five orders of magnitude.

We show that the slight relaxation of the exact preservation of the cardinality distribution allows PIC to achieve scalability on hypergraphs without compromising the clustering quality. Surprisingly, the clustering quality of PIC turned out to be empirically better than that of [20, 34] that preserves the exact cardinality distribution[1].

The paper is organized as follows. Section 2 formally defines the problem. Section 3 proposes the random hypergraph model HEM, Section 4 describes the PI modularity, and Section 5 proposes the PI Clustering algorithm (PIC) with two optimizations. Section 6 discusses the related work, Section 7 shows the empirical results, and Section 8 concludes the paper.

2 PRELIMINARY CONCEPTS

A dyadic graph $G(V, E)$ has a node set V and an edge set E where an edge $e = (u, v) \in E$ connects two nodes $u, v \in V$. A hypergraph $H(V, E)$ is a generalization of a dyadic graph where an edge $e \in E$, called a hyperedge, connects multiple nodes in V and is denoted as $e = \{v_1, v_2, \ldots, v_k\} \subseteq V$. Both dyadic graphs and hypergraphs are called graphs. Define the cardinality of e, $|e|$, as the number of nodes in e. A hyperedge e is trivial if $|e| < 2$ as it does not describe any relation among nodes in V.

Given a hypergraph $H(V, E)$, denote by $n(H) = |V|$ the number of nodes, and $m(H) = |E|$ the number of edges. Let $S = \biguplus_{e \in E} e$ be a multi-set of nodes (where a node can appear multiple times in S). For each node $v \in V$, define the degree of v, denoted as $d_v(H)$, as the number of duplications of v in S. $d_v(H)$ equals the number of hyperedges in H that contain node v. For a set $C \subseteq V$, denote by $\mathrm{vol}(C, H) = \sum_{v \in C} d_v(H)$ the volume of set C in the graph. Denote by $\mathrm{vol}(H) = \mathrm{vol}(V, H)$ the volume of hypergraph H, and by $\mathrm{vol}_2(H) = \sum_{e \in E} \binom{|e|}{2}$ the pairwise volume of hypergraph H. When the hypergraph H is clear in the context, we use simplified notations of n, m, d_v and $\mathrm{vol}(C)$, respectively.

Example 1. *Consider the hypergraph $H(V, E)$ in Figure 1(a) with $V = \{v_0, v_1, \cdots, v_6\}$ and $E = \{e_0, e_1, e_2\}$. The nodes in e_0, e_1 and e_2 form the multi-set $S = \{v_0, v_1, v_2, v_2, v_2, v_3, v_3, v_4, v_4, v_5, v_6\}$. The degree of v_0 is $d_{v_0} = 1$ and that of v_2 is $d_{v_2} = 3$. For set $C = \{v_0, v_1, v_2, v_3\}$, the volume is $\mathrm{vol}(C) = d_{v_0} + d_{v_1} + d_{v_2} + d_{v_3} = 7$. The volume of hypergraph H is $\mathrm{vol}(H) = 11$ and the pairwise volume of H is $\mathrm{vol}_2(H) = 15$.*

Lemma 1. *[17] For a hypergraph $H(V, E)$, the volume $\mathrm{vol}(H) = \sum_{e \in E} |e|$. In particular, when the graph is dyadic, $\mathrm{vol}(H) = 2m$.*

Clustering. Given a graph $H(V, E)$, a clustering $\mathscr{C} = \{C_1, C_2, \cdots, C_k\}$ is a partition of V, that is, V is the disjoint union $\dot{\bigcup}_{i \in [k]} C_i$.

2.1 Random Graph Model and Modularity for Dyadic Graph Clustering

The modularity function proposed by Newman-Girvan (NG) [51] is widely used in dyadic graph clustering as the objective function. Our hypergraph clustering problem also uses modularity as the objective function. In the following, we introduce the random graph model, called configuration model, used to generate a random graph G' from a dyadic graph G and how to calculate the NG modularity of a clustering $\mathscr{C}$ from G and G'.

[1]We believe that there might be other design options of random hypergraph model that lead to better scalability and clustering quality: we are open to this line of discussions.

Configuration Model for Dyadic Graph [10]. Given a graph $G(V, E)$, the configuration model generates a random graph $G'(V, E')$ preserving the degree distribution of G. For any positive integer k, G and G' have the same number of nodes with degree k. G' is generated in a break-and-rewire process as illustrated in Figure 1(b-c).

(1) Break. Break each edge $e(u, v) \in E$ into two stubs in the form of $(u, -)$ and $(v, -)$. A stub, e.g., $(v, -)$, has one end fixed and the other open. Each $v \in V$ has d_v stubs and there are in total $2m$ stubs. We use set T to denote the $2m$ stubs.
(2) Rewire. Select two stubs uniformly at random from T without replacement, denoted as $(u', -)$ and $(v', -)$, then rewire them to form a new edge (u', v'). Repeat step (2) m times, generating m edges to form E'.

Newman-Girvan (NG) Modularity [51] for Dyadic Graph. For a cluster C of a graph G, an edge is an innercluster edge of C if both ends of the edge are in C. The set of innercluster edges is denoted as $E(C) = \{(u, v) \in E | u, v \in C\}$. Lemma 2 shows that the expected number of innercluster edges of G' in C is approximately $\frac{\text{vol}^2(C)}{4m}$.

Lemma 2 ([51]). *Let $G'(V, E')$ be the random graph of $G(V, E)$ generated under the configuration model. Given cluster $C \subseteq V$, the expected number of innercluster edges of G' in C is $Exp[|E(C)|] = \frac{\text{vol}^2(C)}{4m-1} \approx \frac{\text{vol}^2(C)}{4m}$ if we assume that $m \gg 1$.*

The NG modularity computes the difference between the actual number of innercluster edges of G in C and the expected number of innercluster edges of G' in C, i.e., $|E(C)| - Exp[|E(C)|]$, $\forall C \in \mathscr{C}$:

$$\text{NG}(\mathscr{C}) = \frac{\sum_{C \in \mathscr{C}} |E(C)| - Exp[|E(C)|]}{m} = \frac{\sum_{C \in \mathscr{C}} |E(C)| - \frac{\text{vol}^2(C)}{4m}}{m}. \quad (1)$$

2.2 Existing Hypergraph Clustering Methods

Existing hypergraph clustering methods convert a hypergraph into a dyadic graph by clique reduction [7, 38, 39] or into a bipartite graph by bipartite reduction [19, 34, 41]. We briefly describe them below and discuss their limitations.

2.2.1 Clique Reduction. [7, 38, 39] use clique reduction to convert a hypergraph $H(V, E)$ to a dyadic graph G on V, then apply dyadic graph clustering to G. It substitutes each hyperedge $e \in E$ with a clique of dyadic edges that are unweighted [7] or weighted [38, 39]. Figure 1(b) depicts a dyadic graph converted from the hypergraph in Figure 1(a). Clique reduction leads to *scalability* but hinders *effectiveness*, due to the loss of high-order information.

2.2.2 Bipartite Reduction. Bipartite reduction converts a hypergraph $H(V, E)$ to a bipartite graph $B(V \dot{\bigcup} E, E_B)$ based on the node-hyperedge relations in H. B has two sets of nodes V and E. For a node $v \in V$ and a hyperedge $e \in E$, an edge $(v, e) \in E_B$ iff $v \in e$. Figure 1(d) shows a bipartite graph converted from the hypergraph in Figure 1(a). Most existing work along this line uses B to generate random hypergraphs [19, 34, 41] which *preserve both the node degrees and precise hyperedge cardinalities of H* (called the *cardinality sequence* of H). We call this random hypergraph model Bipartite Matching Model (BMM). Specifically, BMM divides each edge (v, e) in B into two stubs (see Figure 1(d-e)), a red stub from v and a blue stub from e. It then selects, for each hyperedge $e \in E$, $|e|$ stubs uniformly at random from the red stubs either with replacement [41] or without replacement [19]. BMM then joins $|e|$ red stubs to form e's *corresponding random hyperedge* having cardinality $|e|$ (see Figure 1(e)). The generated randomized counterpart of $H(V, E)$ is denoted as $H'(V, E')$.

Modularity for Hypergraph. Given a hypergraph H, its randomized counterpart H' under BMM, and a clustering $\mathscr{C}$ of H, the modularity is calculated by $\frac{1}{m}\sum_{C\in\mathscr{C}}(|E(C)| - Exp[|E(C)|])$, the same as Equation 1. Note in a cluster C, a hyperedge e is an innercluster hyperedge of C if all the nodes of e are in C. Thus $|E(C)|$ only counts those hyperedges in H whose nodes all fall in cluster C and ignores the rest, i.e., an All-Or-Nothing (AON) hyperedge-to-cluster contribution [20, 34]. The expected number of innercluster hyperedges $Exp[|E(C)|]$ in H' is calculated by:

$$Exp[|E(C)|] = \sum_{e\in E}(\frac{\text{vol}(C)}{\text{vol}(H)})^{|e|} = \sum_{s\geq 2}|E_s|\cdot(\frac{\text{vol}(C)}{\text{vol}(H)})^{s}, \tag{2}$$

where $|E_s|$ denotes the number of innercluster hyperedges in H having cardinality s. Equation 2 can be computed in $O(1)$ time. Though it brings the benefit of efficiency, the rigid AON constraint leads to degenerated clustering quality since a large number of hyperedges are not counted in the modularity computation.

Kamiński et al. [34] proposed a non-AON hyperedge-to-cluster contribution to calculate modularity: it counts a hyperedge e in calculating $Exp[|E(C)|]$ when e has $> 50\%$ of nodes in C. For each hyperedge $e \in E$, the expected contribution from e's corresponding random hyperedge in H' can be calculated by enumerating all integers $s' \in (|e|/2, |e|]$, accordingly $Exp[|E(C)|]$ is calculated by:

$$Exp[|E(C)|] = \sum_{\substack{e\in E,\\ s'\text{: integers in }(|e|/2,|e|]}} \binom{|e|}{s'} \frac{(\frac{\text{vol}(C)}{\text{vol}(H)})^{s'}}{(1-\frac{\text{vol}(C)}{\text{vol}(H)})^{s'-|e|}}. \tag{3}$$

Computing Equation 3 takes $O(|e|)$ time for a hyperedge e, and $O(\text{vol}(H))$ time in total for all hyperedges in E. Louvain-style [14] modularity-based clustering evaluates the modularity function $|V|$ times, leading to $O(\text{vol}(H)|V|)$ complexity which is very expensive.

Remark. Given the limitations of existing hypergraph clustering methods, in this paper, we design a novel random hypergraph model, a new non-AON hypergraph modularity function, and an effective and scalable hypergraph clustering algorithm.

3 HYPEREDGE EXPANSION MODEL

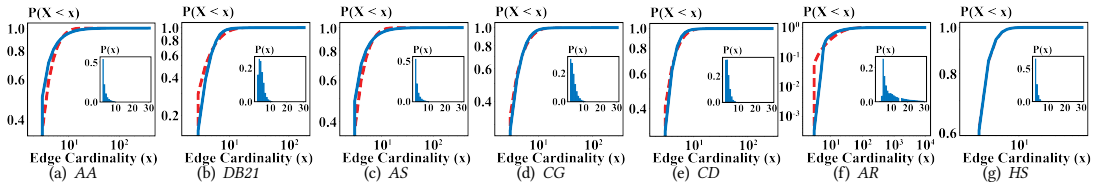


Fig. 2. For each real hypergraph (subfigure), the blue line shows the cumulative distribution, the red line shows the cumulative distributions of exponential models used to fit the blue line, the inset shows the probability density of the edge cardinality x.

This section proposes a random graph model, called Hyperedge Expansion Model (HEM). Given a hypergraph $H(V, E)$, HEM produces a random hypergraph $H'(V, E')$ by generating $m = |E|$ hyperedges on V. Its building block function edge-expansion() (in Algorithm 1) generates a hyperedge e' by repeatedly adding nodes into e'. A node $v \in V$ is selected from V with probability proportional to its degree, $p_v = \frac{d_v(H)}{\text{vol}(H)}$. Let the continuation probability γ be $\frac{\text{vol}(H)-2m}{\text{vol}(H)-m}$. $\gamma \in [0, 1)$ if

Algorithm 1: edge-expansion

Input: A hypergraph $H(V, E)$, a multi-set of nodes e'
Output: A multi-set of nodes e'
$v \leftarrow$ a random node selected with probability $p_v = d_v(H)/\text{vol}(H)$;
return $e' \leftarrow e' \cup \{v\}$;

$m > 0$. As each edge $e \in E$ is non-trivial, i.e., has at least 2 nodes, every random hyperedge e' is non-trivial and is generated in a Bernoulli process:

(1) Initialize $e' \leftarrow$ edge-expansion($\emptyset$). e' now has one node.
(2) Call edge-expansion(e') then terminate the process with probability $(1-\gamma)$. Repeat step (2) until the termination is triggered.

Figure 1(f) shows the generation of a random hyperedge e' by HEM. e' is initialized with a randomly selected node v_3, then v_5 and v_6 are selected and added to e' in the first and second trial after which the Bernoulli process terminates. A random hyperedge $e' = \{v_3, v_5, v_6\}$ is then generated. By running the above process m times, HEM can generate a randomized hypergraph $H'(V, E')$ of $H(V, E)$. Lemma 3 and Theorem 1 show the properties of HEM.

Lemma 3. *Given a hypergraph H, the random hypergraph H' generated under* HEM *has* $Exp[\text{vol}(H')] = \text{vol}(H)$ *when* $\gamma = \frac{\text{vol}(H)-2m}{\text{vol}(H)-m}$.

Proof. Firstly, since each hyperedge in H is non-trivial and $\text{vol}(H) \geq 2m$, we have $\gamma = \frac{\text{vol}(H)-2m}{\text{vol}(H)-m} \in [0, 1)$. Let Y be the cardinality of any hyperedge e' generated under HEM. For each hyperedge e', as $Y-1$ out of its Y nodes are generated in the Bernoulli process, $Y-1$ is a random variable following a geometric distribution with success rate $1-\gamma$. Thus, the probability that the process terminates in the $(Y-1)$-th trial is $\gamma^{Y-2}(1-\gamma)$ and we have $Exp[Y-1] = \frac{1}{1-\gamma}$. As HEM independently generates m hyperedges to produce H', we have $Exp[\text{vol}(H')] = m \cdot Exp[Y] = m \cdot (\frac{1}{1-\gamma}+1) = \text{vol}(H)$. □

Theorem 1. *Given a hypergraph H, by setting $\gamma = \frac{\text{vol}(H)-2m}{\text{vol}(H)-m}$, the random hypergraph $H'(V, E')$ under* HEM *preserves 1) the number of edges m, 2) the volume $Exp[\text{vol}(H')] = \text{vol}(H)$, and 3) the degree sequence of H: for each $v \in V$, $Exp[d_v(H')] = d_v(H)$.*

Proof. 1) is obvious. 2) can be derived from Lemma 3. For 3), we consider a node $v \in V$. Let X be a random variable that v is selected in the edge-expansion(e') of HEM, we have $Exp[X] = \frac{d_v(H)}{\text{vol}(H)}$. Let Y be a random variable denoting the cardinality of hyperedge generated by HEM. From Lemma 3, we have $Exp[Y] = \frac{\text{vol}(H)}{m}$. Since X and Y are independent variables, $Exp[XY] = Exp[X]Exp[Y] = \frac{d_v(H)}{\text{vol}(H)} \cdot \frac{\text{vol}(H)}{m} = \frac{d_v(H)}{m}$ is the expected number of occurrences of v in e'. Therefore, $Exp[d_v(H')] = m \cdot Exp[XY] = d_v(H)$. □

We examine 16 hypergraphs that we found in real-world applications (see details in Table 2) to see how HEM preserves the cardinality distribution – we let the exponential model be a bridge between the real hypergraphs and their corresponding random hypergraphs under HEM. Figure 2 shows the probability densities $p(x)$ of edge cardinality x in the insets and the cumulative distributions $P(X < x)$ (blue lines) of cardinality in logarithmic scale on 7 out of 16 hypergraphs due to space limit. Note that the insets only show a range of x from 3 to 30 because (1) $p(x)$ for larger value x are too small to be visualized, and (2) neither real hypergraphs nor random hypergraphs contain trivial edges and thus $P(X < 2) = 0$. Given γ in HEM, we use the continuous probability density function of the exponential model $p(x) = C \cdot exp(-\lambda x)$ to depict the discrete edge cardinality density function $P[|e| = s] = \gamma^{s-2}(1-\gamma) = \frac{1-\gamma}{\gamma^2}\gamma^s$ (see the proof of Lemma 3), by setting $C = \frac{1-\gamma}{\gamma^2}$ and $\lambda = -\ln(\gamma)$.

The cumulative distributions of exponential models with given γs are plotted in red dashed lines. We measure the "distance" between the cardinality cumulative probability distribution function (CDF) $S(x)$ of the real hypergraphs and the CDF $P(x)$ of the corresponding random hypergraphs using Mean Squared Error $MSE = \sum_x \frac{(S(x)-P(x))^2}{n_x}$. MSE aggregates the squared distance between $S(x)$ and $P(x)$ for each x, n_x is the number of distinct x. The average MSE on all 16 hypergraphs is 10^{-3}. As a comparison, the power-law model (alternative to the exponential model, see [23]) on average obtains 42 times larger MSE than the exponential model. This shows the exponential model can well fit the distribution of empirical data.

Remark. HEM simplifies existing random hypergraph models by relaxing the preservation of the cardinality of every individual hyperedge. In otherw BMM generates, for each $e \in E$, a corresponding random hyperedge e' having the same cardinality while our HEM preserves the cardinality distribution of hyperedges in E. The next section describes how HEM paves the way for efficient computation of our new hypergraph modularity function which can be naturally integrated into the hyperedge generation process of HEM.

4 PI MODULARITY

This section describes how to compute the modularity of a clustering on a hypergraph with HEM. We first define θ-**innercluster hyperedge**, a relaxation of innercluster hyperedge in Section 2.2.

Definition 1 (θ-Innercluster hyperedge). *Given a multi-set e' of s nodes, a cluster C, and a real parameter $\theta \in [0,1]$, let $Y = \{y_1, y_2, \cdots, y_s\}$ be a sequence of random variables $y_i \in \{0,1\}$, s.t., $\sum_{i\in[s]} y_i \leq (1-\theta) \times s$. e' is a θ-innercluster hyperedge of C if for any outcome of Y, there is a node ordering of $e' = \{v_1, v_2, \cdots, v_s\}$ such that $\forall i \in [s]$,either $v_i \in C$ or $y_i = 1$. In other words, if $y_i = 0$, then node v_i must belong to C; if $y_i = 1$, then the constraint can be relaxed and thus v_i can be exempted from the test of $v_i \in C$.*

Note that a random hyperedge e' is a θ-innercluster hyperedge of a cluster C if e' has at least $\theta \times |e'|$ nodes in C. Definition 1 leads to a relaxation-rejection process in estimating the contributions from θ-innercluster hyperedges in a random hypergraph to a cluster. Y is generated such that each $y_i = 1$ with probability $1-\theta$ and $y_i = 0$ with probability θ, $\forall i \in [s]$. For a random hyperedge e' of cardinality s, an expected number of $(1-\theta) \times s$ nodes in e' will have their constraints relaxed. Thus, if there is an ordering of nodes in $e' = \{v_1, v_2, \cdots, v_s\}$ such that for each $i \in [s]$, $y_i = 1$ or $v_i \in C$, then e' is a θ-innercluster hyperedge. In HEM, for a specific outcome of random variables of Y, consider the i-th invocation of procedure edge-expansion(): if the relaxation is not granted (i.e., $y_i = 0$) or the test of $v_i \in C$ fails, this ordering of the nodes in hyperedge will be rejected. An extreme case is when $\theta = 1$ where no relaxation is granted, thus all the nodes in a θ-innercluster hyperedge are forced to be in C; as another extreme case, when $\theta = 0$, all the hyperedges are θ-innercluster hyperedges without being rejected. This relaxation-rejection process leads to a simplified computation of the hyperedge-to-cluster contributions as described below.

Since each node v_i in the edge expansion of HEM is selected proportional to the degree distribution over V, the probability $Pr(y_i = 1 \text{ or } v_i \in C)$ that the edge expansion passes is $(1-\theta) + \theta\frac{\text{vol}(C)}{\text{vol}(H)}$. Denote by $\eta = \theta\left(1 - \frac{\text{vol}(C)}{\text{vol}(H)}\right)$. According to Definition 1, the condition $(y_i = 1 \text{ or } v_i \in C)$ has to hold for all $i \in [s]$, thus the probability that a random hyperedge of size s can pass every edge expansion without being rejected is $[(1-\eta)]^s$. Combining the continuation probability γ in the HEM model, the expected number of non-rejected hyperedges in cluster C under HEM, $Exp[|E(C)|]$, is

$$Exp[|E(C)|] = \frac{1-\gamma}{\gamma^2}|E| \sum_{s=2}^{\infty} \left[\gamma\left(1-\theta+\theta\frac{\text{vol}(C)}{\text{vol}(H)}\right)\right]^s. \tag{4}$$

Lemma 4 provides an efficient way to calculate the expected hyperedge-to-cluster contributions of θ-innercluster hyperedges to cluster C, i.e., $Exp[|E(C)|]$ defined in Equation 4.

Lemma 4. *The expected contributions of random hyperedges under* HEM *to cluster C can be computed via $Exp[|E(C)|] = |E|(1-\eta)^2(1+\frac{\gamma\eta}{1-\gamma})^{-1}$, where $\eta = \theta\left(1-\frac{\text{vol}(C)}{\text{vol}(H)}\right)$.*

Proof. Since $0 \le \gamma < 1$, $1-\theta+\theta\frac{\text{vol}(C)}{\text{vol}(H)} = 1-\eta \le 1$, term $\gamma(1-\eta) \in [0,1)$. Thus, $\sum_{s=2}^{\infty}(\gamma(1-\eta))^s$ converges to $\frac{\gamma^2(1-\eta)^2}{1-\gamma(1-\eta)} = \frac{\gamma^2(1-\eta)^2}{1-\gamma+\gamma\eta}$. Then we have $Exp[|E(C)|] = \frac{1-\gamma}{\gamma^2}|E|\sum_{s=2}^{\infty}(\gamma(1-\eta))^s = \frac{1-\gamma}{\gamma^2}|E|\frac{\gamma^2(1-\eta)^2}{1-\gamma+\gamma\eta} = |E|(1-\eta)^2(1+\frac{\gamma\eta}{1-\gamma})^{-1}$. □

We then define Partial Innerclusteredge (PI) hypergraph modularity.

Definition 2 (Partial Innerclusteredge (PI) hypergraph modularity). *For an edge e and a cluster C, denote by $l(e,C) = \frac{|e\cap C|}{|e|}$ the loyalty of e to C. Let $\rho : [0,1] \mapsto [0,1]$ be a monotonically increasing function that maps the loyalty of an edge e to a cluster C to its contribution $\rho(l(e,C))$. When an edge e has positive loyalties to two disjoint clusters, e should support the merge of the two clusters. In other words, ρ needs to be super-additive [31] that for $\forall x, y \in (0,1]$, if $x+y \le 1$, $\rho(x)+\rho(y) \le \rho(x+y)$. Let $E_\theta(C)$ be the set of θ-innercluster hyperedges of C in the hypergraph $H(V,E)$. The support to C on the hypergraph is the aggregation of the contributions from all edges in $E_\theta(C)$: $\text{supt}_\theta(C) = \sum_{e\in E_\theta(C)}(\rho(l(e,C)))$. The modularity of a clustering $\mathscr{C}$ on a hypergraph H is:*

$$\text{PI}_H(\mathscr{C}) = \frac{1}{|E|}\sum_{C\in\mathscr{C}}\left[\text{supt}_\theta(C) - Exp[|E(C)|]\right], \tag{5}$$

where $Exp[|E(C)|]$ can be computed via Lemma 4 in $O(1)$ time.

The proposed PI modularity measures the deviation between the actual and expected contribution from θ-innercluster hyperedges to a cluster C, summed over all the clusters in $\mathscr{C}$. PI modularity together with the proposed random graph model HEM follow the framework of the Newman-Girvan modularity on dyadic graphs, but $\text{PI}_H(\mathscr{C})$ replaces the traditional concept of innercluster edges with θ-innercluster hyperedges. It counts the contributions from a hyperedge e to a cluster C even if a small portion of nodes in e fall out of C (i.e., non-AON contribution). The expected contribution $Exp[|E(C)|]$ under HEM can be computed in $O(1)$ time by Lemma 4. In contrast, $Exp[|E(C)|]$ calculated by Equation 3 takes $O(\text{vol}(H))$ time. This gives PI modularity better scalability than its counterpart adopting non-AON contribution under the BMM model.

Remark. PI modularity controls the relaxations with Bernoulli trials instead of counting the exact number of θ-innercluster hyperedges. The relaxation-rejection random process can be creatively integrated into the edge expansion process of HEM, and the expected hyperedge-to-cluster contributions of θ-innercluster hyperedges are interpretable by the edge expansion process. The merits of our HEM and PI modularity are: (1) preservation of high-order information and properties such as cardinality distribution (in Theorem 1) in hypergraphs, and (2) efficient computation of the modularity.

5 PI CLUSTERING

This section proposes a clustering algorithm called PI Clustering (PIC) to optimize the PI modularity with two novel optimization techniques for improving efficiency and scalability.

5.1 Loyalty Function and PI Modularity Computation

To quantify the contribution of an individual node of a hyperedge to a cluster, we denote a weight of a node u in a hyperedge e by $w(u,e)$ and initialize $w(u,e) = \frac{1}{|e|}$. Using the node weight notation, the (weighted) degree of a node u can be expressed as $d_u = \sum_{e \in E} w(u,e)$, and the loyalty can be equivalently written as $l(e,C) = \sum_{u \in e \cap C} w(u,e)$. When e and C are clear from the context, we use the simplified notation l to denote the loyalty.

As defined in Definition 2, function ρ maps loyalty $l(e,C)$ (from hyperedge e to a cluster C) to the contribution of e to C in the modularity. We consider 4 candidate functions for ρ, all of which satisfy the monotonically increasing and super-additive requirements.

- All-Or-Nothing (AON): $\rho(l) = 1$ if $l = 1$ and 0 otherwise;
- Linear-over-Logarithm: $\rho(l) = \frac{l}{\log(1/l+1)}$;
- Quadratic: $\rho(l) = l^2$;
- Exponential: $\rho(l) = \frac{\exp(l)-1}{\exp(1)-1}$. The denominator, Euler's number ($\exp(1) = 2.71828\cdots$) minus 1, transforms the range to $[0,1]$.

The rationale behind the monotonicity property is that the more nodes of a hyperedge e fall in a cluster C, the higher contribution e makes to C in the modularity. The rationale behind the super-additive property is that, for two clusters C_1, C_2 each having some nodes of a hyperdge e, they should be brought closer by e, thus the merge of them is encouraged. This rationale is realized by the super-additive property, i.e., $\rho(x+y) \geq \rho(x) + \rho(y)$, where $\rho(x+y)$, $\rho(x)$, and $\rho(y)$ are the contribution of e to $C_1 \cup C_2$, C_1, C_2, respectively. The instantiation of ρ will not affect the following discussions and thus we leave the selection of ρ as a super parameter, which shall be discussed in the experiment, similar to the parameter θ.

Consider two clusterings $\mathscr{C}$ and $\mathscr{C}'$ on the same hypergraph H where $\mathscr{C}'$ merges cluster $\{u\} \in \mathscr{C}$ with another cluster $C \in \mathscr{C}$. Denote by $\Delta\mathsf{PI}_{u \to C} = \mathsf{PI}_H(\mathscr{C}') - \mathsf{PI}_H(\mathscr{C})$ the PI modularity gain when these two clusters in $\mathscr{C}$ are merged. Recall that Definition 2 defines $\eta = \theta\left(1 - \frac{\mathrm{vol}(C)}{\mathrm{vol}(H)}\right)$ for cluster C, we denote it as η_C where C is treated as a parameter to denote the η value for different clusters.

$$\begin{aligned}\Delta\mathsf{PI}_{u \to C} = &\frac{1}{|E|}(\mathsf{supt}_\theta(C \cup \{u\}) - \mathsf{supt}_\theta(\{u\}) - \mathsf{supt}_\theta(C)) \\ &+(1-\gamma)\left(\frac{(1-\eta_C)^2}{1-\gamma+\gamma\eta_C} + \frac{(1-\eta_{\{u\}})^2}{1-\gamma+\gamma\eta_{\{u\}}} - \frac{(1-\eta_{C\cup\{u\}})^2}{1-\gamma+\gamma\eta_{C\cup\{u\}}}\right)\end{aligned} \tag{6}$$

The value of $\mathsf{supt}_\theta(C \cup \{u\}) - \mathsf{supt}_\theta(\{u\}) - \mathsf{supt}_\theta(C)$ is the support gain by merging the two clusters. Symmetrically, the PI modularity gain $\Delta\mathsf{PI}_{C \to u}$ of removing a node $u \in C$ from cluster C is

$$\Delta\mathsf{PI}_{C \to u} = -1 \times \Delta\mathsf{PI}_{u \to C \setminus \{u\}}. \tag{7}$$

PROOFS OF EQUATIONS 6-7. As all clusters other than $\{u\}$ and C remain unchanged after the merge, $\Delta\mathsf{PI}_{u \to C} = \mathsf{PI}_H(C \cup \{u\}) - \mathsf{PI}_H(\{u\}) - \mathsf{PI}_H(C)$. As we have $(1-\eta)^2(1+\frac{\gamma\eta}{1-\gamma})^{-1} = (1-\gamma)\frac{(1-\eta)^2}{1-\gamma+\gamma\eta}$,

$$\begin{aligned}\Delta\mathsf{PI}_{u \to C} = &\frac{1}{|E|}(\mathsf{supt}_\theta(C \cup \{u\}) - \mathsf{supt}_\theta(\{u\}) - \mathsf{supt}_\theta(C)) \\ &+ \frac{(1-\gamma)(1-\eta_C)^2}{1-\gamma+\gamma\eta_C} + \frac{(1-\gamma)(1-\eta_{\{u\}})^2}{1-\gamma+\gamma\eta_{\{u\}}} - \frac{(1-\gamma)(1-\eta_{C\cup\{u\}})^2}{1-\gamma+\gamma\eta_{C\cup\{u\}}}.\end{aligned}$$

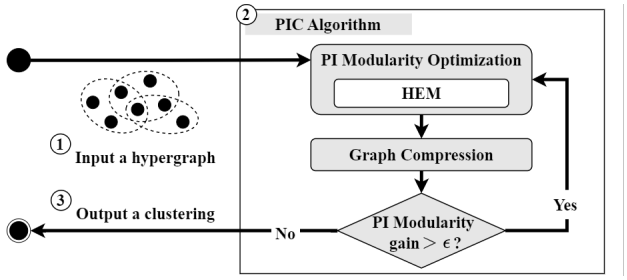


Fig. 3. Flow diagram of PI Clustering (PIC) algorithm.

By extracting the common factor $1-\gamma$ for the last three terms, the proof of Equation 6 completes. For Equation 7 we have:

$$\begin{aligned}\Delta\mathsf{PI}_{C\to u} &= \mathsf{PI}_H(C\setminus\{u\}) + \mathsf{PI}_H(\{u\}) - \mathsf{PI}_H(C)\\ &= -1\times(\mathsf{PI}_H(C) - \mathsf{PI}_H(\{u\}) - \mathsf{PI}_H(C\setminus\{u\})) = -1\times\Delta\mathsf{PI}_{u\to C\setminus\{u\}}.\end{aligned}$$

This completes the proof of Equation 7. □

5.2 PI Clustering Algorithm

Our clustering algorithm PIC is Louvain-style. Start with singleton clustering where each cluster contains a single node. Each iteration scans all the nodes in the graph: for each v, try to mobilize v from its own cluster to its neighbors' clusters. For node v, a node u is v's neighbor if u and v belong to a common hyperedge. We use an adjacency matrix A to represent the neighborhood relationship, the size of which is $\mathrm{vol}_2(H) = \sum_{e\in E}\binom{|e|}{2}$. Keep the change if the modularity can be increased and discard the change otherwise. Repeat the iterative process by moving any node in V from its own cluster to its neighbors' clusters until the modularity gain in one scan of V is no greater than a threshold ϵ. Then we compress all the nodes in each cluster into a supernode. On the compressed graph, we repeat the above node movement and graph compression steps until the modularity gain is no greater than ϵ. Figure 3 shows a flow diagram to illustrate the concrete steps of PIC algorithm.

Algorithm 2 shows the PI clustering method. For a given hypergraph $H(V,E)$, Line 1 initializes the singleton clustering. incSum accumulates the modularity gain in each iteration and indicates a termination if it falls below ϵ after an iteration (Line 6). γ defined in Section 3 is computed in Line 2. cids records the cluster id of each node and vols stores the volume of each node. These two parameters are global – they will be, by default, implicitly passed to any function called by Algorithm 2. Line 3 computes the adjacency matrix A. Line 4-5 initializes the indices for computation optimization, we leave its elaboration to Section 5.3.

Line 6-25 follows a Louvain framework. Each iteration (Line 8-24) repeatedly traverses each node $u\in V$, attempting to improve the modularity by mobilizing u from its own cluster C to its neighbor's cluster C', until the accumulated modularity gain in one scan of V falls below ϵ. After each iteration, the graph is compressed – each cluster into a supernode – with all information updated (Line 25). The iteration stops when the gain of an iteration falls below ϵ (Line 6). We abuse C (also for C') to denote both the cluster itself and the cluster id (cid). Given a node u, NbrClusters() in Line 12 finds a collection CC of the clusters of all the u's neighbors by checking the adjacency list of u. EdgeContributions() in Line 13 visits every node in u's incident hyperedges to compute the support gain by moving u to each of u's neighbors' clusters. It thus computes, for all the clusters C' in CC, the total support $\Delta supts[C']$ from all the edges incident on u to C'. Now we take two steps, step 1 moves u from its current cluster C and step 2 places u in another cluster C'. Line 14 calculates

Algorithm 2: PIC

Input: $H(V, E)$, parameters θ, ϵ : termination condition, two booleans rule-1 and rune-2: switches of the two optimization techniques proposed in Section 5.3.
Output: clusters $\mathscr{C}$

1 $\mathscr{C} \leftarrow \{\{\}\}$; incSum $\leftarrow +\infty$; $t \leftarrow 0$; $n \leftarrow |V|$;
2 $\gamma \leftarrow$calculated by Theorem 1; cids$[v] \leftarrow v$, vols$[v] \leftarrow d_v, \forall v \in V$;
3 $A \leftarrow$adjacency matrix, $A[u]$ records the neighbors of u, for $\forall u$;
4 **if** rule-1 **then** lastTime $\leftarrow$initR1Index() // OPT Rule 1;
5 **if** rule-2 **then** $\overline{\text{hlyt}}$, edgeChgs $\leftarrow$ initR2Index() // OPT Rule 2;
6 **while** incSum $> \epsilon$ **do**
7 incSum $\leftarrow 0$; inc $\leftarrow +\infty$;
8 **while** inc $> \epsilon$ **do**
9 inc $\leftarrow 0$;
10 **for** *each* $u \in V$ **do**
11 $C \leftarrow$ cids$[u]$, $\Delta^* \leftarrow 0$, $C^* \leftarrow C$;
12 CC $\leftarrow$ NbrClusters(u);
13 $\Delta supts \leftarrow$ EdgeContributions(u);
14 $\Delta\text{PI}_{C\rightarrow u} \leftarrow$ - deltaPI($u, C \setminus \{u\}, \Delta supts[C]$);
15 **for** *each* $C' \in$ CC **do**
16 $\Delta\text{PI}_{u\rightarrow C'} \leftarrow$ deltaPI($u, C', \Delta supts[C']$);
17 $\Delta \leftarrow \Delta\text{PI}_{u\rightarrow C'} + \Delta\text{PI}_{C\rightarrow u}$;
18 **if** $\Delta > \Delta^*$ **then** $\Delta^* \leftarrow \Delta$, $C^* \leftarrow C'$;
19 **if** $\Delta^* > 0$ **then**
20 inc+ = Δ^*, vols $[C]-= d_u$, vols $[C^*]+= d_u$, cids$[u] \leftarrow C^*$;
21 **if** rule-1 **then** lastTime$[C] \leftarrow$ lastTime$[C^*] \leftarrow t$;
22 **if** rule-2 **then**
23 **for** each e on u **do** edgeChgs$[e]+ = w(u, e)$;
24 incSum+ = inc;
25 $H, A \leftarrow$ Compress(cids, H, A); Update γ, cids, and vols (Line 2);
26 **return** $\mathscr{C}$ generated by adding u to $\mathscr{C}$[cids$[u]$] for all $u \in V$;
27 **Function** deltaPI(u, C, $\Delta supt$) from Eqn 6-7 **return**
28 $\frac{\Delta supt}{|E|} + (1-\gamma)\left(\frac{(1-\eta_C)^2}{1-\gamma+\gamma\eta_C} + \frac{(1-\eta_{\{u\}})^2}{1-\gamma+\gamma\eta_{\{u\}}} - \frac{(1-\eta_{C\cup\{u\}})^2}{1-\gamma+\gamma\eta_{C\cup\{u\}}}\right)$;

the modularity gain of step 1 by calling the function deltaPI which leverages Equations 6-7. For each cluster C' in CC (Line 15), the modularity gain is computed similarly (Line 16). The total modularity gain Δ updates the maximum gain Δ^* (Line 17-18) and when there is a positive gain, we make the move (Line 19-20). The resulting clustering $\mathscr{C}$ can be extracted using cids (Line 26).

Lemma 5. *The time complexity of Algorithm 2 is* $O(\text{vol}(H) \cdot |\tilde{e}| + \text{vol}_2(H))$, *where* $|\tilde{e}|$ *is the average edge cardinality in the graph.*

Proof. Given a node u as input, NbrClusters() iterates each adjacent node of u and EdgeContributions() visits every node in u's incident hyperedges. Thus, for a round of Algorithm 2's process (Line 10-23), NbrClusters() and EdgeContributions() take $O(\text{vol}_2(H))$ and $O(\text{vol}(H) \cdot |\tilde{e}|)$ time, respectively. Plus the total $O(\text{vol}_2(H))$ time taken to visit neighbor clusters of each of n nodes

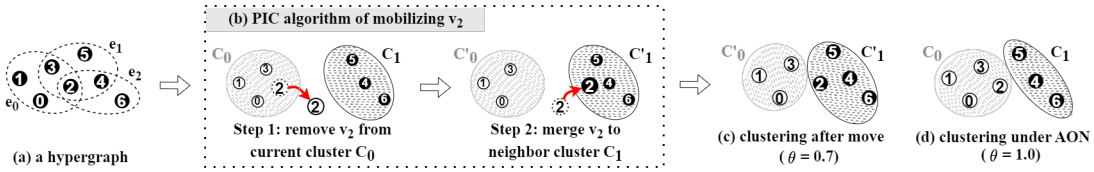


Fig. 4. An example to illustrate the step of mobilizing a node in PIC algorithm and the computation of PI modularity.

in Line 15-18 and $O(\mathrm{vol}(H) \cdot |\tilde{e}|)$ for graph compression in Line 25, the total time complexity of Algorithm 2 is thus $O(\mathrm{vol}(H) \cdot |\tilde{e}| + \mathrm{vol}_2(H))$. □

Example 2. *Figure 4 shows intermediate steps of* PIC *in clustering a hypergrpah H. Figure 4(a) shows H, Figure 4(b) shows two clusters of H before the step: $C_0 = \{v_0, v_1, v_2, v_3\}$ and $C_1 = \{v_4, v_5, v_6\}$. Let $\rho(l) = \frac{l}{\log(1/l+1)}$ and $\theta = 0.7$. The process of moving v_2 from C_0 to C_1 and the corresponding modularity gain calculation is shown below.*

- *Step 1:* PIC *removes v_2 from its current cluster C_0, the modularity gain $\Delta\mathsf{PI}_{C_0 \to v_2}$ is calculated with Equation (7):* $\Delta\mathsf{PI}_{C_0 \to v_2} =$

$$-(1-\gamma)\left(\frac{(1-\eta_{C_0\setminus\{v_2\}})^2}{1-\gamma+\gamma\eta_{C_0\setminus\{v_2\}}} + \frac{(1-\eta_{\{v_2\}})^2}{1-\gamma+\gamma\eta_{\{v_2\}}} - \frac{(1-\eta_{C_0})^2}{1-\gamma+\gamma\eta_{C_0}}\right)$$
$$-\frac{\mathrm{supt}_\theta(C_0) - \mathrm{supt}_\theta(\{v_2\}) - \mathrm{supt}_\theta(C_0 \setminus \{v_2\})}{|E|} = -0.06,$$

  where $\mathrm{supt}_\theta(C_0) = \rho(l(e_0, C_0)) = \frac{1}{\log(1+1)} = 1$. *Similarly computed:* $\mathrm{supt}_\theta(\{v_2\}) = 0$ *and* $\mathrm{supt}_\theta(C_0 \setminus \{v_2\}) = 0.61$. *Then* $\eta_{C_0} = \theta(1 - \frac{\mathrm{vol}(C_0)}{\mathrm{vol}(H)}) = 0.7 \times (1 - 1.83/3) = 0.27$ *where* $\mathrm{vol}(C_0) = 1.83$ *and* $\mathrm{vol}(H) = 3$. *Similarly we have* $\eta_{\{v_2\}} = 0.51$ *and* $\eta_{C_0\setminus\{v_2\}} = 0.47$.
- *Step 2:* PIC *merges v_2 to its neighbor cluster C_1, the modularity gain $\Delta\mathsf{PI}_{v_2 \to C_1} = 0.44$ is calculated by Equation (6), similar to Step 1.*

The total modularity gain incurred from the move of v_2 is $\Delta\mathsf{PI} = \Delta\mathsf{PI}_{C_0 \to v_2} + \Delta\mathsf{PI}_{v_2 \to C_1} = -0.06 + 0.44 = 0.38 > 0$. *Thus, v_2 moves to C_1 and Figure 4(c) shows the updated clustering $\mathscr{C}' = \{C_0', C_1'\}$.*

Now consider the PI *modularity of the clustering in Figure 4(c).*

$$\mathsf{PI}(C_0') = \frac{1}{|E|}\left(\mathrm{supt}_\theta(C_0') - |E|(1-\eta_{C_0'})^2\left(1 + \frac{\gamma\eta_{C_0'}}{1-\gamma}\right)^{-1}\right)$$
$$= \frac{1}{3} \times \left(0.61 - 3 \times (1-0.47)^2 \times \left(1 + \frac{0.63 \times 0.47}{1-0.63}\right)^{-1}\right) = 0.04$$

based on Equation (5), where $\mathrm{supt}_\theta(C_0') = \rho(l(e_0, C_0')) = \frac{3/4}{\log(4/3+1)} = 0.61$, $\eta_{C_0'} = 0.47$ *and* $\gamma = 0.63$. *We compute* $\mathsf{PI}(C_1') = 0.11$ *similarly. The* PI *modularity of the clustering $\mathscr{C}'$ in Figure 4(c) is thus*

$$\mathsf{PI}_H(\mathscr{C}') = \mathsf{PI}(C_0') + \mathsf{PI}(C_1') = 0.04 + 0.11 = 0.15.$$

Besides our clustering result, we give another clustering result under the AON contribution by letting $\theta = 1.0$ in Figure 4(d) for comparison. In this scenario, v_2 will not move to C_1 as the modularity gain is negative: $\Delta\mathsf{PI} = \Delta\mathsf{PI}_{C_0 \to v_2} + \Delta\mathsf{PI}_{v_2 \to C_1} = -0.19 + 0.16 = -0.03 < 0$. *It is because after moving v_2 from C_0 to C_1, not all nodes of e_0 and e_1 belong to cluster $C_0 \setminus \{v_2\}$ or cluster $C_1 \cup \{v_2\}$, thus e_0 and e_1 can contribute to neither of these two clusters due to the AON constraint. Intuitively, the clustering in*

Figure 4(c) is more reasonable, as v_2 has a tighter connection with v_4, v_5, v_6 through hyperedges e_1, e_2. This shows the benefit of non-AON contribution for hypergraph clustering.

5.3 Optimizations

The proof of Lemma 5 indicates that the bottleneck of PIC is on traversing i) adjacency lists to check each neighbor cluster (Line 15, Algorithm 2) and/or ii) a large number of incident edges and their nodes (in EdgeContributions()). We alleviate the bottleneck by designing three indices for skipping the clusters which have not been changed for a certain time and the non θ-innercluster hyperedges. We set time t, a counter of the number of nodes examined in Line 1 of Algorithm 2, and design three indices in Line 4-5.

(1) lastTime: Maintains the latest change time of each cluster for the purpose of skipping the cluster of u's neighbor v if v's cluster has not been changed in the last scan. A cluster is changed if there is node addition or removal.
(2) $\overline{\mathsf{hlyt}}$: Records the historical maximum loyalty of each hyperedge for skipping the traversal of non θ-innercluster hyperedges in EdgeContributions(). For each edge e, we keep $\overline{\mathsf{hlyt}}[e].l = \max_C l(e, C)$, the maximum loyalty of e to any cluster C. For any e, $\overline{\mathsf{hlyt}}[e]$ is maintained in a lazy manner, updated only when e is visited and the condition of Line 7, Algorithm 4 is met.
(3) edgeChgs: Used to calculate the upper bound of the loyalty of a hyperedge e to any cluster C, $l(e, C)$, with the purpose of reducing the update frequency of $\overline{\mathsf{hlyt}}[e]$. Given an edge e and the current time t, assume the last update of $\overline{\mathsf{hlyt}}[e]$ is done at t' and denote $C' = \arg\max_C l(e, C)$ at t'. Let $\Delta V \subseteq e$ be the set of nodes in e whose cluster memberships have changed during the time $(t', t]$. We maintain $\mathsf{edgeChgs}[e] = \sum_{u \in \Delta V} w(u, e)$. Let $C^* = \arg\max_C l(e, C)$ at time t, then we have $l(e, C^*) \leq \overline{\mathsf{hlyt}}[e].l + \mathsf{edgeChgs}[e]$ – if all nodes in ΔV are moved to C' during $(t', t]$, the equality is established and C^* is C'.

Lemma 6. *Let t be the current time of Algorithm 2 and u be a node in cluster C_u. For any neighbor $v \in A[u]$ in cluster C_v, denote by time t'_u and t'_v the last time C_u and C_v were updated (Line 20), respectively. If $t - t'_u > n$ and $t - t'_v > n$ (i.e., $t - t'_u > n$ means there are more than n nodes examined since time t'_u where $n = |V|$), then u will not be moved to C_v at time t.*

Proof. Suppose u is moved to C_v at time t, we have $\Delta\mathsf{PI}_{C_u \to u} + \Delta\mathsf{PI}_{u \to C_v} > \Delta\mathsf{PI}_{C_u \to u} + \Delta\mathsf{PI}_{u \to C_u}$. The previous visit to u was done at time $t - n$ during the last round of Line 10. With $t - t'_u > n$ and $t - t'_v > n$, t'_u and t'_v are both earlier than $t - n$, thus neither C_u nor C_v has been updated since (incl.) $t - n$. Thus we have $\Delta\mathsf{PI}_{C_u \to u} + \Delta\mathsf{PI}_{u \to C_v} > \Delta\mathsf{PI}_{C_u \to u} + \Delta\mathsf{PI}_{u \to C_u}$ at $t - n$, then u should have been moved to C_v at $t - n$, contradicting the fact that $u \in C_u$ at t and both C_u and C_v have not been changed since (incl.) $t - n$. □

Lemma 7 proves the correctness of loyalty upper bound which is then used to skip the traversal of non θ-innercluster hyperedges in the merge node u and cluster C.

Lemma 7. *Let t be the current time of Algorithm 2. Given a node u and a cluster C, consider the merge of u and C. For any incident edge e of u, when calculating the support that u and C receive from e, we have $\bar{l} = \left(w(u, e) + \overline{\mathsf{hlyt}}[e].l + \mathsf{edgeChgs}[e]\right)$ as the upper bound of the maximum loyalty of e to $C \cup \{u\}$ at time t, that is, for any cluster C at time t, $\bar{l} \geq l(e, C \setminus \{u\}) + w(u, e)$.*

Proof. Given a node u, an incident edge e of u, and any cluster C, we have 2 cases at time t. Case one (when $u \in C$) we have $C \cup \{u\} = C$ so $l(e, C \setminus \{u\}) + w(u, e) = l(e, C)$. As $\overline{\mathsf{hlyt}}[e].l + \mathsf{edgeChgs}[e] \geq l(e, C)$ and $w(u, e) > 0$, we have $\bar{l} = \overline{\mathsf{hlyt}}[e].l + \mathsf{edgeChgs}[e] + w(u, e) > l(e, C)$ as desired. Case 2 (when $u \notin C$), we have $C \setminus \{u\} = C$ so $l(e, C \setminus \{u\}) + w(u, e) = l(e, C) + w(u, e)$. As

Algorithm 3: NbrClusters

Input: Node u
Output: Set of distinct neighbor clusters of u, CC

```
1  CC ← [];
2  if rule-1 & t − lastTime[cids [u]] > n then
3      for each v ∈ A[u] do
4          if cids[v] ∉ CC & t − lastTime[cids[v]] ≤ n then CC.add(cids[v]);
5  else for each v ∈ A[u] do if cids[v] ∉ CC then CC.add(cids[v]);
6  return CC;
```

Algorithm 4: EdgeContributions

Input: Node u
Output: Changes in support after and before the move, $\Delta supts[]$

```
1  Δsupts ← [];
2  for each incident edge e of u do
3      if rule 2 & w(u,e) / Σ_{u∈e} w(u,e) + hlyt[e].l + edgeChgs[e] < θ then
4          continue;
5      lyts← [];
6      for each v ∈ e do if v ≠ u then lyts[cids[v]]+ = w(v, e);
7      if rule-2 & edgeChgs[e] > 0 then
8          hlyt[e] ← maxLoyalty(u, e, lyts), edgeChgs ← 0;
9          if w(u, e) + hlyt[e].l < θ then continue;
10     l_1 ← w(u, e);
11     for each C′ that e has positive loyalty with do
12         l_2 ← lyts[C′], l_3 ← l_1 + l_2;
13         Δsupts[C′] += ρ(l_3) − ρ(l_1) − ρ(l_2);
14 return Δsupts;
```

$\overline{\mathsf{hlyt}}[e].l + \mathsf{edgeChgs}[e] \geq l(e, C)$, by adding $w(u, e)$ on both sides, we have $\bar{l} \geq l(e, C) + w(u, e) = l(e, C \setminus \{u\}) + w(u, e)$. □

Based on Lemmas 6-7, our two optimizations are as follows.

- Rule 1: skip the settled clusters. As Lemma 6 suggests, we call a cluster that has not been updated in the last scan (n ticks in time) 'settled'. Algorithm 3 can exclude a cluster from CC with a simple test to alleviate the bottleneck in Line 15-18 of Algorithm 2. Specifically, for a node u and a neighbor $v \in A[u]$, if both C_u and C_v are settled, we can safely exclude C_v from CC. Index lastTime is maintained in Line 21: if there is a node added/removed from a cluster C at time t, lastTime$[C]$ is set to t.
- Rule 2: skip the neutral edges. As Lemma 7 suggests, given a node u and for an incident edge e of u, if $w(u, e) + \overline{\mathsf{hlyt}}[e].l + \mathsf{edgeChgs}[e] < \theta$, then we call e 'neutral'. With the loyalty upper bound, we can skip an edge from the calculation of contribution $\Delta supts$ (Line 13, Algorithm 2) and alleviate the bottleneck incurred in EdgeContributions(). Specifically, for a node u and an incident edge e of u, if the loyalty upper bound of e, $\bar{l} < \theta$, then we can safely skip the calculation of the contributions from e. Index edgeChgs is maintained in Line 23: if a node u changes its cluster id, the value of edgeChgs$[e]$ is increased by $w(u, e)$ for all incident edges e of u.

	Time	Space
LOUV [7]	$O(\tilde{\lvert e\rvert} \cdot \text{vol}(H) + \text{vol}_2(H))$	$O(\text{vol}_2(H))$
IRMM [39]	$O(m^2n + mn^2)$	$O(mn + m^2 + n^2)$
CNMR [34]	$O(mn^2)$	$O(\text{vol}(H))$
CNMO [34]	$O(m^2n^2)$	$O(\text{vol}(H))$
HMLL [20]	$O(\tilde{d_G} \cdot \tilde{\lvert e\rvert} \cdot \text{vol}(H) + \text{vol}_2(H))$	$O(\text{vol}(H) + \text{vol}_2(H))$
HPPR [58]	$O(\lvert\mathscr{C}\rvert \cdot \log \text{vol}(H) \cdot \tilde{\lvert e\rvert} \cdot \text{vol}(H))$	$O(\text{vol}(H))$
EDVW [30]	$O(m^2n + mn^2 + n^3)$	$O(m^2 + mn + n^2)$
PBCC [61]	$O((m+n) \cdot \text{vol}(H) + (m+n)^2)$	$O(\text{vol}(H))$
PIC (ours)	$O(\tilde{\lvert e\rvert} \cdot \text{vol}(H) + \text{vol}_2(H))$	$O(\text{vol}(H) + \text{vol}_2(H))$

Table 1. Time and Space Complexitiy

The functions NbrClusters() and EdgeContributions() with the above optimization techniques are presented in Algorithm 3 and Algorithm 4, respectively. Algorithm 3 applies Rule 1 in computing CC: when the switch rule-1 in Algorithm 2 is on, Line 2-4 skips the settled clusters. Algorithm 4 applies Rule 2 in computing the support array $\Delta supts$ in Line 13: if the switch rule-2 is on, Line 3 and 9 skip the neutral edge e. Indices $\overline{\text{hlyt}}$ and edgeChgs are maintained in Line 8. If e passes the test, in Line 10-13, we first calculate e's loyalty to $\{u\}$, $C' \setminus \{u\}$ and C' stored in variables l_1, l_2 and l_3 respectively, then calculate the support of e to the merge of $\{u\}$ and $C' \setminus \{u\}$.

6 RELATED WORK

Random Graph Model. By comparing a given graph G with its corresponding random graph, a clustering algorithm detects densely innerconnected clusters. For dyadic graphs, two existing random graph models are Erdős-Rényi model [25] and configuration model [10, 21, 23]. The former preserves the number of edges in G and the latter further preserves the degree sequence of G. Planted partition model [64] specifies, additionally, the number k of communities and parameters p and q: edges across and within communities are chosen with probability q and p, resp. For a hypergraph H, existing work turns H into a dyadic graph [7, 38, 39] or a bipartite graph [19, 34, 41] before applying a random graph model. A recently proposed random hypergraph model [52] retains the core-periphery structure of H. We are the first to challenge the necessity of preserving the precise cardinality sequence to scale up non-AON contribution computation in random hypergraph model.

Graph Clustering and Modularity. Graph clustering [26, 55] has been extensively studied on dyadic graphs. With fitness measures such as modularity [21], conductance [15], normalized cut [56], etc., clusters can be found via optimization. Exact modularity optimization is computationally hard, leading to approximation approaches [14, 22, 24, 50].

Resolution Limit. Modularity, with other quality functions that are mathematically similar to it, i.e., based on global optimization of intra- and extra-community links and on a comparison to null model, has resolution limit [27, 40]: clusters smaller than a certain scale may not be identified. In other words, the clusters found by modularity maximization algorithms on large networks may have hidden sub-clusters that require deeper investigations to reveal. Existing solutions to the resolution limit in the literature include adopting a tunable resolution parameter [12, 54] and constructing a hierarchical structure [14] which allows a user to zoom in the clustering; the latter applies to our PIC via iteratively compressing the clusters into supernodes. As argued by [14], such multi-level structure allows a user to uncover and reveal the intermediate clustering with proper resolution by zooming the hierarchical structure.

Hypergraph Clustering. The following work studies hypergraph clustering with different clustering fitness measures. [58] connects conductance with personalized page rank, [30, 66] generalizes

spectral method in minimizing the normalized cut, [61] proposes a bipartite correlation clustering objective and proves to be related to normalized cut and modularity, [44, 45] studies various hyperedge weight functions to reflect different structural importance of hyperedges. The main drawback of the spectral method is that it assumes that the users know the number of clusters beforehand, without knowing which, the performance can drop dramatically [62]. Our paper proposes a hypergraph modularity function under a newly designed random hypergraph model.

Most existing modularity-based hypergraph clustering methods are Louvain-like. LOUV [7] and IRMM [38, 39] apply clique reduction to convert a hypergraph H to a dyadic graph G. LOUV [7] applies Louvain [14] directly to G with unweighted edges. IRMM [38, 39] extends LOUV by reweighing each dyadic edge after each pass. Specifically, it assigns each hyperedge e a weight of 1 initially and assigns the corresponding dyadic edges a weight of $1/(|e|-1)$. After a pass of Louvain, it readjusts the weight of every hyperedge e based on the clusters of nodes in e and then the weight of dyadic edges in G. [34] proposes two clustering methods CNMR and CNMO which iteratively select a hyperedge e and then merge the clusters of all the nodes in the edge. CNMR selects e randomly while CNMO follows a prioritized order in selecting e. They both adopt the AON contribution. A modularity-based generative model (BMM-like) was proposed by [20] upon which algorithm HMLL optimizes the generative function in a Louvain manner. Table 1 lists the complexities of existing hypergraph clustering methods. $\tilde{d}_G$ denotes the average node degree in the dyadic graph G (after clique reduction). $|\tilde{e}|$ denotes the average edge cardinality in H. $|\mathscr{C}|$ denotes number of clusters a method find. The time complexity of our PIC and LOUV is the lowest among the 9 clustering methods.

Recently, an extensive study has been conducted on local hypergraph clustering [9, 29, 46, 48, 60, 64] which finds a cluster biased to a (set of) given node(s) by optimizing local clustering functions.

Matrix-based Graph Clustering. Matrix-based clustering such as spectral clustering cannot replace graph-based clustering due to its computation complexity. For example, spectral clustering [8] has two phases: affinity matrix construction takes $O(n^2m)$ time while eigendecomposition takes $O(n^3)$ time, where $n = |V|$ and $m = |E|$. For the large-scale hypergraphs (n can be up to 15 million) tested in this paper, the complexity of spectral clustering can be prohibitive. Though matrix sparsification and sub-matrix construction [32] may reduce the time by introducing approximation [28], many graph clustering applications still resort to graph-based methods for better effectiveness and scalability.

7 EXPERIMENTATIONS

This section evaluates the performance of our proposed PIC method on 16 real-world hypergraphs. Table 2 shows the statistics of them. We compare PIC with 8 state-of-the-art hypergraph clustering methods introduced in Section 6: LOUV [7] (code provided by [16, 34]), IRMM [39], CNMO [34], CNMR [34], HMLL [20], HPPR [58], EDVW [30], and PBCC [61].

The algorithms were implemented in Java with library JavaSE-9. All experiments were conducted on a machine with Intel XeonE5-2697 CPU, 504GB main memory and Linux(centos), engaged one core for all the algorithms. All algorithms were run 20 times to report the average. The cut-off running time was set to be 6 hours.

Parameters. As our experiments suggested that the clustering performance of PIC is insensitive to ϵ, we followed Scikit network [16] in setting $\epsilon = 10^{-3}$ for Louvain-style baselines LOUV, IRMM, and PBCC as well as PIC. For the setting of parameter θ, we train a predictive model on selected hypergraph features under the following guidelines. i) Select features that can collectively capture the cardinality distribution which has a significant influence on θ: to allow hyperedges with high cardinalities to contribute to the clustering, θ should be smaller; if the cardinalities are small, then

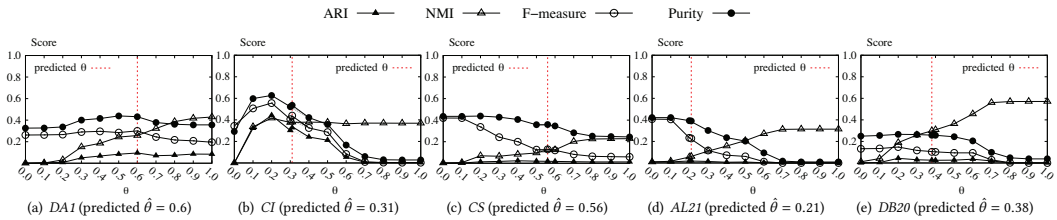
Name	Data set	n	m	$\tilde{\lvert e \rvert}$	vol(H)	vol$_2$(H)	Has Ground Truth	# of Classes
SO	Stack Overflow [4]	15,211,989	1,103,218	23.7	26,146,267	29,306,268,738	False	
HS	Threads Stack [13]	2,301,086	8,578,968	2.6	22,624,290	41,990,824	False	
AR	Amazon Reviews [3]	2,268,192	4,242,398	17.2	72,855,752	6,064,824,282	False	
CD	Coauth DBLP [13]	1,659,954	2,093,835	3.5	7,252,705	16,238,552	False	
CG	Coauth Geology [13]	903,793	834,152	4.0	3,315,150	10,706,346	False	
AS	Amazon Sport [3]	695,253	1,946,419	3.7	7,264,704	45,663,728	False	
DB21	DBLP 2021 [1]	528,315	291,371	4.4	1,277,736	5,142,290	True	2,977
DB20	DBLP 2020 [1]	516,230	292,801	4.3	1,251,107	4,922,964	True	3,221
AX20	Arxiv 2020	352,443	116,219	6.4	743,044	33,810,830	False	
AA	Amazon Art [3]	221,004	383,934	3.8	1,461,184	11,365,750	False	
AL21	ACL 2021 [2]	5,295	2,079	4.8	9,959	42,678	True	11
AL20	ACL 2020 [2]	3,969	1,599	4.6	7,401	28,910	True	9
CS	Coauth DBLP Small [13]	3,965	4,278	2.7	11,388	17,164	True	14
CI	Citeseer Cociting [5]	1,318	597	4.7	2,800	26,468	True	6
DA1	Drug Abuse Network 1 [6]	273	692	2.9	1,982	2,480	True	6
DA0	Drug Abuse Network 0 [6]	226	578	2.9	1,665	2,058	True	6

Table 2. Data Sets

	F-measure								**Purity**							
	DA0	*DA1*	*CI*	*CS*	*AL21*	*AL20*	*DB20*	*DB21*	*DA0*	*DA1*	*CI*	*CS*	*AL21*	*AL20*	*DB20*	*DB21*
LOUV	0.202	0.225	0.203	0.066	0.046	0.047	0.018	0.018	**0.402**	**0.416**	0.306	0.258	0.143	0.153	**0.256**	**0.277**
IRMM	0.213	0.231	0.316	0.069	0.052	0.050	\	\	0.386	0.402	0.451	0.262	0.164	**0.177**	\	\
CNMO	**0.224**	**0.254**	**0.342**	\	\	\	\	\	0.354	0.410	**0.487**	\	\	\	\	\
CNMR	0.010	0.004	0.058	0.003	0.014	0.020	\	\	0.092	0.072	0.122	0.042	0.075	0.101	\	\
HMLL	0.065	0.059	0.003	**0.090**	0.000	0.000	0.000	0.000	0.291	0.285	0.028	**0.277**	0.009	0.010	0.013	0.026
HPPR	0.105	0.097	0.241	0.006	**0.133**	0.006	**0.130**	**0.145**	0.341	0.318	0.270	0.052	**0.174**	0.034	0.248	0.273
EDVW	0.120	0.101	0.177	0.075	0.052	**0.072**	\	\	0.352	0.303	0.271	0.117	0.105	0.117	\	\
PBCC	0.052	0.048	0.069	0.001	0.004	0.004	0.000	0.000	0.223	0.200	0.145	0.025	0.030	0.032	0.092	0.106
PIC	**0.304**	**0.303**	**0.446**	**0.129**	**0.229**	**0.100**	**0.105**	**0.106**	**0.450**	**0.435**	**0.535**	**0.359**	**0.393**	**0.298**	**0.257**	**0.275**
	ARI								**NMI**							
	DA0	*DA1*	*CI*	*CS*	*AL21*	*AL20*	*DB20*	*DB21*	*DA0*	*DA1*	*CI*	*CS*	*AL21*	*AL20*	*DB20*	*DB21*
LOUV	**0.055**	0.082	0.144	0.012	0.006	0.004	**0.007**	**0.007**	0.245	0.263	0.354	0.092	0.080	0.078	0.215	0.202
IRMM	0.053	0.088	0.226	0.011	0.006	0.005	\	\	0.243	0.271	0.362	0.088	0.078	0.074	\	\
CNMO	0.031	**0.097**	**0.246**	\	\	\	\	\	0.322	0.348	**0.380**	\	\	\	\	\
CNMR	0.006	0.002	0.041	0.001	0.004	**0.006**	\	\	**0.582**	**0.573**	0.378	**0.309**	0.298	0.265	\	\
HMLL	0.028	0.025	0.002	**0.016**	0.000	0.000	0.000	0.000	0.496	0.451	0.369	0.226	**0.314**	**0.284**	**0.576**	**0.562**
HPPR	0.026	0.031	0.048	0.002	**0.009**	0.002	0.000	0.001	0.325	0.334	0.252	0.211	0.163	0.199	0.006	0.023
EDVW	0.036	0.027	0.114	0.003	0.003	0.005	\	\	0.364	0.362	0.332	0.231	0.173	0.214	\	\
PBCC	0.022	0.022	0.052	0.001	0.003	0.002	0.000	0.000	**0.518**	**0.511**	0.372	**0.299**	**0.306**	**0.279**	**0.610**	**0.604**
PIC	**0.086**	**0.099**	**0.340**	**0.018**	**0.020**	**0.007**	**0.026**	**0.015**	0.296	0.265	**0.389**	0.124	0.065	0.087	0.305	0.294

Table 3. Clustering Quality of Different Methods

a larger θ is preferred. ii) Keep the number of hypergraph features small and the predictive model simple to avoid overfitting because among the datasets we've collected, only 8 have ground truth. iii) Select features that are irrelevant to the ground truth clustering and use *leave-one-out* strategy on the datasets with ground truth to facilitate a fair comparison, i.e., use 7 out of 8 datasets for training with the target θ value achieving the best overall performance (found through grid search) on the training hypergraphs, and then use the trained model to predict θ for clustering the remaining hypergraph and evaluating the quality. Strictly following the above guidelines, we selected two features, the average hyperedge cardinality $\tilde{|e|}$ and λ (the parameter introduced in Section 3 that is calculated from the hypergraph volume and the number of hyperedges), and a simple linear regression model to determine θ. To choose a loyalty function ρ, we used the above *leave-one-out* strategy similarly. We chose Linear-over-Logarithm as the default loyalty function: averaged over 4 measures and 8 datasets, the performance using Linear-over-Logarithm is 1.3%, 10.2%, and 14168% better than that using Quadratic, Exponential, and AON functions.



(a) *DA1* (predicted $\hat{\theta} = 0.6$) (b) *CI* (predicted $\hat{\theta} = 0.31$) (c) *CS* (predicted $\hat{\theta} = 0.56$) (d) *AL21* (predicted $\hat{\theta} = 0.21$) (e) *DB20* (predicted $\hat{\theta} = 0.38$)

Fig. 5. Sensitivity Test on θ

7.1 Effectiveness

We evaluate the clustering quality in the alignment to the ground truth clustering with four widely used metrics: F-measure [49], Purity [49], Adjusted Rand Index (ARI) [33], and Normalized Mutual Information (NMI) [35].

Exp 1. Table 3 shows the clustering performance of PIC and the 8 baselines on 8 datasets with ground truth. Top-3 scores for each dataset are highlighted with bold&underline, bold, and underline, resp. '\' denotes no result due to time-out or out-of-memory reason. We set θ in PIC with the predictive model. On F-measure, PIC outperforms all 8 baselines (in top-down order as listed in Table 3 unless otherwise specified) by 222%, 107%, 29%, 2871%, 3923%, 516%, 160% and 3732%, resp, averaged over all datasets. On purity, PIC outperforms all 8 baselines by 50%, 48%, 15%, 435%, 1492%, 208%, 134% and 531%, resp. On ARI, PIC is 120%, 77%, 73%, 1507%, 4404%, 520%, 286% and 620% higher than the 8 baselines. On NMI, PIC outperforms LOUV, IRMM, and HPPR by 18%, 12%, and 754%, resp. CNMR, HMLL, and PBCC gain higher NMI than our PIC because they report very small-sized clusters that are preferred by NMI empirically with bias, as explained in the next paragraph.

CNMR, HMLL and PBCC tend to find small-sized clusters, which has also been observed by [20]. In terms of the average size, CNMR, HMLL and PBCC have only 1.2, 1.6 and 2.5 nodes per cluster, resp. The almost singleton-sized clusters found may not conform to the real communities [42]. The tendency can also be shown in terms of the maximum cluster size, on *DB21*, that of HMLL and PBCC is only 19 and 242 while that of ground truth and ours is 141, 649 and 166, 990. The reason is that CNMR and PBCC can stop early in the iterative merge of clusters and HMLL adopts AON hyperedge contribution which makes the hyperedges with relatively large cardinality hard to contribute to the forming of large clusters. It shows that our partial (non-AON) contribution function well addresses the issue and reports more reasonable clusters.

Exp 2. We perform a sensitivity test and evaluate the clustering quality of PIC when the parameter θ varies from 0 to 1 with step size 0.1 and that of the predicted θ (denoted as $\hat{\theta}$). Figure 5 reports the F-measure, purity, ARI, and NMI on 5 data sets (due to the space limit, we omit the results on *DA0*, *AL20*, and *DB21* which are similar to those on *DA1*, *AL21*, and *DB20*, resp.). $\hat{\theta}$ and the corresponding performances are highlighted with a red vertical line.

When θ varies from 0 to 1, different measures show different trends and fall in different value ranges. NMI increases with θ but F-measure, purity and ARI drop at a large θ. Take *DA1* as an example, Figure 5(a) shows that when θ increases, NMI increases steadily while the other three measures first rise, reach the peak at $\theta = 0.6$, then drop. The value ranges of F-measure, ARI, purity and NMI are quite different: 0.20-0.30, 0-0.10, 0.33-0.44, and 0-0.43, resp..

Visually $\hat{\theta}$ seems to achieve a good balance of the four measures. We want to further *quantitatively* evaluate how good the overall performance is by $\hat{\theta}$ compared with the best performance by grid search. However, we cannot simply aggregate the four measures under the same θ as the overall

\	Time (sec.)															
	DA0	*DA1*	*CI*	*CS*	*AL20*	*AL21*	*AA*	*AX20*	*DB20*	*DB21*	*AS*	*CG*	*CD*	*AR*	*HS*	*SO*
LOUV	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.03	10.14	10.56	3.98	4.09	71.48	14.62	28.30	2,277.88	75.20	\
IRMM	1.53	1.82	7.50	1,253.65	163.39	416.87	\	\	\	\	\	\	\	\	\	\
CNMO	13.09	28.01	382.31	\	\	\	\	\	\	\	\	\	\	\	\	\
CNMR	**0.01**	**0.01**	0.78	94.25	30.32	60.10	\	\	\	\	\	\	\	\	\	\
HMLL	0.07	0.08	3.68	0.53	0.79	1.78	2,856.77	8,795.43	1,191.89	4,151.22	12,028.23	5,694.31	11,616.99	\	\	\
HPPR	1.66	2.05	1.18	30.05	8.54	14.31	\	4,043.35	11,590.39	13,476.95	\	\	\	\	\	\
EDVW	0.64	0.96	46.11	3,109.80	2,148.09	18,233.85	\	\	\	\	\	\	\	\	\	\
PBCC	20.69	25.84	9.56	35.42	19.50	27.64	\	\	503.81	521.98	\	\	\	\	\	\
PIC	**0.01**	**0.01**	**0.01**	0.02	0.02	0.04	7.76	39.77	4.49	4.53	23.39	16.89	28.87	1,451.29	58.14	13,548.25
PIC1	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.02**	6.25	23.53	3.21	3.36	19.95	12.86	24.57	978.28	48.42	6,877.25
PIC2	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.03	5.31	18.11	3.50	3.59	20.35	12.62	24.47	740.55	49.37	6,195.13
PIC12	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.02**	**4.72**	**9.24**	**3.02**	**3.16**	**19.49**	**12.08**	**24.32**	**665.56**	**47.98**	**5,204.51**

Table 4. Time Cost of Different Methods

performance, because different measures have different value ranges and one measure can dominate the others. Thus we use a scaling function to transform the original measures into the same range for fair aggregation.

Specifically, let $x_{k,\theta}$ denote one of the four measures at θ, where $k \in \{0, 1, 2, 3\}$ and $\theta \in I = \{0, 0.1, \ldots, 1\} \cup \{\hat{\theta}\}$. Denote by $x_{k,max} = \max_{\theta \in I} x_{k,\theta}$ the highest score and $x_{k,min} = \min_{\theta \in I} x_{k,\theta}$ the lowest score. We transform each measure score using the equation below:

$$q_{k,\theta} = \frac{\log(x_{k,\theta}/x_{k,min})}{\log(x_{k,max}/x_{k,min})}, \text{ for } k \in \{0, 1, 2, 3\} \text{ and } \forall \theta \in I.$$

In other words, we use $x_{k,min}$, $x_{k,max}$ and log function to turn each $x_{k,\theta}$ to a relative score $q_{k,\theta}$ in the range of $[0, 1]$. $q_{k,\theta} = 0$ when $x_{k,\theta} = x_{k,min}$ and $q_{k,\theta} = 1$ when $x_{k,\theta} = x_{k,max}$. Through this function, the four measures are transformed to the same range, and then we compute the average score for each θ by $q_\theta = \text{avg}_{k \in \{0,1,2,3\}} q_{k,\theta}$ to represent the overall performance.

We measure the quality of the predicted $\hat{\theta}$ with the loss function, $LOSS = (\max_{\theta \in I} q_\theta) - q_{\hat{\theta}}$, which indicates the gap between the best quality over all grid searched θ and that of the predicted $\hat{\theta}$. The lower the loss, the higher the quality of $\hat{\theta}$. Averaged over the 8 datasets, the loss is 0.03. This shows the effectiveness of our predictive model for setting θ – the overall clustering performance with the predicted $\hat{\theta}$ is close to or even the same as the best performance, e.g., on dataset *DA1*, the loss is 0 which means that the predicted $\hat{\theta}$ achieves the best performance as by grid search.

7.2 Scalability

Exp 3. Table 4 shows the time costs of PIC and the baselines on 16 datasets. To quantify the impact of our optimization techniques on time cost, we conduct an ablation study by creating three variants of our method: PIC denotes our algorithm without optimization, PIC1 uses only the first optimization, PIC2 uses the second, and PIC12 uses both. When averaged over all datasets, PIC1, PIC2 and PIC12 are 29%, 27% and 36% faster than PIC, resp., and PIC12 achieves the largest efficiency improvement. These improvements show the effectiveness of both optimization techniques.

Baselines LOUV, IRMM, CNMO, CNMR, HMLL, HPPR, EDVW, PBCC are 0.6, 27271, 14112, 2589, 417, 1510, 240374, 1600 times slower than PIC12, resp, on average. In particular, PIC12 is five orders of magnitude faster than IRMM on *CS* and than EDVW on *CS*, *AL20*, and *AL21*. LOUV failed on *SO* with an out-of-memory error. The other baselines failed on many large hypergraphs with out-of-time errors. The result echoes the time complexity in Table 1.

Exp 4. Table 5 shows the memory costs of all methods. The space for storing and preprocessing the graph is not counted. We first conduct an ablation study to quantify the impact of our optimization techniques on memory cost. The memory costs of PIC and its variants are very close. PIC1, PIC2,

\	Memory (MB)															
	DA0	*DA1*	*CI*	*CS*	*AL20*	*AL21*	*AA*	*AX20*	*DB20*	*DB21*	*AS*	*CG*	*CD*	*AR*	*HS*	*SO*
LOUV	0.14	0.17	1.57	1.37	2.00	2.89	632.47	1,846.29	322.01	335.12	2,518.80	675.27	1,057.26	324,165.2	2,507.00	\
IRMM	3.44	4.94	18.97	324.28	163.90	288.87	\	\	\	\	\	\	\	\	\	\
CNMO	**0.03**	**0.03**	**0.08**	\	\	\	\	\	\	\	\	\	\	\	\	\
CNMR	**0.03**	**0.03**	**0.08**	**0.27**	**0.22**	**0.30**	\	\	\	\	\	\	\	\	\	\
HMLL	0.32	0.38	1.43	2.81	2.59	3.59	559.05	1189.33	404.77	416.50	2,356.99	898.03	1,694.01	\	\	\
HPPR	0.11	0.13	0.27	1.04	0.76	1.02	\	**69.36**	**113.37**	**115.86**	\	\	\	\	\	\
EDVW	7.70	11.13	121.18	1,358.50	1,078.48	1,913.91	\	\	\	\	\	\	\	\	\	\
PBCC	0.30	0.36	0.62	2.59	1.74	2.31	\	\	266.34	270.75	\	\	\	\	\	\
PIC	0.11	0.13	0.40	0.88	0.75	1.04	**163.26**	308.09	119.64	122.96	**707.80**	**275.66**	**537.39**	**49,277.73**	**1,504.04**	**225,429.26**
PIC1	0.11	0.13	0.40	0.89	0.77	1.06	164.10	309.43	121.61	124.97	710.45	279.10	543.72	49,286.38	1,512.82	225,487.29
PIC2	0.12	0.14	0.41	0.94	0.78	1.07	169.11	309.86	124.11	127.40	737.50	288.38	569.34	49,342.46	1,634.95	225,446.10
PIC12	0.12	0.14	0.41	0.96	0.79	1.09	169.96	311.21	126.08	129.42	740.15	291.83	575.67	49,351.11	1,643.73	225,504.13

Table 5. Memory Cost of Different Methods

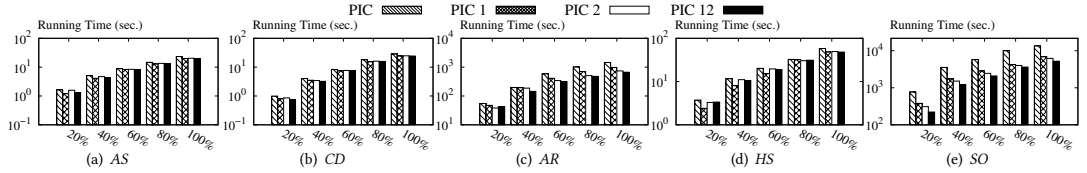


Fig. 6. Scalability: Time Cost by Varying Node Number

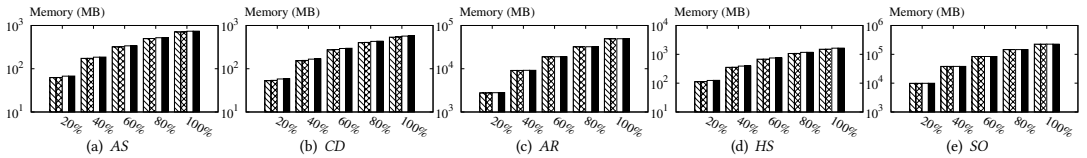


Fig. 7. Scalability: Memory Cost by Varying Node Number

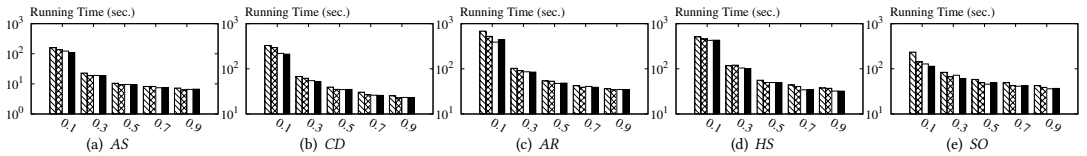


Fig. 8. Scalability: Time Cost on Cardinality Distributions (vary λ)

and PIC12 only use 1%, 4%, and 5% more memory than PIC, averaged over all datasets. This shows that the space overhead to maintain the additional indices for optimizations is negligible, but the improvement in efficiency brought by them is quite notable.

Compared with baselines, the memory cost of PIC is much less than that of LOUV, IRMM, HMLL, PBCC and EDVW. Though CNMO, CNMR, and HPPR take less memory space than PIC on the smaller hypergraphs, they suffer from high running time and fail with out-of-time errors on seven or more larger hypergraphs. The memory cost conforms to the space complexity in Table 1.

Exp 5. Figures 6-7 show the time and memory costs of PIC when varying the number of nodes of the graph. Due to the space limit, we only show the results on 5 largest datasets. We randomly divided the nodes of a graph into 5 groups (each 1/5 of nodes) and created 5 graphs with the i-th graph the induced subgraph of the first i groups of nodes. The experiments were performed on the 5 graphs.

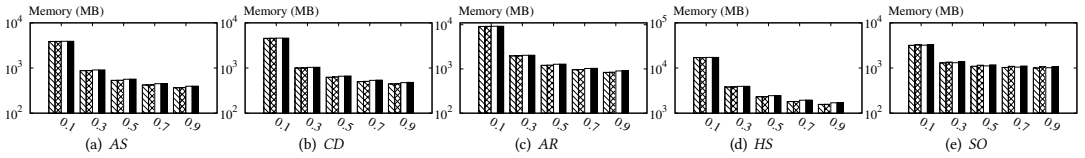


Fig. 9. Scalability: Memory Cost on Cardinality Distributions (vary λ)

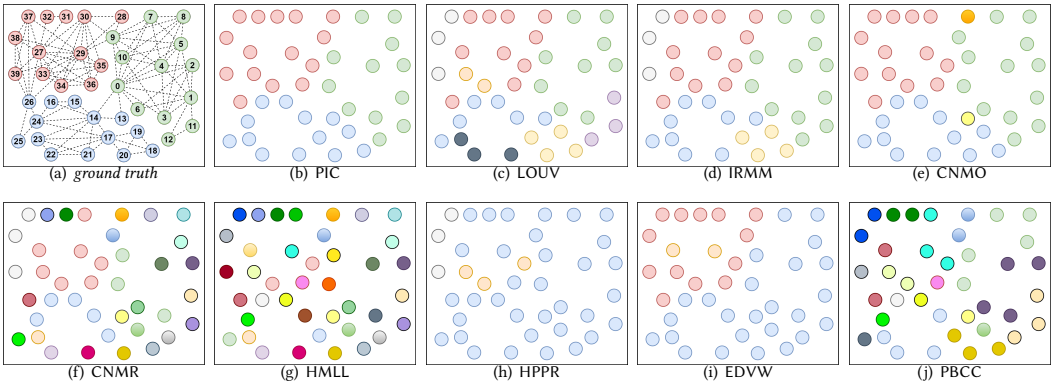


Fig. 10. Case Study

Figure 6 shows that the time cost increases not linear to the number of nodes but instead, empirically linear to $\text{vol}(H) \cdot |\tilde{e}| + \text{vol}_2(H)$ (i.e., the time complexity of PIC). Figure 7 shows that the memory cost increases not linear to the node number, but empirically linear to $\text{vol}(H) + \text{vol}_2(H)$ (i.e., the space complexity of PIC).

Exp 6. Figure 8 and Figure 9 show the time and memory costs of PIC when varying the edge cardinality distribution: we fix the node size n, edge size m, node degree distribution, and change λ (a parameter in the exponential model depicting the edge cardinality distribution in Section 3) from 0.1 to 0.9. We observed that with other factors fixed, the smaller the λ, the larger the average cardinality $|\tilde{e}|$, and thus the higher the volume (in terms of either $\text{vol}(H)$ or $\text{vol}_2(H)$) of the induced graphs, e.g., $|\tilde{e}|$ of the graph with $\lambda = 0.1$ is about 3 times larger than that of the graph with $\lambda = 0.9$.

Figure 8 shows that the time costs decrease with λ, not linear to λ but empirically linear to $\text{vol}(H) \cdot |\tilde{e}| + \text{vol}_2(H)$. Figure 9 shows that the memory costs decrease, empirically linear to $\text{vol}(H) + \text{vol}_2(H)$. This result echoes the complexity of PIC listed in Table 1.

7.3 Case Study

To show the effectiveness and meaningfulness of our clustering method, we conduct a case study on a subgraph of the real hypergraph *DB21*. The subgraph, as shown in Figure 10, consists of 40 nodes representing authors and 35 hyperedges representing the groups of authors who published a paper together in the year 2021. There are three ground truth clusters as shown in Figure 10(a), corresponding to three conferences: *pvldb* (in red), *icml* (in green), and *emnlp* (in blue). Each author belongs to one cluster which is determined by the conference in which he/she published the most papers in 2021. There is a dashed line between two nodes if and only if they share at least one hyperedge. Figures 10(b)-(j) show the clustering results of our method PIC and 8 baseline methods.

We observe that PIC produces the most accurate clustering compared with the baselines. LOUV and IRMM in Figures 10(c)-(d) break the red and blue clusters into several smaller ones. CNMO in Figure 10(e) reports two extra singleton clusters. CNMR, HMLL and PBCC in Figures 10(f), (g) and (j) generate many small-sized clusters. HPPR produces a giant blue cluster in Figure 10(h) which is inaccurate, and EDVW merges the green and blue clusters into a single one by mistake in Figure 10(i).

8 CONCLUSIONS

This paper proposes a scalable modularity-based hypergraph clustering approach that can effectively capture the non-AON hyperedge-cluster relation. Our experiments show that PIC outperforms the state-of-the-art methods on real-world hypergraphs in terms of both clustering quality and scalability and is up to five orders of magnitude faster than the baseline methods.

ACKNOWLEDGMENTS

Zijin Feng and Hong Cheng are supported in part by NSFC Grant No. U1936205, project #MMT-p2-23 of the Shun Hing Institute of Advanced Engineering, The Chinese University of Hong Kong and by grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14217622). Miao Qiao is supported by Marsden Fund UOA1732 and NZ-Singapore Data Science Research Programme UOAX2001, the Catalyst: Strategic Fund from Government Funding, administrated by MBIE, New Zealand.

REFERENCES

[1] 2021. XML release of DBLP. https://dblp.org/xml/release/.
[2] 2022. ACL Anthology. https://aclanthology.org/.
[3] 2022. Amazon Review Data. https://nijianmo.github.io/amazon/index.html.
[4] 2022. ARB dataset. https://www.cs.cornell.edu/~arb/data/.
[5] 2022. LINQS dataset. https://linqs.soe.ucsc.edu/data.
[6] 2022. The Substance Abuse and Mental Health Services Administration. https://www.samhsa.gov/.
[7] Sameer Agarwal, Jongwoo Lim, Lihi Zelnik-Manor, Pietro Perona, David J. Kriegman, and Serge J. Belongie. 2005. Beyond Pairwise Clustering. In *CVPR*. 838–845. https://doi.org/10.1109/CVPR.2005.89
[8] Charu Aggarwal and Chandan Reddy. 2013. *Data clustering: algorithms and applications.* https://doi.org/10.1201/9781315373515
[9] Ali Aghdaei, Zhiqiang Zhao, and Zhuo Feng. 2021. HyperSF: Spectral Hypergraph Coarsening via Flow-based Local Clustering. In *ICCAD*. 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643555
[10] William Aiello, Fan Chung, and Linyuan Lu. 2000. A random graph model for massive graphs. In *STOC*. 171–180. https://doi.org/10.1145/335305.335326
[11] Ilya Amburg, Nate Veldt, and Austin Benson. 2020. Clustering in graphs and hypergraphs with categorical edge labels. In *WWW*. 706–717. https://doi.org/10.1145/3366423.3380152
[12] Alex Arenas, Alberto Fernandez, and Sergio Gomez. 2008. Analysis of the structure of complex networks at different resolution levels. *New journal of physics* 10, 5 (2008), 053039. https://doi.org/10.1088/1367-2630/10/5/053039
[13] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proc. Natl. Acad. Sci.* 115, 48 (2018), E11221–E11230. https://doi.org/10.1073/pnas.1800683115
[14] Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast Unfolding of Communities in Large Networks. *J. Stat. Mech.* 2008, 10 (04 2008), P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008
[15] Béla Bollobás and Bela Bollobas. 1998. *Modern graph theory*. Vol. 184. Springer Science & Business Media. https://doi.org/10.1007/978-1-4612-0619-4
[16] Thomas Bonald, Nathan de Lara, Quentin Lutz, and Bertrand Charpentier. 2020. Scikit-network: Graph Analysis in Python. *J. Mach. Learn.* 21, 185 (2020), 1–6.
[17] Alain Bretto. 2013. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer* (2013). https://doi.org/10.1007/978-3-319-00080-0
[18] Jiajun Bu, Shulong Tan, Chun Chen, Can Wang, Hao Wu, Lijun Zhang, and Xiaofei He. 2010. Music recommendation by unified hypergraph: combining social media information and music content. In *ACM Multimedia*. 391–400. https://doi.org/10.1145/1873951.1874005

[19] Philip S Chodrow. 2020. Configuration models of random hypergraphs. *J. Complex Networks* 8, 3 (08 2020), cnaa018. https://doi.org/10.1093/comnet/cnaa018

[20] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. 2021. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances* 7, 28 (2021), eabh1303. https://doi.org/10.1126/sciadv.abh1303

[21] Fan Chung and Linyuan Lu. 2002. The average distances in random graphs with given expected degrees. *Natl Acad. Sci.* 99, 25 (2002), 15879–15882. https://doi.org/10.1073/pnas.252631999

[22] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70, 6 (2004), 066111. https://doi.org/10.1103/PhysRevE.70.066111

[23] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703. https://doi.org/10.1137/070710111

[24] Jordi Duch and Alex Arenas. 2005. Community detection in complex networks using extremal optimization. *Phys. Rev. E* 72, 2 (2005), 027104. https://doi.org/10.1103/PhysRevE.72.027104

[25] Paul Erdős, Alfréd Rényi, et al. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 1 (1960), 17–60.

[26] Santo Fortunato. 2010. Community detection in graphs. *Phys. Reps.* 486, 3-5 (2010), 75–174. https://doi.org/10.1016/j.physrep.2009.11.002

[27] Santo Fortunato and Marc Barthelemy. 2007. Resolution limit in community detection. *Proceedings of the national academy of sciences* 104, 1 (2007), 36–41. https://doi.org/10.1073/pnas.0605965104

[28] Santo Fortunato and Darko Hric. 2016. Community detection in networks: A user guide. *Phys. Reps.* 659 (2016), 1–44. https://doi.org/10.1016/j.physrep.2016.09.002

[29] Kimon Fountoulakis, Pan Li, and Shenghao Yang. 2021. Local hyper-flow diffusion. *NeurIPS* 34 (2021), 27683–27694.

[30] Koby Hayashi, Sinan G Aksoy, Cheong Hee Park, and Haesun Park. 2020. Hypergraph random walks, laplacians, and clustering. In *CIKM*. 495–504. https://doi.org/10.1145/3340531.3412034

[31] Einar Hille and Ralph Saul Phillips. 1996. *Functional analysis and semi-groups*. Vol. 31. American Mathematical Soc.

[32] Dong Huang, Chang-Dong Wang, Jian-Sheng Wu, Jian-Huang Lai, and Chee-Keong Kwoh. 2019. Ultra-scalable spectral clustering and ensemble clustering. *TKDE* 32, 6 (2019), 1212–1226. https://doi.org/10.1109/TKDE.2019.2903410

[33] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *J. Classif.* 2, 1 (1985), 193–218. https://doi.org/10.1007/BF01908075

[34] Bogumił Kamiński, Valérie Poulin, Paweł Prałat, Przemysław Szufel, and François Théberge. 2019. Clustering via hypergraph modularity. *PloS one* 14, 11 (2019), e0224307. https://doi.org/10.1371/journal.pone.0224307

[35] Min-Soo Kim and Jiawei Han. 2009. A particle-and-density based evolutionary clustering method for dynamic networks. *VLDB* 2, 1 (2009), 622–633. https://doi.org/10.14778/1687627.1687698

[36] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Yoo. 2011. Higher-order correlation clustering for image segmentation. *NIPS* 24 (2011), 1530–1538.

[37] Larkshmi Krishnamurthy, Joseph Nadeau, Gultekin Ozsoyoglu, M Ozsoyoglu, Greg Schaeffer, Murat Tasan, and Wanhong Xu. 2003. Pathways database system: an integrated system for biological pathways. *Bioinformatics* 19, 8 (2003), 930–937. https://doi.org/10.1093/bioinformatics/btg113

[38] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. 2019. A New Measure of Modularity in Hypergraphs: Theoretical Insights and Implications for Effective Clustering. In *Complex Networks*, Vol. 881. 286–297. https://doi.org/10.1007/978-3-030-36687-2_24

[39] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. 2020. Hypergraph clustering by iteratively reweighted modularity maximization. *Appl. Netw. Sci.* 5, 1 (2020), 52. https://doi.org/10.1007/s41109-020-00300-3

[40] Jussi M Kumpula, Jari Saramäki, Kimmo Kaski, and János Kertész. 2007. Limited resolution in complex network community detection with Potts model approach. *Eur. Phys. J. B* 56 (2007), 41–45. https://doi.org/10.1140/epjb/e2007-00088-4

[41] Geon Lee, Minyoung Choe, and Kijung Shin. 2021. How Do Hyperedges Overlap in Real-World Hypergraphs? - Patterns, Measures, and Generators. In *WWW*. 3396–3407. https://doi.org/10.1145/3442381.3450010

[42] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123. https://doi.org/10.1080/15427951.2009.10129177

[43] Lei Li and Tao Li. 2013. News recommendation via hypergraph learning: encapsulation of user behavior and news content. In *WSDM*. 305–314. https://doi.org/10.1145/2433396.2433436

[44] Pan Li and Olgica Milenkovic. 2017. Inhomogeneous Hypergraph Clustering with Applications. In *NIPS*, Vol. 30. 2308–2318.

[45] Pan Li and Olgica Milenkovic. 2018. Submodular hypergraphs: p-laplacians, cheeger inequalities and spectral clustering. In *ICML*. 3014–3023.

[46] Wentao Li, Miao Qiao, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2022. On Scalable Computation of Graph Eccentricities. In *SIGMOD*. 904–916. https://doi.org/10.1145/3514221.3517874
[47] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. 2019. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *DAC*. 1–6. https://doi.org/10.1145/3316781.3317803
[48] Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F Gleich. 2021. Strongly local hypergraph diffusions for clustering and semi-supervised learning. In *WWW*. 2092–2103. https://doi.org/10.1145/3442381.3449887
[49] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press. https://doi.org/10.1017/CBO9780511809071
[50] Mark EJ Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* 74, 3 (2006), 036104. https://doi.org/10.1103/PhysRevE.74.036104
[51] Mark E. J. Newman. 2010. *Networks: An Introduction*. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001
[52] Marios Papachristou and Jon Kleinberg. 2022. Core-Periphery Models for Hypergraphs. In *KDD*. 1337–1347. https://doi.org/10.1145/3534678.3539272
[53] Emad Ramadan, Arijit Tarafdar, and Alex Pothen. 2004. A hypergraph model for the yeast protein complex network. In *IPDPS*. 189. https://doi.org/10.1109/IPDPS.2004.1303205
[54] Jörg Reichardt and Stefan Bornholdt. 2006. Statistical mechanics of community detection. *Physical review E* 74, 1 (2006), 016110. https://doi.org/10.1103/PhysRevE.74.016110
[55] Satu Elisa Schaeffer. 2007. Graph clustering. *Comput. Sci. Rev.* 1, 1 (2007), 27–64. https://doi.org/10.1016/j.cosrev.2007.05.001
[56] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *TPAMI* 22, 8 (2000), 888–905. https://doi.org/10.1109/34.868688
[57] Sucheta Soundarajan and John E. Hopcroft. 2012. Using community information to improve the precision of link prediction methods. In *WWW*. 607–608. https://doi.org/10.1145/2187980.2188150
[58] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. 2020. Hypergraph Clustering Based on PageRank. In *KDD*. 1970–1978. https://doi.org/10.1145/3394486.3403248
[59] Bertrand Thirion, Gaël Varoquaux, Elvis Dohmatob, and Jean-Baptiste Poline. 2014. Which fMRI clustering gives good brain parcellations? *Front. Neurosci.* 8 (2014), 167. https://doi.org/10.3389/fnins.2014.00167
[60] Nate Veldt, Austin R Benson, and Jon Kleinberg. 2020. Minimizing localized ratio cut objectives in hypergraphs. In *KDD*. 1708–1718. https://doi.org/10.1145/3394486.3403222
[61] Nate Veldt, Anthony Wirth, and David F Gleich. 2020. Parameterized correlation clustering in hypergraphs and bipartite graphs. In *KDD*. 1868–1876. https://doi.org/10.1145/3394486.3403238
[62] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17 (2007), 395–416. https://doi.org/10.1007/s11222-007-9033-z
[63] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. 2012. A model-based approach to attributed graph clustering. In *SIGMOD*. 505–516. https://doi.org/10.1145/2213836.2213894
[64] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *KDD*. 555–564. https://doi.org/10.1145/3097983.3098069
[65] Chen Zhe, Aixin Sun, and Xiaokui Xiao. 2019. Community Detection on Large Complex Attribute Network. In *KDD*. 2041–2049. https://doi.org/10.1145/3292500.3330721
[66] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NeurIPS*. 1601–1608. https://doi.org/10.7551/mitpress/7503.003.0205

Received January 2023; revised April 2023; accepted May 2023