Comparison of modularity-based approaches for nodes clustering in hypergraphs

Veronica Poda[1] and Catherine Matias[2]

[1] University of Trento, Via Sommarive, 14, 38123, Povo, Italy. Email: very.poda@gmail.com

[2] Sorbonne Université, Université de Paris Cité, Centre National de la Recherche Scientifique, Laboratoire de Probabilités, Statistique et Modélisation, 4 place Jussieu, 75252 PARIS Cedex 05, France. Email: catherine.matias@math.cnrs.fr

Abstract

Statistical analysis and node clustering in hypergraphs constitute an emerging topic suffering from a lack of standardization. In contrast to the case of graphs, the concept of nodes' community in hypergraphs is not unique and encompasses various distinct situations. In this work, we conducted a comparative analysis of the performance of modularity-based methods for clustering nodes in binary hypergraphs.

To address this, we begin by presenting, within a unified framework, the various hypergraph modularity criteria proposed in the literature, emphasizing their differences and respective focuses. Subsequently, we provide an overview of the state-of-the-art codes available to maximize hypergraph modularities for detecting node communities in hypergraphs. Through exploration of various simulation settings with controlled ground truth clustering, we offer a comparison of these methods using different quality measures, including true clustering recovery, running time, (local) maximization of the objective, and the number of clusters detected.

Our contribution marks the first attempt to clarify the advantages and drawbacks of these newly available methods. This effort lays the foundation for a better understanding of the primary objectives of modularity-based node clustering methods for binary hypergraphs.

Keywords: community detection; higher-order interaction; hypergraph; modularity; node clustering

1 Introduction

The interest in higher-order interactions stems from the recognition that many phenomena are inherently more complex than what can be effectively represented by pairwise relationships alone. While graphs model pairwise interactions, hypergraphs generalize this concept by capturing higher-order interactions involving more than two elements. This extension provides a more expressive framework for modeling intricate dependencies and interactions in various fields, ranging from social network analysis (early acknowledged in Simmel, 1950) or co-authorship relations (Roy and Ravindran, 2015) to ecological systems (Muyinda et al.,

2020), neurosciences (Chelaru et al., 2021) or even chemistry (Flamm et al., 2015). We refer to Battiston et al. (2020); Bick et al. (2023); Torres et al. (2021) for recent reviews on higher-order interactions.

With the emergence of hypergraph datasets (see for e.g. Lee et al., 2021) to model higher-order interactions, the question of nodes clustering and, more specifically, the detection of communities in hypergraphs arises. In the context of graphs, the seminal paper by Newman and Girvan (2004) introduced the concept of modularity (commonly known as the Newman-Girvan modularity), paving the way for a flourishing literature on community detection in networks. In the context of hypergraphs, the past few years have witnessed the surge of modularity-based proposals for hypergraph community detection. One of the first challenges is to propose a modularity criterion that measures the extent to which a hypergraph is composed of communities. This raises a more fundamental question: What is a community of nodes in a hypergraph? While in the context of graphs, a community is simply a set of nodes with more within-cluster interactions than between-clusters ones, generalizing that concept to hypergraphs is not immediate. As hypergraph interactions have a heterogeneous size (i.e., the number of nodes they contain), a primary issue is whether one should weigh the links with respect to (wrt) their sizes and put more emphasis on larger hyperedges (see Figure 1 for an illustration). Consequently, various modularity criteria have recently emerged in the literature.

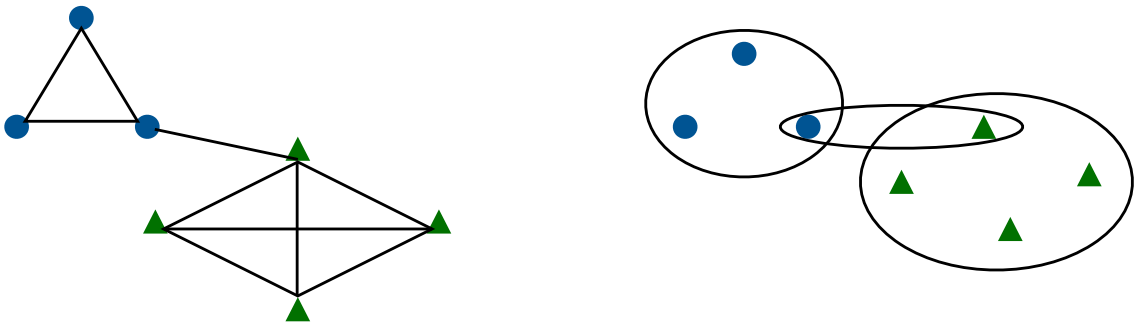

Figure 1: On the left, a modular graph with two clusters is depicted, represented as circle-blue and triangle-green nodes, respectively. In each cluster, the number of within-cluster interactions is much larger than the between-clusters ones. On the right, a hypergraph is shown using the same set of nodes, where each clique from the previous graph is replaced by a hyperedge. In this hypergraph, the number of within-cluster interactions in each of the two clusters is the same as the number of between-clusters interactions. Is this hypergraph modular? Should we consider weighting hyperedges with respect to their sizes to analyze how modular the hypergraph is?

For a long time now, the computer science literature has tackled hypergraphs by simplifying them into graphs, employing two primary methods: the clique reduction graph, also known as the two-section graph, and the star-expansion graph. In the clique reduction graph, each hyperedge of a hypergraph is transformed into a clique in a graph over the same set of nodes (as illustrated in Figure 1, where the graph on the left represents the clique reduction of the hypergraph on the right). Conversely, the star-expansion graph constructs a bipartite graph by treating the original vertex set as the first part and introducing a new vertex for every original hyperedge in a second part. These parts are then connected whenever a node

is contained in a hyperedge in the hypergraph. While the former reduction loses information (the original hypergraph cannot be reconstructed from its clique reduction graph), the latter transformation is one-to-one, given that the two parts are labeled (allowing the distinction between original nodes and original hyperedges) and hypergraphs with self-loops and multiple hyperedges are allowed. Consequently, a natural approach is to define hypergraph modularity by relying on graphs. Figure 1 illustrates the limitations of such a method, where the clique reduction graph (on the left) appears clearly modular, while one may question whether the original hypergraph (on the right) should be considered modular or not.

In this article, we explore the current state-of-the-art and challenges posed by modularity-based community detection methods in binary hypergraphs. In the context of graphs, Yang et al. (2016) propose a comparative analysis of community detection algorithms for undirected and binary graphs. In the same vein, we here restrict our attention to modularity-based methods whose performances for community detection in binary hypergraphs are compared. The methodology is described in Section 2. After introducing general notation, we first present a reformulated version of the different hypergraph modularities existing in the literature (Section 2.2). The goal of this reformulation is to facilitate the comparison of concepts introduced independently from each other and never fully connected before. To be a valuable concept, a hypergraph modularity should come with a (local and/or heuristic) maximization algorithm that outputs a node clustering. Available implementations of such algorithms are presented in Section 2.3. To compare the different modularities and maximization algorithms, it is mandatory to work with synthetic datasets where ground truth clustering is known and hypergraph statistics can be controlled. While in the graph context, recent years have seen the emergence of benchmark datasets for such a task, as for instance the Lancichinetti-Fortunato-Radicchi benchmark graph (LFR, Lancichinetti et al., 2008) used in Yang et al. (2016), there is yet no such benchmark for hypergraphs. We thus rely on several models for generating synthetic modular hypergraphs, described in Section 2.4. Then Section 3 describes our experiments: which scenarios have been explored in each model generating method (Section 3.1) and quality assessment through the lens of different measures, namely true clustering recovery, running time, (local) maximization of the objective and the number of clusters detected (Section 3.2). All the results are presented in Section 4 and a discussion follows in Section 5. The scripts to reproduce the experiments are available online (see details and links in Section 6).

To conclude this introduction, we mention that there are other methods to cluster the nodes of a hypergraph, such as spectral clustering approaches (Ghoshdastidar and Dukkipati, 2017; Chodrow et al., 2023) or model-based methods (Brusa and Matias, 2022a; Ruggeri et al., 2023). It is also possible to cluster hyperedges instead of nodes (Ng and Murphy, 2022). However, our focus in this work is on clustering nodes through modularity-based methods.

2 Material and methods

2.1 General notation and definitions

A hypergraph $H = (V, \mathcal{E})$ is defined as a set of nodes $V = \{1, \dots, n\}$ and a set of hyperedges $\mathcal{E} \subset \mathcal{P}(V)$, where $\mathcal{P}(V)$ is the set of all subsets of V. In other words, each hyperegde $e \in \mathcal{E}$ is a subset of nodes in V (namely, $e \subset V$ or $e \in \mathcal{P}(V)$). A hypergraph can either be binary (presence/absence of subsets of nodes) or *weighted* (also equivalently called *multiple*). In the latter case, the hypergraph $H = (V, \mathcal{E}, w)$ comes with a weight function $w : \mathcal{P}(V) \to \mathbb{N} \cup \{0\}$ such that $\forall e \notin \mathcal{E}$, we have $w(e) = 0$, and $w(e) \in \mathbb{N}$ otherwise. The weight counts how

many times a hyperedge appears in the hypergraph. Multiple (i.e., weighted) hypergraphs can be viewed as hypergraphs where the set of hyperedges $\mathcal{E}$ is allowed to be a multiset (some hyperedges may appear several times). A binary hypergraph is a particular case of a weighted hypergraph with weight function being the indicator function $w(e) = 1\{v \in e\}$ (i.e., each hyperedge has multiplicity 1). The size of a hyperedge e is the number of nodes it contains $|e| = \sum_{v \in V} 1\{v \in e\}$. A hypergraph is said to be *s-uniform* if it only contains hyperegdes of size s. Any 2-uniform hypergraph is simply a graph. We let $\mathcal{E}_s$ denote the subset of $\mathcal{E}$ of hyperedges with size s. We can allow hyperedges $e \in \mathcal{E}$ to be multisets of V, in which case nodes may appear more than once in the same hyperedge. Such hypergraphs are called *multiset* hypergraphs and can either be binary or multiple. In a multiset hypergraph, each node $v \in V$ has a multiplicity in hyperedge $e \in \mathcal{E}$, denoted by $m_e(v) \in \mathbb{N} \cup \{0\}$, which counts the number of times this node appears in that hyperedge. Moreover, the hyperedge size accounts for the nodes multiplicity and becomes $|e| = \sum_{v \in V} m_e(v)$. For example, a self-loop $\{u, u\}$ is a (multiset) hyperedge of size 2. In the following, unless otherwise stated, all sets can be multisets in which case all counts include multiplicities (be it for nodes or for hyperedges). A hypergraph is said *simple* whenever it is binary and non-multiset, i.e., neither nodes or hyperedges may be repeated. The (weighted) degree $\deg_H(v)$ of a node v in a hypergraph H is the (weighted) count of the hyperedges it belongs to, namely $\deg_H(v) = \sum_{e \in \mathcal{E}} w(e)$. The incidence matrix H of the hypergraph has size $|V| \times |\mathcal{E}|$ and entries $H(v, e) = 1\{v \in e\}$ or $m_e(v)$ for multiset hypergraphs. Note that we use the same notation H for a graph and its incidence matrix, the difference should be clear from the context. Letting $w = (w(e))_{e \in \mathcal{E}}$ denote the (column) vector of the hyperedges weights and $w^\intercal$ its transpose, we obtain the vectors of node degrees and hyperedges sizes as Hw and $w^\intercal H$, respectively. Two nodes are said *incident* whenever they belong to a same hyperedge $e \in \mathcal{E}$.

For any subset of nodes $C \subset V$, we define its volume:

$$\mathrm{Vol}_H(C) = \sum_{v \in C} \deg_H(v),$$

and the (weighted) number of hyperedges whose nodes are all included in C:

$$e_H(C) = \sum_{e \subset C} w(e).$$

Note that $\mathrm{Vol}_H(V) = \sum_s s|\mathcal{E}_s|$ and $e_H(V) = |\mathcal{E}| = \sum_s |\mathcal{E}_s|$.

From a (weighted) hypergraph $H = (V, \mathcal{E})$ we may construct its clique reduction graph $G^{\text{clique}} = (V, E)$. This graph has the same set of nodes V as the hypergraph and every hyperedge $e \in \mathcal{E}$ in the hypergraph is *reduced* into a complete clique in the graph. In other words, for any hyperedge $e \in \mathcal{E}$ with size $|e| \geq 2$ and for any pair of incident nodes $u, v \in e$, the graph G^{clique} contains the edge $\{u, v\} \in E$ and only edges obtained in this way are contained in E. The (weighted) adjacency matrix A^{clique} of the clique reduction graph satisfies $A^{\text{clique}} = H\mathrm{diag}(w)H^\intercal$, where H is the incidence matrix of the hypergraph and $\mathrm{diag}(w)$ is the diagonal matrix induced by the vector of hyperedge weights w. In general, self-loops are removed from A^{clique} and A^{clique}_{uu} is set to 0 for any $u \in V$. This can be done directly by setting $A^{\text{clique}} = H\mathrm{diag}(w)H^\intercal - D_V$ where D_V is the diagonal matrix of node degrees $(\deg_H(v))_{v \in V}$.

A nodes clustering is a partition $\mathcal{C} = (C_1, \ldots, C_K)$ of the set of nodes V into parts called clusters. For any partition $\mathcal{C} = (C_1, \ldots, C_K)$ of the set of nodes V and any subset $e \subset V$, we

let $e \cap \mathcal{C} = (e_1, \dots, e_J)$ denote the partition of the subset e induced by $\mathcal{C}$. It has J parts with $J \leq K$ and is indeed a partition of e, namely

$$\forall j \neq k \in \{1, \dots, J\}, \quad e_j \neq \emptyset, \qquad e_j \cap e_k = \emptyset, \qquad \cup_{j=1}^{J} e_j = e.$$

The *adjacent* clusters of a node $u \in V$ are the parts C_k that contain at least one node $v \in C_k$ that is incident to u, or in other words such that there is a hyperedge $e \in \mathcal{E}$ such that $u, v \in e$.

In this manuscript, the identity matrix is denoted by I (its size should be clear from the context). We already used notation $|S|$ for the cardinality of a set S (or a multiset), and $1\{\mathcal{S}\}$ for the indicator function of an event $\mathcal{S}$.

2.2 Modularities in hypergraphs

Different hypergraph modularity criteria have been proposed in the literature up to now (Kumar et al., 2020; Kamiński et al., 2019a; Kamiński et al., 2021; Chodrow et al., 2021). We recall these different quantities, using a unified presentation that highlights similarities and differences between them. As we will see, these are all constructed in the same way, namely the difference between a first term that is a specific hyperedge count and a second term that in some cases corresponds to the expected value of this count under some null model, and otherwise is a correction term. The differences between the expressions of those hypergraph modularities come from: i) the type of hyperedges that are counted; ii) the null model used for computing the expectation or the correcting term; iii) possible weights to each of these terms.

Kumar et al. (2020)'s definition of hypergraph modularity corresponds to a graph modularity as originally defined in Newman and Girvan (2004) and applied to a specific graph choice. Considering the clique reduction graph of a hypergraph, Kumar et al. noticed that the reduction does not preserve the node degrees: in the clique reduction graph G^{clique}, the degree of a node differs from its initial value in the hypergraph H. Indeed, a simple computation shows that

$$\deg_{G^{\text{clique}}}(v) = \sum_{e \in \mathcal{E}} H(v,e) w(e) (|e| - 1) \geq \sum_{e \in \mathcal{E}} H(v,e) w(e) = \deg_H(v).$$

Thus, Kumar et al. (2020) simply modified the weights in the clique reduction graph to preserve these degrees. Let $D_{\mathcal{E}} = \text{diag}(|e|)_{e \in \mathcal{E}}$ denote the diagonal matrix of the hyperedges sizes. We define the *weighted clique reduction graph* $G^{\text{w-clique}}$ through its adjacency matrix $A^{\text{w-clique}} = (A^{\text{w-clique}}_{uv})_{u,v \in V}$ by

$$A^{\text{w-clique}} = H \text{diag}(w)(D_{\mathcal{E}} - I)^{-1} H^{\intercal} - D_V.$$

The node degrees in this graph $G^{\text{w-clique}}$ are equal to the initial node degrees in the hypergraph H (where self-loops are discarded). This construction is equivalent to saying that for each hyperedge $e \in \mathcal{E}$, we create $G^{\text{w-clique}}$ by forming a total of $\binom{|e|}{2}$ edges with weights $w(e)/(|e| - 1)$, between any pair of nodes incident in the hypergraph H.

Then for any hypergraph $H = (V, \mathcal{E})$ and any partition $\mathcal{C} = (C_1, \dots, C_K)$ of its set of

nodes V, we let

$$Q^{\text{w-clique}}(H,\mathcal{C}) = \frac{1}{2|\mathcal{E}|}\sum_{k=1}^{K}\sum_{u,v\in C_k}\left(A_{uv}^{\text{w-clique}} - \frac{\deg_H(u)\deg_H(v)}{\sum_{i=1}^{n}\deg_H(i)}\right)$$
$$= \frac{1}{2|\mathcal{E}|}\sum_{k=1}^{K}\left(e_{G^{\text{w-clique}}}(C_k) - \frac{(\text{Vol}_H(C_k))^2}{\text{Vol}_H(V)}\right). \quad (1)$$

Note that $Q^{\text{w-clique}}$ ranges in $[-1;1]$. It is an average over all pairs of nodes u, v belonging to the same cluster C_k of the difference between the weighted edge value $A_{uv}^{\text{w-clique}}$ in the weighted clique reduction graph and its expectation under a configuration model (Chung and Lu, 2002) that accounts only for nodes degrees and plays the role of a null model. A high value of modularity $Q^{\text{w-clique}}$ means dense connections in the weighted clique reduction graph $G^{\text{w-clique}}$ between the nodes within the same cluster and sparse connections between nodes in different clusters. Going back to the hypergraph H, that means node pairs $u, v \in V$ belonging to the same cluster participate more in the same hyperedge than node pairs in different clusters.

Kamiński et al. (2019a) introduce a *strict* hypergraph modularity such that only the hyperedges $e \in \mathcal{E}$ entirely included in a same cluster contribute to increasing modularity, which is in sharp contrast with the previous proposal. For any hypergraph $H = (V, \mathcal{E})$ and any partition $\mathcal{C} = (C_1, \ldots, C_K)$ of its set of nodes V, we let

$$Q^{\text{strict}}(H,\mathcal{C}) = \frac{1}{|\mathcal{E}|}\sum_{k=1}^{K}\left(e_H(C_k) - \sum_{s\geq 2}|\mathcal{E}_s|\left(\frac{\text{Vol}_H(C_k)}{\text{Vol}_H(V)}\right)^s\right). \quad (2)$$

Note that Q^{strict} also ranges in $[-1;1]$. Here, the first term inside the sum accounts for the number of hyperedges whose all nodes are within the same cluster. The second term comes from a generalization of the Chung and Lu model to hypergraphs. Again, it plays the role of an expected value of the first term $e_H(C_k)$ under some null model which preserves both node degrees and the (weighted) number $|\mathcal{E}_s|$ of size-s hyperedges. This quantity is called by its authors the *degree tax*.

Kamiński et al. (2021) propose a more general modularity that accounts for the *homogeneity* of each hyperedge, namely, the fraction of its vertices that belong to the largest cluster (provided it is more than 50%). For any subset $C \subset V$, any size $s \geq 2$ and any integer $c \in \{\lfloor s/2 \rfloor + 1, \ldots, s\}$, we let $e_H^{s,c}(C)$ denote the number of size-s hyperedges that have exactly c nodes included in their majority part C. With our previous notation, we have

$$e_H(C) = \sum_{s\geq 2} e_H^{s,s}(C).$$

In the following, $\mathbb{P}(\text{Bin}(s,p) = c) = \binom{s}{c}p^c(1-p)^{s-c}$ is the probability that a Binomial random variable with parameters (s,p) takes the value c. Then for any partition $\mathcal{C} = (C_1, \ldots, C_K)$ of the set of nodes, Kamiński et al. (2021) introduce the modularity

$$Q^{\text{wsc}}(H,\mathcal{C}) = \frac{1}{|\mathcal{E}|}\sum_{k=1}^{K}\sum_{s\geq 2}\sum_{c=\lfloor s/2\rfloor+1}^{s} w_{s,c}\left[e_H^{s,c}(C_k) - |\mathcal{E}_s|\mathbb{P}\left(\text{Bin}\Big(s, \frac{\text{Vol}_H(C_k)}{\text{Vol}_H(V)}\Big) = c\right)\right], \quad (3)$$

where $w_{s,c} \in [0,1]$ are hyper-parameters to be specified. Note that we have

$$\mathbb{P}\left(\mathrm{Bin}\Big(s, \frac{\mathrm{Vol}_H(C_k)}{\mathrm{Vol}_H(V)}\Big) = s\right) = \left(\frac{\mathrm{Vol}_H(C_k)}{\mathrm{Vol}_H(V)}\right)^s$$

so that (2) is a special case of (3) where $w_{s,c} = 1\{c = s\}$. Different setups may be considered for the hyper-parameters $w_{s,c}$ and we focus here on the choices for which an optimisation algorithm is available, namely

$$w_{s,c} = \begin{cases} 1\{c = s\} & \text{strict setting,} \\ 1\{c > s/2\} & \text{majority setting,} \\ c/s 1\{c > s/2\} & \text{linear setting.} \end{cases} \tag{4}$$

As already mentioned, the **strict** setting gives back Q^{strict}, already introduced in (2). For the other 2 settings, we call the corresponding modularities Q^{majority} and Q^{linear}, respectively.

Finally, Chodrow et al. (2021) first defined a general *symmetric* modularity, where for any partition $\mathcal{C}$ of the set of nodes, the contribution of a hyperedge $e \in \mathcal{E}$ to the modularity of this partition is characterized only by the vector $\mathbf{p}$ whose entries p_k count the number of nodes in e belonging to the k-th largest part in $e \cap \mathcal{C}$. It is based on a general affinity function $\Omega : \mathcal{P} \to \mathbb{R}$ that modulates the weight of the contribution of each partition vector $\mathbf{p}$, where the set of partition vectors is

$$\mathcal{P} = \{\mathbf{p} = (p_1, \dots p_J); p_1 \geq \cdots \geq p_J \geq 1, \text{ for some } J \geq 1\}.$$

For instance, a s-tuple of nodes with $s = 7$ that are clustered by a partition $\mathcal{C}$ into the parts $\{v_1\}; \{v_2, v_3\}; \{v_4\}; \{v_5, v_6, v_7\}$ induces the partition vector $\mathbf{p} = (3, 2, 1, 1)$. The symmetric modularity from Chodrow et al. (2021) will thus account for the different clusters counts that compose a hyperedge, treating all the clusters in an exchangeable way. We present the details of this modularity in Section A from the Appendix. Then, the authors consider particular cases of their general symmetric modularity, relying on specific forms of the affinity function Ω (see Table 1 in that reference). However, an implementation of the algorithm for optimising the induced specific modularities is available only for the *all-or-nothing* affinity function on which we focus now.

The all-or-nothing modularity function is defined as:

$$Q^{\text{aon}}(H, \mathcal{C}) = \sum_{k=1}^{K} \sum_{s \geq 2} \hat{\beta}_s \left(\sum_{C'_k \subset C_k; |C'_k| = s} e_H(C'_k) - \hat{\gamma}_s (\mathrm{Vol}_H(C_k))^s \right), \tag{5}$$

where $\hat{\beta}_s$ and $\hat{\gamma}_s$ are parameters estimated from the data. While in general we may expect that both $\hat{\beta}_s, \hat{\gamma}_s > 0$ (see Section B in the Appendix for more details on these parameters), we then recover in this expression a sum of difference terms between a count of specific hyperedges, namely those entirely included in a cluster, and a correcting volume term. The extra parameters $\hat{\beta}_s, \hat{\gamma}_s$ might not seem natural at first. In fact, they appear as the result of an approximate maximum likelihood approach in a specific degree-corrected hypergraph stochastic blockmodel (DCHSBM), in the same way as Newman (2016) did in a graph context.

As a final remark, Chodrow et al. (2021) notice that considering the specific choices $\hat{\beta}_s = 1$ and $\hat{\gamma}_s = |\mathcal{E}_s| / \mathrm{Vol}_H(V)^s$ in their modularity Q^{aon}, they recover (up to a scaling factor and an additional term not depending on the partition $\mathcal{C}$ and which can thus be discarded) the expression of the modularity Q^{strict} from (2). However, they argue that leaving these parameters free (adapting to the data) lends important flexibility to their approach.

Additional comments. We already highlighted similarities and differences between the different modularities defined above. Let us add some more comments.

Two extreme cases are represented by the modularities $Q^{\text{w-clique}}$ and Q^{strict}, the former being less stringent than the latter. Whenever a hyperedge is split by the partition $\mathcal{C}$ into different clusters, it will be ignored by Q^{strict} but as soon as this hyperedge contains at least 2 nodes in the same cluster, the modularity $Q^{\text{w-clique}}$ will account for it. The weakness in $Q^{\text{w-clique}}$ lies in that the exact composition of each hyperedge in nodes falling into the different clusters is captured only through pairs of nodes. The modularity Q^{wsc} represents a compromise between the 2 previous extremes: it accounts for *homogeneous* hyperedges, namely hyperedges such that (at least) half of their nodes fall into a cluster that becomes a majority cluster. In particular, Kamiński et al. (2021) argue that the hyper-parameters $w_{s,c}$ may be chosen so that Q^{wsc} well approximates $Q^{\text{w-clique}}$ because contributions in the latter from parts that contain at most $s/2$ vertices may often be neglected. Finally, the modularity Q^{aon} is as strict as Q^{strict} and focuses on hyperedges with nodes split into a unique cluster by the partition $\mathcal{C}$. As already stressed, the major difference between Q^{strict} and Q^{aon} lies in that the latter, while summing similar differences as the former, weights differently each terms in those differences (with weights adaptive to the data, as they are estimated from these).

Note that possible self-loops in the hypergraph H never contribute to a modularity and may thus be discarded from the dataset. However, we highlight that all these modularities are developed for multiset hypergraphs, where nodes may be repeated in a same hyperedge. In particular, the Chung and Lu null models (for graphs and hypergraphs) used in defining modularities $Q^{\text{w-clique}}, Q^{\text{strict}}$ and Q^{wsc} as well as the DCHSBM underlying the definition of the modularity Q^{aon}, all rely on models for multiset hypergraphs. While it is known in the case of graphs that this is inadequate (Massen and Doye, 2005; Cafieri et al., 2010; Squartini and Garlaschelli, 2011), that assumption has not yet been discussed in the context of hypergraph modularities. It might be that the computational simplifications enabled by this assumption prevent from any attempt not to use it (see for e.g. Section B2 in Supplementary Material from Brusa and Matias, 2022a).

2.3 Modularity maximization methods

In this section, we focus on available implementations for hypergraph nodes clustering through modularity-based methods. We briefly describe the corresponding algorithms and their major characteristics, as well as the options that were chosen for our comparison study. All the algorithms require an initialization, most of the times relying on an initial partition where each node is in its own part, i.e., $\mathcal{C}^{\text{own}} = (\{1\}, \ldots, \{n\})$. We group the different methods by the packages where they can be found. A summary is given in Table 1.

Note that we did not include in our experiments a comparison with methods based on clique reductions. Indeed, Kumar et al. (2020) already did so and concluded that "hypergraph based methods perform consistently better than their clique based equivalents" (end of page 16 in that reference).

HyperNetX package. The `HyperNetX` Python package (PNNL, 2023) contains a `modularity` submodule (see `https://pnnl.github.io/HyperNetX/modularity.html`) including various functions for hypergraph clustering through modularity maximization.

Kumar et al. (2020) propose to maximize their modularity $Q^{\text{w-clique}}$ relying on the popular and fast Louvain algorithm for graphs (Blondel et al., 2008). More precisely, they do not simply apply Louvain algorithm on the graph $G^{\text{w-clique}}$ but rather propose an Iteratively Reweighted Modularity Maximization (`IRMM`) algorithm where they iteratively apply Louvain on a weighted clique reduction graph, and compute new hyperedge weights (see Algorithm 1 in Kumar et al., 2020). The hyperedge re-weighting step puts a larger weight on hyperedges which are cut into more *unbalanced* partition vectors by the current partition $\mathcal{C}$. For example, a size-s hyperedge cut into the partition vector $\mathbf{p} = (s-1, 1)$ (meaning a unique node falls in a cluster different from the majority one) is much more unbalanced than another one cut into the partition vector $\mathbf{p} = (s/2, s/2)$ (namely half of the nodes belong to a first cluster, the other half belonging to a second cluster) and thus gets a larger weight (see Figure 1 in Kumar et al., 2020). By getting a larger weight, it is more likely that the unique node in this hyperedge will join the majority cluster at Louvain's next step. The function `hmod.kumar` implements the `IRMM` algorithm.

The last step refinement (`LSR`) is an algorithm described in Kamiński et al. (2021). This is a general method that starting from an initial partition of the nodes, iteratively moves one vertex at a time (in a random order) to a neighboring cluster whenever it improves Q^{wsc}, until convergence. The authors propose to start by running the `IRMM` on the weighted clique reduction graph, then the resulting partition is used as initialisation in their `LSR` procedure, that aims at maximizing Q^{wsc}. For the specific choices `strict, majority` and `linear` of the hyper-parameters $w_{s,c}$ described above, implementations are provided. The modularity Q^{wsc} is obtained through the function `hmod.modularity` from the `HyperNetX` package and the `LSR` algorithm is implemented in the function `hmod.last_step` from this same package. Both functions contain the 3 different options for hyper-parameters $w_{s,c}$ defined in (4) and the default choice is `linear`. This is this option that we choose for our comparisons.

strictModularity package. Kamiński et al. (2019a) propose a Clauset-Newman-Moore like (`CNM-like`) algorithm to maximize Q^{strict} (see Clauset et al., 2004, for the original CNM algorithm). Starting with partition $\mathcal{C}^{\text{own}}$ where each node is in its own part, this algorithm iterates over the set of hyperedges that are split into more than 2 clusters by the current partition, trying to merge all the parts it touches and looking for a modularity improvement. More precisely, the algorithm comes in two versions. In the first one, a loop over all hyperedges is taken, so that at each step all hyperedges are searched and evaluated for merging. In the second one, a stochastic approach is taken which evaluates at each step just one randomly chosen hyperedge (see Algorithm 1 in Kamiński et al., 2019a, for more details). The stochastic version is computationally less expensive, especially for larger hypergraphs; however it requires to set a maximal number of iterations. In what follows, we choose that second version and set the number of iterations to twice the total number of hyperedges. The implementation is available from Kamiński et al. (2019b), in a mix of Python and Julia files. More precisely, a script `strictModularity.py` contains a "quick" Python implementation that should work on small datasets only, while a Julia function `find_comms` is more generally provided to perform the `CNM-like` algorithm. We rely on the latter in our experiments.

HyperModularity package. Chodrow et al. (2021) propose the Hypergraph Maximum Likelihood Louvain (`HMLL`) algorithm to maximize their symmetric modularity (defined in Section A from the Appendix) and the simpler and faster `AON-HMLL` algorithm for maximizing

the specific all-or-nothing Q^{aon} modularity. Both the `HMLL` and the `AON-HMLL` are implemented in the Julia package `HyperModularity` (Chodrow et al., 2022). However the current version of the `HyperModularity` package does not contain an implementation of an estimation of a general affinity function $\widehat{\Omega}$ that is required to compute the symmetric modularity. That is why we focus on the AON modularity Q^{aon} and the corresponding `AON-HMLL` algorithm.

The `AON-HMLL` algorithm is an iterative algorithm that mimics the standard graph Louvain algorithm in that it starts with initial configuration $\mathcal{C}^{\text{own}}$ (each node is in its own part) and at the first iteration, it greedily moves nodes to adjacent clusters (i.e., clusters that contain incident nodes) until no more improvement of Q^{aon} is possible. The subsequent iterations however differ from Louvain's approach and instead of considering a weighted graph on "supernodes", it greedily moves entire clusters to adjacent ones whenever this improves Q^{aon}. Note that the option `startclusters` from `Simple_AON_Louvain_mod` determines which initial partition is used to estimate the parameters $\hat{\beta}_s, \hat{\gamma}_s$. We rely on `startclusters == "cliquelouvain"` that gives the best results in general.

Function (package or script)	Modularity	Algorithm	Language	Options choices
`hmod.kumar` (`HyperNetx`, PNNL, 2023)	$Q^{\text{w-clique}}$	`IRMM`	Python	Init: $\mathcal{C}^{\text{own}}$
`hmod.last_step` (`HyperNetx`, PNNL, 2023)	Q^{linear}	`LSR`	Python	Init: Output(`IRMM`), $w_{s,c}$= linear
`find_comms` (Kamiński et al., 2019b)	Q^{strict}	`CNM-like`	Julia	Init: $\mathcal{C}^{\text{own}}$, Stochastic version
`Simple_AON_Louvain` (`HyperModularity`, Chodrow et al., 2022)	Q^{aon}	`AON-HMLL`	Julia	Init: $\mathcal{C}^{\text{own}}$, `startclusters == "cliquelouvain"`

Table 1: Summary of functions (with package name and reference) for clustering hypergraphs through modularity-based approaches. We indicate which modularity is maximized by the function (second column), the corresponding algorithm (third column), the implementation language (fourth column) and our option choices (fifth column).

2.4 Synthetic models for binary and modular hypergraphs

To compare the different modularity-based approaches for clustering hypergraphs nodes, it is mandatory to rely on simulations of modular hypergraphs where ground truth clusters are known. As mentioned earlier, there is no single standard method for generating modular hypergraphs, and, to our knowledge, there are two main approaches. The first approach is based on hypergraph stochastic block models, with several variants proposed in the literature. The second approach involves a generalization of the LFR model for graphs (Lancichinetti et al., 2008). We chose to consider two variants of the first approach and the only one that we are aware of in the second approach. A summary of these models is given in Table 2. We highlight the similarities and differences between those different generating models and the characteristics of the hypergraphs generated by those approaches.

In all those models, we fix a number of nodes n, either fix or randomly generate a true number of clusters K (that might depend on n) as well as a true partition of the nodes $\mathcal{C}^{\text{true}} = (C_1^{\text{true}}, \dots, C_K^{\text{true}})$ and a maximal size S of hyperedges.

Hypergraphs with HSBM. We consider datasets simulated under a simple (binary and non-multiset) Hypergraph Stochastic Blockmodel (HSBM, see Brusa and Matias, 2022a) generated through the `R` package `HyperSBM` (Brusa and Matias, 2022b). In this model, we fix the true number of clusters K, their proportions $\pi = (\pi_1, \dots, \pi_K)$ such that $\pi_k \in (0, 1)$ and $\sum_k \pi_k = 1$ and the following parameters, for any $2 \leq s \leq S$,

$$\alpha_s = \mathbb{P}(e \in \mathcal{E}_s | \exists 1 \leq k \leq K, e \subset C_k^{\text{true}}),$$
$$\beta_s = \mathbb{P}(e \in \mathcal{E}_s | \forall 1 \leq k \leq K, e \not\subset C_k^{\text{true}}),$$

so that α_s (resp. β_s) is the probability for a s-tuple of nodes to form a hyperedge given that they belong to the same cluster (resp. given that they are not all in same cluster). The parameters should be chosen in order to ensure that the generated hypergraphs are modular. To this aim, we consider the ratios ρ_s of the number of within-cluster size-s hyperedges over the number of between-clusters size-s hyperedges obtained as:

$$\rho_s = \frac{\alpha_s \sum_{k=1}^K \pi_k^s}{\beta_s (1 - \sum_{k=1}^K \pi_k^s)}.$$

In our simulations, we impose $\rho_s > 1$, with larger values corresponding to more modular hypergraphs. Note that in this setting, the total number of size-s hyperedges is random and has expected value

$$\mathbb{E}(|\mathcal{E}_s|) = \binom{n}{s} \Big[\alpha_s \sum_{k=1}^K \pi_k^s + \beta_s \Big(1 - \sum_{k=1}^K \pi_k^s \Big) \Big].$$

We simulate hypergraphs with decreasing values $\mathbb{E}(|\mathcal{E}_s|)$ when s increases, which is more realistic than the constant case.

Hypergraphs with DCHSBM-like. We consider datasets simulated under the DCHSBM-like generating model proposed by Chodrow et al. (2021). This model relies on a fixed true number of clusters K, balanced clusters $|C_1^{\text{true}}| = \dots = |C_K^{\text{true}}| = n/K$ and equal numbers of size-s hyperedges for $2 \leq s \leq S$; so that for each size s, a total of $|\mathcal{E}|/(S-1)$ hyperedges are drawn. With probability p_s, such a hyperedge is placed on a s-tuple of (distinct) nodes within the same cluster and with probability $1 - p_s$, it is placed on any s-tuple of (distinct) nodes.

The ratio ρ_s of within-cluster over between-clusters size-s hyperedges is random and its expectation is

$$\mathbb{E}(\rho_s) = \frac{p_s + (1 - p_s) c_s}{(1 - p_s)(1 - c_s)}, \qquad \text{where } c_s = \frac{K\binom{\lfloor n/K \rfloor}{s}}{\binom{n}{s}} = \frac{K(\lfloor n/K \rfloor)!(\lfloor n/K \rfloor - s)!}{n!(N-s)!}.$$

Note that the DCHSBM-like generating model has been originally proposed for multisets hypergraphs, where nodes may be repeated in hyperedges, and hyperedges may be multiple. In practice, as we consider sparse hypergraphs where the number of hyperedges is linear wrt the number of nodes, multiple hyperedges are rare. Section C from the Appendix contains some further considerations on the links between parameters in HSBM and DCHSBM-like.

h-ABCD benchmark dataset. Recently, Kamiński et al. (2023b) proposed a hypergraph artificial benchmark for community detection, called h-ABCD, together with a code for generating these modular hypergraphs (Kamiński et al., 2023a). This generating model is an appropriate candidate to compare modularity approaches. In this model, we fix the number of nodes n and we either input a sequence of node degrees or it is sampled from a power-law distribution with some input exponent $\gamma \in (2,3)$ and input minimum/maximum degree values. The true clusters sizes are also either input or sampled from a power-law with some input exponent $\beta \in (1,2)$ and minimum/maximum sizes values. In our case, we choose to fix the cluster sizes so that the number of clusters is fixed rather than random. The model requires a sequence $q = (q_1, \ldots, q_S)$ of weights summing to 1 such that S is the maximal hyperedge size and q_s is the fraction of size-s hyperedges. For instance fixing $q_1 = 0$ prohibits self-loops. The script `abcdh.jl` also handles the proportion of homogeneous hyperedges, where homogeneity is the concept discussed in Section 2.2 when introducing Q^{wsc}. We recall that a homogeneous hyperedge has more than half of its nodes within the same (majority) cluster. Let $\omega_{c,s}$ denote the fraction of homogeneous hyperedges of size s that have exactly $c \geq \lfloor s/2 \rfloor$ nodes belonging to their majority cluster, so that $\sum_{c=\lfloor s/2 \rfloor+1} \omega_{c,s} = 1$. This notation is not to be confused with the weights $w_{s,c}$ introduced in (4). To link it to previously introduced quantities, we remark that $\omega_{c,s} = \sum_{e \in \mathcal{E}_s} \sum_{k=1}^{K} e_H^{s,c}(C_k^{\text{true}})/|\mathcal{E}_s|$. The current implementation handles 3 different options: `linear, strict, majority`, corresponding to the following choices

$$\omega_{c,s} = \begin{cases} (\lceil s/2 \rceil)^{-1} & \text{if majority,} \\ \frac{2c1\{c \geq s/2\}}{(s+\lfloor s/2 \rfloor+1)\lceil s/2 \rceil} & \text{if linear,} \\ 1\{c = s\} & \text{if strict.} \end{cases}$$

Thus in the `majority` setting, a homogeneous hyperedge is randomly drawn among all hyperedges with more than half of their nodes in their majority cluster, while in the `strict` setting, homogeneous hyperedges are exactly within-cluster hyperedges (i.e., all nodes belong to the majority cluster). The `linear` setting spreads the homogeneous hyperedges in a linear fashion across the different values c of the number of nodes in the majority cluster. In that setting, there is thus a larger number of homogeneous hyperedges showing a larger number of nodes in their majority cluster. Having set which hyperedges are homogeneous ones, a mixing parameter $\xi \in (0,1)$ controls for the proportion of the degree of each node that is assigned to non-homogeneous hyperedges. In this generating model, the total number of hyperedges is random and equals

$$|\mathcal{E}| = \frac{\sum_{v \in V} \deg_H(v)}{\sum_s s q_s}.$$

In the strict setting, we can also express the ratio ρ_s of within-cluster over between-cluster size-s hyperedges as

$$\rho_s = \frac{1-\xi}{\xi} \quad \text{for the strict setting of h-ABCD.}$$

3 Experiments

3.1 Scenarios

General principles and base case scenario. Our simulations explore various settings in order to i) highlight the global behaviors of the methods and compare their performances;

Synthetic model	Parameters	Characteristics
HSBM	K; clusters proportions π; within- and between-clusters probabilities α_s, β_s	random clusters sizes; random $\lvert\mathcal{E}_2\rvert, \ldots, \lvert\mathcal{E}_S\rvert$
DCHSBM-like	K; $\lvert\mathcal{E}\rvert$; probability p_s that a hyperedge is placed on a s-tuple of within-cluster nodes	balanced clusters; equal nb of size-s hyperedges $\lvert\mathcal{E}_2\rvert = \cdots = \lvert\mathcal{E}_S\rvert = \lvert\mathcal{E}\rvert/(S-1)$
h-ABCD	degrees (power-law distribution or fixed values); cluster sizes (power-law distribution or fixed values); proportions q_s of size-s hyperedges; setting of homogeneous hyperedges (majority, linear or strict); proportion ξ of node degree assigned to homogeneous hyperedges	random number and cluster sizes; random $\lvert\mathcal{E}_2\rvert, \ldots, \lvert\mathcal{E}_S\rvert$

Table 2: Summary of synthetic models for modular hypergraphs and their characteristics. In all the models, the number of nodes n and the maximal hyperedge size S are chosen by the user. The numbers of clusters K and hyperedges $|\mathcal{E}|$ (resp. size-s hyperedges $\mathcal{E}_s$) can either be fixed or random.

ii) explore which hypergraphs characteristics most impact those performances. In all the settings, we choose to focus on *sparse* hypergraphs, for which the number of hyperedges grows linearly with the number of nodes, as this is a most realistic setting.

We start with "base case" scenarios, called scenarios A, that we defined in the 3 different generating models (HSBM, DCHSBM and h-ABCD). Then we explore other scenarios (called B to F and Z), simulated under the most convenient model to do so, and in which we modify only one characteristic at a time wrt the base case. Each scenario is composed of sub-cases with different samples sizes, comprising in general cases 1 to 6 corresponding to $n \in \{50, 100, 150, 200, 500, 1000\}$. Moreover, in each scenario explored, we randomly generated 25 hypergraphs. Table 3 gives a summary of the scenarios considered and the empirical characteristics of the 25 hypergraphs generated for each of them. In this table, each simulation is summarized through its differing characteristic wrt the base case (namely, scenarios A). For example, ScenB-DCHSBM is a simulation of hypergraphs less sparse than the base case.

Simulation	Scenario	n	$\lvert\mathcal{E}_2\rvert$	$\lvert\mathcal{E}_3\rvert$	$\bar{d}$	$\max(d)$
ScenA-HSBM	A1	50	198	85	13	32
(base case)	A2	100	397	178	13	26
$K = S = 3$	A3	150	592	265	13	28
balanced clusters	A4	200	795	354	13	28
$\rho_s = 1.7$; $\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.7$	A5	500	1990	885	13	28
ScenA-DCHSBM	A1	50	194	89	13	26
(base case)	A2	100	400	174	13	29
$K = S = 3$	A3	150	604	263	13	29
balanced clusters	A4	200	804	345	13	32
$\mathbb{E}(\rho_s) = 1.7$	A5	500	2015	860	13	30

Continued on next page

Table 3 – continued from previous page

Simulation	Scenario	n	$\lvert\mathcal{E}_2\rvert$	$\lvert\mathcal{E}_3\rvert$	$\bar{d}$	$\max(d)$
$\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.7$	A6	1000	4030	1720	13	31
ScenA-hABCD	A1	50	42	12	2.4	31
(base case)	A2	100	80	24	2.3	32
$K = S = 3$	A3	150	117	37	2.3	32
balanced clusters	A4	200	157	51	2.3	32
$\rho_s = 1.7$	A5	500	394	132	2.3	32
$\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.75$	A6	1000	782	266	2.3	32
ScenB-DCHSBM	B1	50	554	245	37	74
(less sparse)	B2	100	1117	483	37	61
$K = S = 3$	B3	150	1680	720	37	59
balanced clusters	B4	200	2242	958	37	61
$\mathbb{E}(\rho_s) = 1.7$	B5	500	5614	2386	37	63
$\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.7$	B6	1000	11198	4802	37	62
ScenC - HSBM	C1	50	227	100	15	31
(unbalanced clusters)	C2	100	460	208	15	37
$K = S = 3$	C3	150	690	313	15	40
$\pi = (1/6, 1/3, 1/2)$	C4	200	929	423	15.5	35
$\rho_s = 1.7$	C5	500	2319	1063	15.5	39
ScenD-DCHSBM	D1	50	84	199	15	32
$(\lvert\mathcal{E}_3\rvert \gg \lvert\mathcal{E}_2\rvert)$	D2	100	173	402	15	33
$K = S = 3$	D3	150	258	607	16	31
balanced clusters	D4	200	344	805	16	31
$\mathbb{E}(\rho_s) = 1.7$	D5	500	860	2015	16	32
$\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.3$	D6	1000	1722	4028	16	36
ScenE-DCHSBM	E1	50	195	88	13	28
(larger ρ)	E2	100	403	172	13	26
$K = S = 3$	E3	150	605	262	13	27
balanced clusters	E4	200	805	344	13	27
$\mathbb{E}(\rho_s) = 2$	E5	500	2008	867	13	29
$\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.7$	E6	1000	4040	1710	13	31
ScenF-DCHSBM	F1	50	199	84	13	27
(smaller ρ)	F2	100	408	167	13	26
$K = S = 3$	F3	150	605	262	13	30
balanced clusters	F4	200	811	334	13	27
$\mathbb{E}(\rho_s) = 1.4$	F5	500	2004	871	13	31
$\lvert\mathcal{E}_2\rvert/\lvert\mathcal{E}\rvert \simeq 0.7$	F6	1000	4024	1726	13	30
ScenZ-hABCD	Z1	100	49	18	1.5	10
(default h-ABCD)	Z2	150	72	27	1.5	10
K random, $S = 3$	Z3	200	96	37	1.5	10
unbalanced clusters	Z4	500	239	94	1.5	10
linear setting	Z5	1000	478	187	1.5	10

Continued on next page

Table 3 – continued from previous page

Simulation	Scenario	n	$\lvert\mathcal{E}_2\rvert$	$\lvert\mathcal{E}_3\rvert$	$\bar{d}$	$\max(d)$

Table 3: Simulation settings and empirical descriptors of the 25 simulated hypergraphs in each scenario (line): number of clusters (K), maximal hyperedge size (S), within-cluster over between-clusters size-s hyperedges ratio (ρ_s), number of nodes (n), mean number of size-s hyperedges ($|\bar{\mathcal{E}}_s|$), mean node degree ($\bar{d}$) and maximum node degree ($\max(d)$).

We started from a first series of scenarios, called scenarios A, that play the role of a reasonable sparse case for the methods to work. To explore the robustness of our conclusions, these scenarios are presented under the 3 different generating models (HSBM, DCHSBM and h-ABCD) relying on similar settings for sample size n and number of hyperedges $|\mathcal{E}|$. We set the numbers of hyperedges such that they grow linearly with the number of nodes n (sparse setting). We generated $K = 3$ clusters with equal size or probability (depending on the generating model) and the maximum hyperedge size $S = 3$. This latter choice ensures both reasonable computing times and simplicity of model parametrization. The ratio $|\mathcal{E}_2|/|\mathcal{E}| = |\mathcal{E}_2|/(|\mathcal{E}_2| + |\mathcal{E}_3|)$ is set to 0.7 (on average) to reflect the fact that we expect larger sizes hyperedges to be less frequent than smaller-sizes ones. The within-cluster over between-cluster hyperedge ratio is constant wrt size $s \in \{2, 3\}$ and set to $\rho_s = 1.7$ (either exactly or on average), in order to obtain modular hypergraphs.

For this scenario A, we first generated hypergraphs under HSBM with a number of nodes n up to 500, the algorithm becoming too slow for $n = 1000$. Under DCHSBM, we went up to sample size $n = 1000$. Finally we generated samples under h-ABCD again up to a number of nodes $n = 1000$. In this latter model, we considered the strict setting regarding homogeneous hyperedges and choose the parameter ξ such that the resulting $\rho_s = 1.7$ and we set $|\mathcal{E}_2| = 3|\mathcal{E}_3|$, which is approximately the case in the other 2 models. The degree distribution is scale-free with $\gamma = 2.07$ and minimum and maximum value set to 1 and 32, respectively (the observed values in the other 2 models). Note that the range of $\gamma \in (2, 3)$ did not allow us to select mean degrees with similar values than with the other 2 models. In this sense, this scenario A under h-ABCD generating model diverges from the other ones (under HSBM and DCHSBM).

Variant scenarios. We further contrasted scenarios A by varying one characteristic at a time, keeping all others fixed. As our conclusions on scenarios A were globally robust against the choice of the generating model (at least among HSBM and DCHSBM, see next Section 4), we explored those variations in the most convenient model to do so. In scenario B, we decrease the sparsity of the model by generating more hyperedges (keeping all other parameters identical as in scenario A). In scenario C, we explore the effect of unbalanced clusters, while in scenario D, we explore the effect of varying the proportions of size-2 and size-3 hyperedges, namely considering more size-3 than size-2 hyperedges. Scenarios E (resp. F) considers the case where the within-cluster over between-cluster hyperedge ratio ρ_s is increased (resp. decreased) wrt scenario A. Finally, because we obtained pretty bad results for all modularity clustering methods relying on hypergraphs generated by h-ABCD (see next

Section 4), we explored in scenario Z the author's default values of that model to generate modular hypergraphs. Note that in this case, the true number of clusters K is random and the ratio ρ_s cannot be obtained from the model parameters.

3.2 Quality assessment

We now describe the different properties explored to assess the quality of each method. These properties are summarized in Table 4.

We first consider accuracy of the clustering, relying on the Adjusted Rand Index (ARI, Hubert and Arabie, 1985) that measures similarity between $\hat{\mathcal{C}}$ and $\mathcal{C}^{\text{true}}$ (up to label switching). It is upper bounded by 1, where a value of 1 indicates perfect agreement between the clusterings, and negative values indicate less agreement than expected by chance. Then we consider running times (expressed in seconds) of each method. The results have been obtained on a computer with a AMD EPYC 7542 32-Core processor, 128 CPU (2 sockets of 32 double threads cores; we used just one core for each job as none of the procedure is parallelized) and 675Gb RAM. We already mentioned that modularity maximization is far from trivial because of the size of the search space. Thus, an important question is whether the method at stake indeed maximizes its objective. To assess this, we measure the relative error between the ground truth modularity $Q^{\text{true}} = Q(H, \mathcal{C}^{\text{true}})$ and the resulting value $\hat{Q} = Q(H, \hat{\mathcal{C}})$ at the estimated classification $\hat{\mathcal{C}}$, namely

$$\text{error} = \frac{Q(H, \mathcal{C}^{\text{true}}) - Q(H, \hat{\mathcal{C}})}{Q(H, \mathcal{C}^{\text{true}})}.$$

A method that reaches its objective (modularity maximization) without being able to recover the true modular clusters would reveal that it is based on a definition of modularity that is not appropriate. Also note that this error has a sign, with negative values indicating that ground truth modularity is not the maximum value. The mean values and standard deviations for ground truth modularity Q^{true} are also reported (Table 5 in Appendix), since values close to zero could induce unstable errors.

We finally also consider the estimated number of clusters $\hat{K}$ wrt its true value K. In general we present a barplot of the estimated values, to be compared to the true and fixed one. Only for scenarios Z where the true value K is random, we plotted the difference $\hat{K} - K$.

Question	Measure
Is the classification correct?	ARI($\hat{\mathcal{C}}$; $\mathcal{C}^{\text{true}}$)
Is the method fast ?	Running times
Is the modularity maximized?	Relative error between Q^{true} and $\hat{Q}$
Is the number of clusters correct ?	distribution of $\hat{K}$ wrt K

Table 4: Quality assessment.

4 Results

General comparison. We first analyze the results under the simplest scenarios (namely scenarios A, which represent our base case) and the HSBM generating model. Results are presented in Figure 2. First, the `CNM-like` algorithm does not recover the ground truth

clusters, with ARI values around 0 (Figure 2, top left). In fact, the algorithm did not improve over its initialization at $\mathcal{C}^{\text{own}} = (\{1\}, \ldots, \{n\})$ and the number of estimated clusters corresponds to the actual number of nodes (bottom right). Its relative error on modularity is constant and corresponds to the relative difference between the modularity of $\mathcal{C}^{\text{true}}$ and that of $\mathcal{C}^{\text{own}}$. It is positive, so that the modularity maximization goal is clearly not achieved here. The other 3 methods successfully recover the true clusters. For those 3 methods, median ARI values are above 0.7 (top left) and the number of estimated clusters varies between 3 and 6 (bottom right). While the `AON-HMLL` globally obtains the best ARI results (top left), it is also the fastest method (top right) and it attains its objective of modularity maximization (relative error around 0, see bottom left). The `LSR` algorithm was proposed to improve over the `IRMM`. While its relative error on modularity (bottom left) seems in general improved over the latter (with smaller values), Table 5 in Appendix shows that the modularity $Q^{\text{w-clique}}$ optimized by `IRMM` is close to 0 for the ground truth clusters, thus giving unstable errors; while Q^{linear} optimized by `LSR` is strictly positive at those ground truth clusters. Most importantly, from the clustering point of view, ARI is not improved (top left) and computing times are much larger (top right). This seems to indicate that the `LSR` places too much emphasis on maximizing modularity at the expense of clustering recovery. As the number of nodes n increases, we observe that ARI values globally have a lower dispersion, but do not seem to overall improve (top left). This might be due to our setting where the within-cluster over between-cluster hyperedge ratio ρ_s is kept constant when n varies.

Let us now compare these results with those obtained on scenarios A generated under DCHSBM and presented in Figure 3. From these simulations, we confirm the previous conclusions: the `AON-HMLL` is globally the best method and the `CNM-like` algorithm has very low performance for clustering recovery (ARI values very small). The other 2 methods successfully recover the clusters but the `LSR` does not improve on the `IRMM` and has a much larger computing time. Computing times are similar in this simulation and the former one; to see this, we choose to remove computing times for the `LSR` method in scenario A6 (Figure 3, top right). Indeed, those values are all above 15,000 seconds and including them would have changed the y-scale in a way preventing from any possible comparison. As a consequence, we conclude that our analysis is robust against the choice of HSBM or DCHSBM generating model.

To finish with these settings from scenarios A, we consider Figure 4 where the results for hypergraphs generated under the h-ABCD benchmark method are provided. Let us recall that while we tried to mimic as much as possible the characteristics of the scenarios A obtained under HSBM and DCHSBM, it was impossible to obtain similar node degrees within that h-ABCD generating process (see Table 3) and the ones obtained here are much smaller. We observe that in this setting, none of the proposed methods is able to reconstruct the true clusters: ARI values are generally lower than 0.3 (see Figure 4, top left) and the number of estimated clusters is too large (bottom right). Nonetheless, the modularity maximization seems to work as the relative error between the ground truth modularity and its estimation is small (bottom left). Note also that the `LSR` algorithm seems to find a clustering with larger value of the Q^{linear} modularity than at the ground truth clusters (negative errors). Overall, our conclusions raise the following question: are these datasets indeed modular? We will come back to this later when discussing scenarios Z.

We now explore additional insights on the methods performances provided by other scenarios.

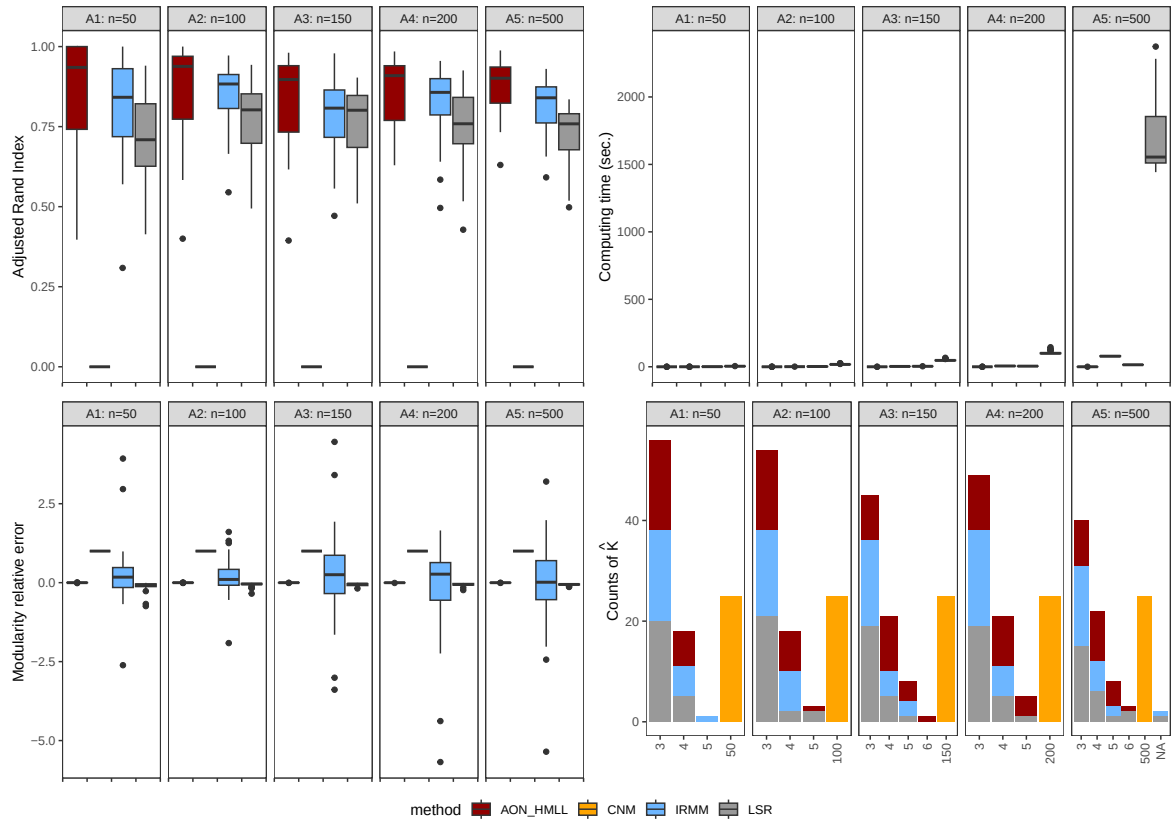

Figure 2: Datasets HSBM, scenarios A1 to A5. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). The `IRMM` and (consequently) the `LSR` methods both gave an error on one dataset in scenario A5. Outlier points have been removed: from the relative error plot (bottom left), 1 value below -500 concerning the `IRMM` method in scenario A1. Moreover, one dataset from scenario A5 gave an error with the `IRMM` and (consequently) the `LSR` methods; corresponding results were removed from the plots.

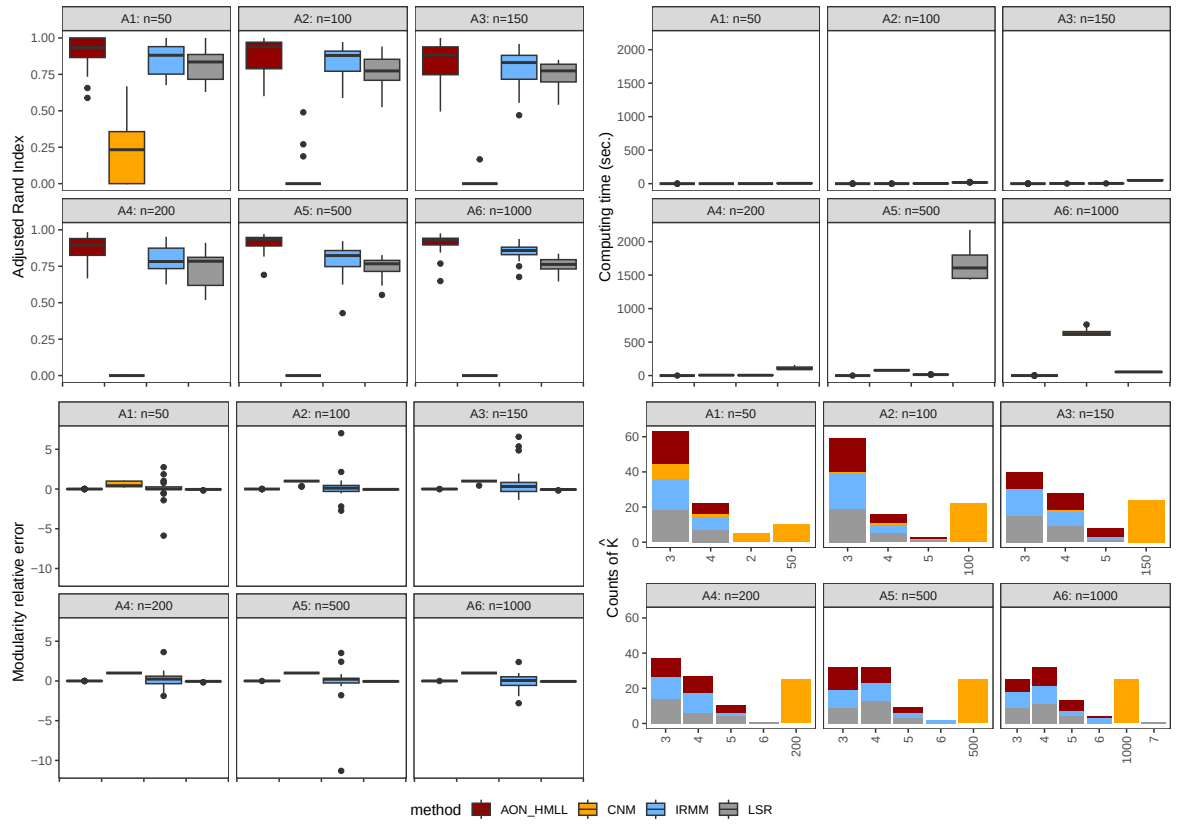

Figure 3: Datasets DCHSBM, scenarios A1 to A6. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). From the time plot (top right), values for the `LSR` method in scenario A6 range between 15,796 and 22,350 seconds and are not shown. Outlier points have been removed from the relative error plot (bottom left): 1 value above 300 concerning the `IRMM` method in scenario A4.

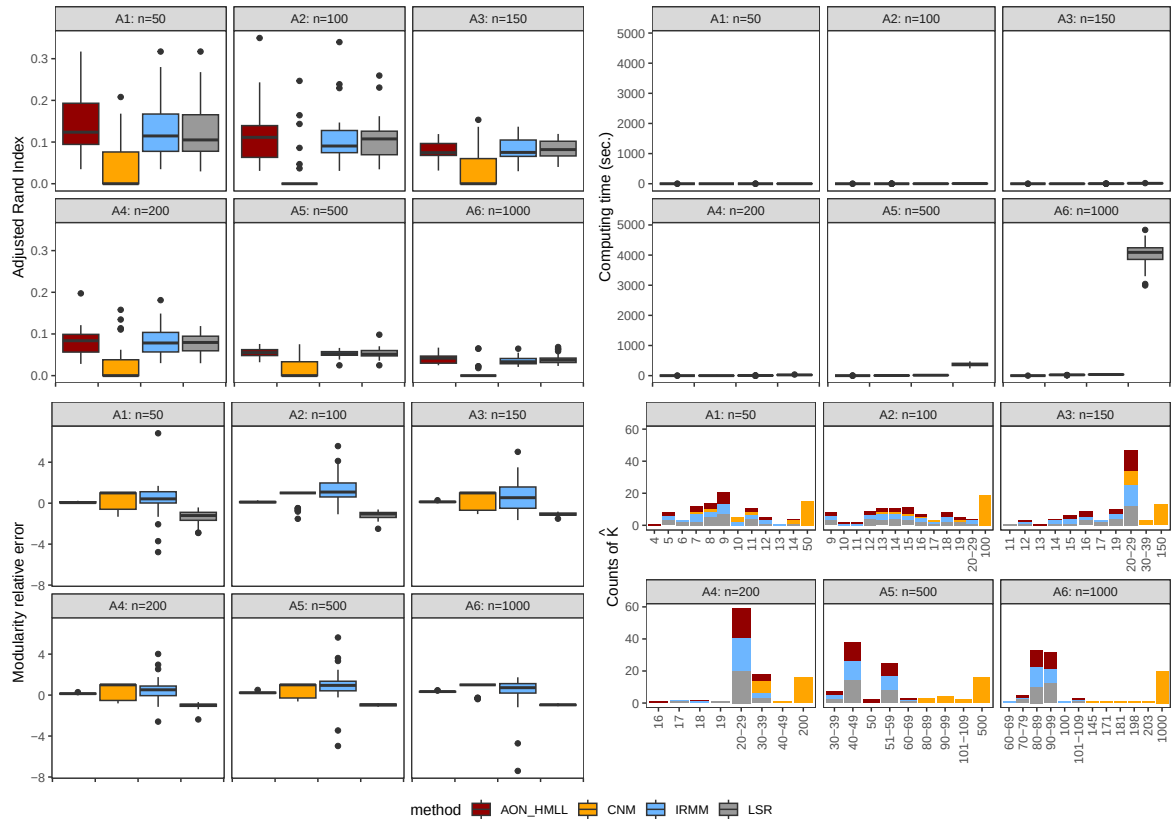

Figure 4: Datasets h-ABCD, scenarios A1 to A6. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). Outlier points have been removed: from the relative error plot (bottom left), 3 values at 25, -50 and -55 concerning the `IRMM` method with in scenarios A6, A2 and A3 respectively.

Impact of sparsity. In scenario B, we decreased the sparsity wrt to scenario A (note that the hypergraphs remain nonetheless sparse, see Table 3). Results are presented in Figure 5. Here again, we removed from the time plot (top right) all values for the `LSR` method in scenario B6. Their range between 16,961 seconds and 17,895 seconds would have changed the y-scale. We mostly observe that while the above conclusions are still valid, the performances of the 3 "working" methods (`AON-HMLL, IRMM` and `LSR`) increase wrt to scenario A. Indeed, except for the `CNM-like` algorithm, the methods exactly recover the true number of clusters (bottom right) and ARI values are almost equal to 1 (top left). Relative errors on modularity are also almost zero for those 3 methods, indicating that the local maximization of the modularity works. We note that the `CNM-like` method has relative error equal to 1. This comes from the fact that the maximized modularity is zero while the ground truth modularity is not zero. Also note that the computing time for this method in scenario B6 becomes significantly larger.

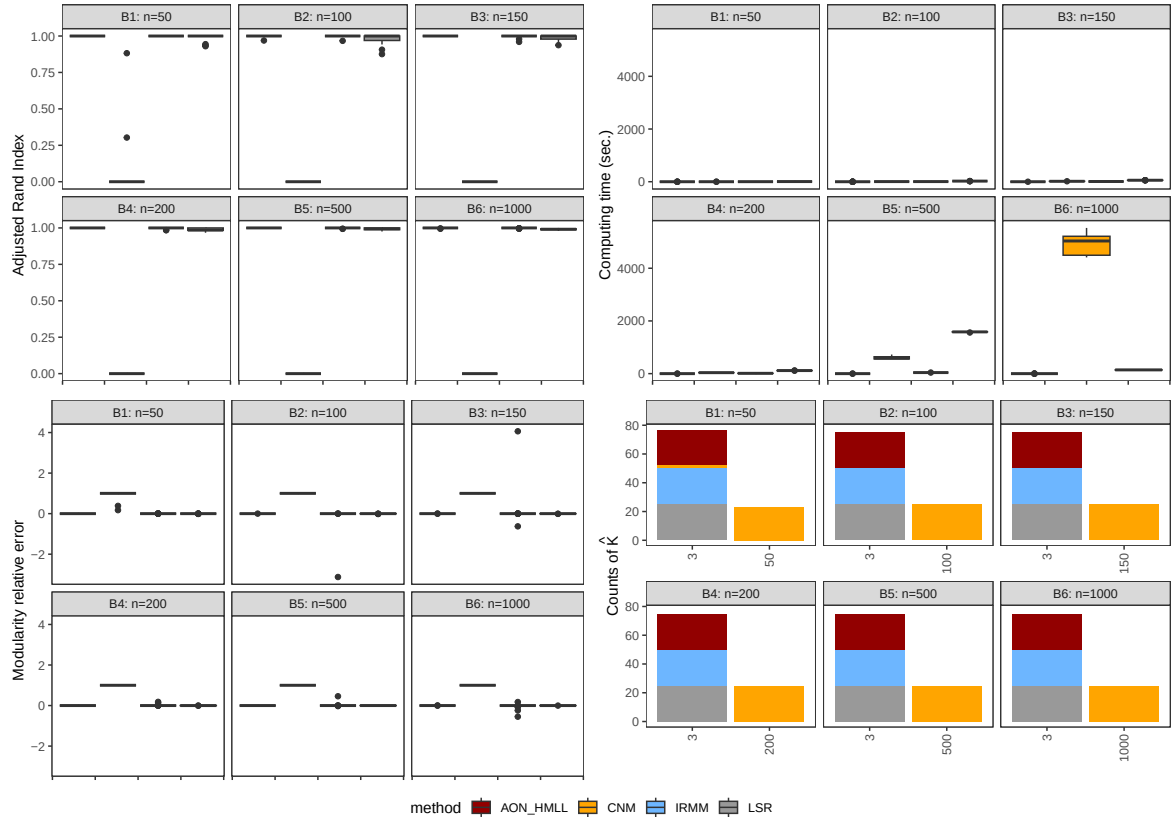

Figure 5: Datasets DCHSBM, scenarios B1 to B6. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). From the time plot (top right), values for the `LSR` method in scenario B6 range between 16,961 and 17,895 seconds are not shown.

Impact of unbalanced clusters. Let us now turn to scenario C where we explore the impact of unbalanced clusters. Results are presented in Figure 6, where we removed from

the time plot (top right) all values for the `LSR` method in scenario C5 as they range between 2,646 and 3,842 seconds. We observe that the overall performances of the methods have decreased wrt scenario A: ARI values are quite low (top left) and the number of clusters is over-estimated (bottom right). Contrarily to scenario A, increasing the number of nodes n degrades the performance of ARI. This is quite counter intuitive, as we expect that with larger values of n, the clusters sizes increase and thus should be easier to detect.

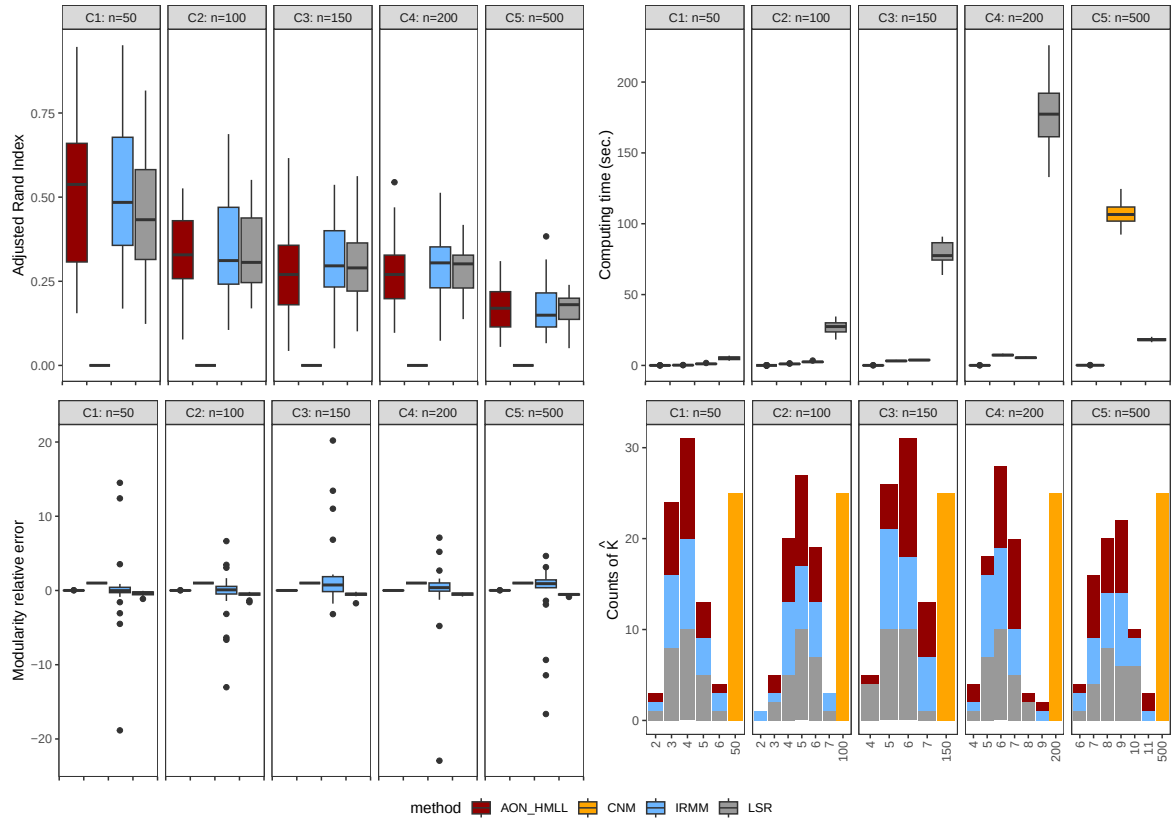

Figure 6: Datasets HSBM, scenarios C1 to C5. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). From the time plot (top right), values for the `LSR` method in scenario C5 range between 2,646 and 3,842 seconds and are not shown. Outlier points have been removed: from the relative error plot (bottom left), 2 values concerning the `IRMM` method, one above 30 in scenario C3 and the second below -60 in scenario C4.

Impact of proportions of size-s hyperedges. In scenario D, we explore the impact of the proportions of size-s hyperedges. More precisely, while scenario A relied on a realistic setting of a smaller number of size-3 than size-2 hyperedges (namely, $|\mathcal{E}_2| \gg |\mathcal{E}_3|$), we explore here the converse setting where $|\mathcal{E}_3| \gg |\mathcal{E}_2|$. Results are presented in Figure 7, where again, we removed from the time plot (top right), all values for the `LSR` method in scenario D6 for comparison purposes, as their range is between 17,153 and 22,377 seconds. Here, we observe as in scenario B that the performances of the 3 methods `AON-HMLL`, `IRMM` and `LSR` increase wrt to scenario A. Indeed, the `AON-HMLL` and the `IRMM` have ARI values equal or close to 1

(top left) and find (almost always) the correct number of clusters (bottom right), indicating (almost) perfect clustering recovery. Relative errors on modularity are also almost zero for those 3 methods, indicating that the local maximization of the modularity works. From that simulation we conclude that clustering via modularity maximization is easier for datasets with a larger proportion of large-size hyperedges and conversely, more difficult in the realistic setting where larger sizes hyperedges are in smaller proportion.

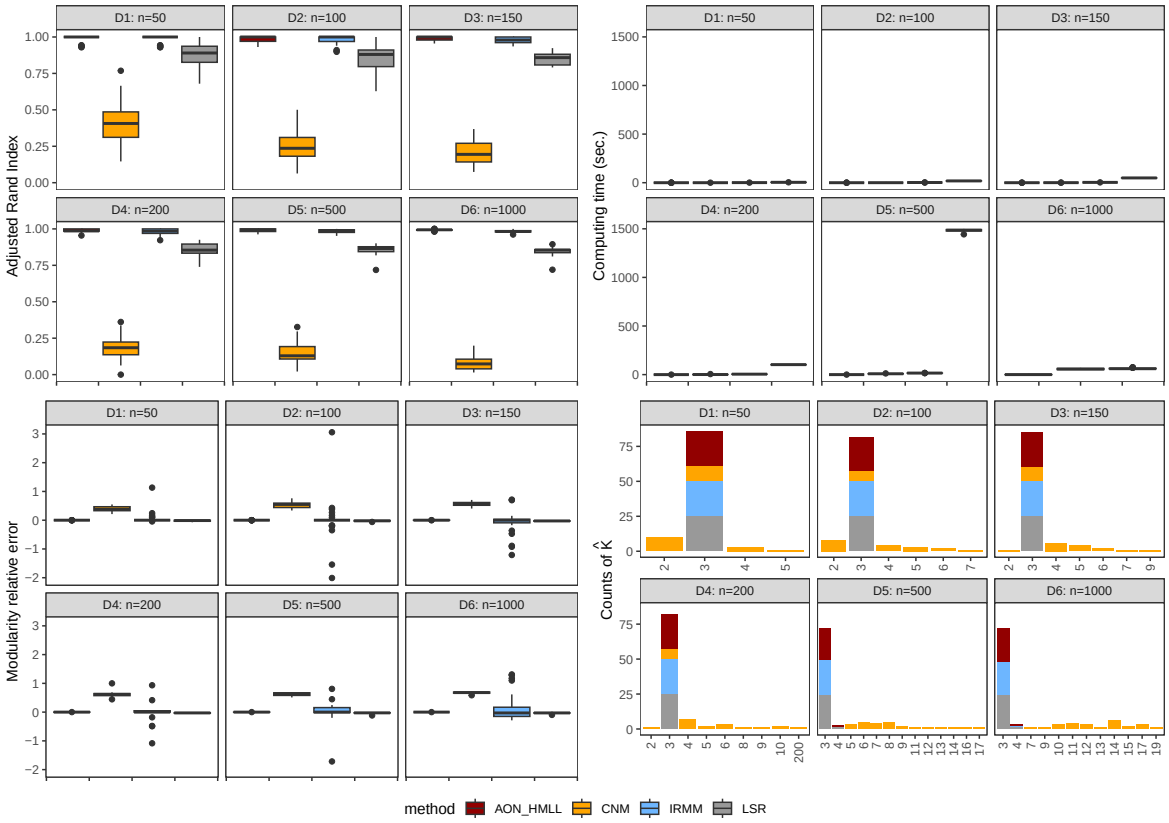

Figure 7: Datasets DCHSBM, scenarios D1 to D6. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). From the time plot (top right), values for the `LSR` method in scenario D6 range between 17,153 and 22,377 seconds and are not shown. Outlier points have been removed: from the time plot (top right), 1 value above 2,800 seconds concerning the `LSR` method in scenario D5 and from the relative error plot (bottom left): 1 value larger than 14 and 1 smaller than -11 concerning the `IRMM` method in scenario D5.

Impact of within-cluster over between-cluster hyperedges ratio. Scenario E (resp. F) rely on a larger (resp. smaller) value for the within-cluster over between-cluster hyperedge ratio ρ_s (still constant with hyperedge size s) compared to scenario A. The results of this simulation are presented in Figure 8 (resp. Figure 9). In those figures again, time values for the `LSR` method in scenarios E6 and F6 respectively have been removed. We can observe that the modularity based methods are sensitive to this parameter ρ_s, with better clustering

results obtained when this ratio is large. As expected, the more modular the hypergraphs are, the easier it is to recover the clusters.

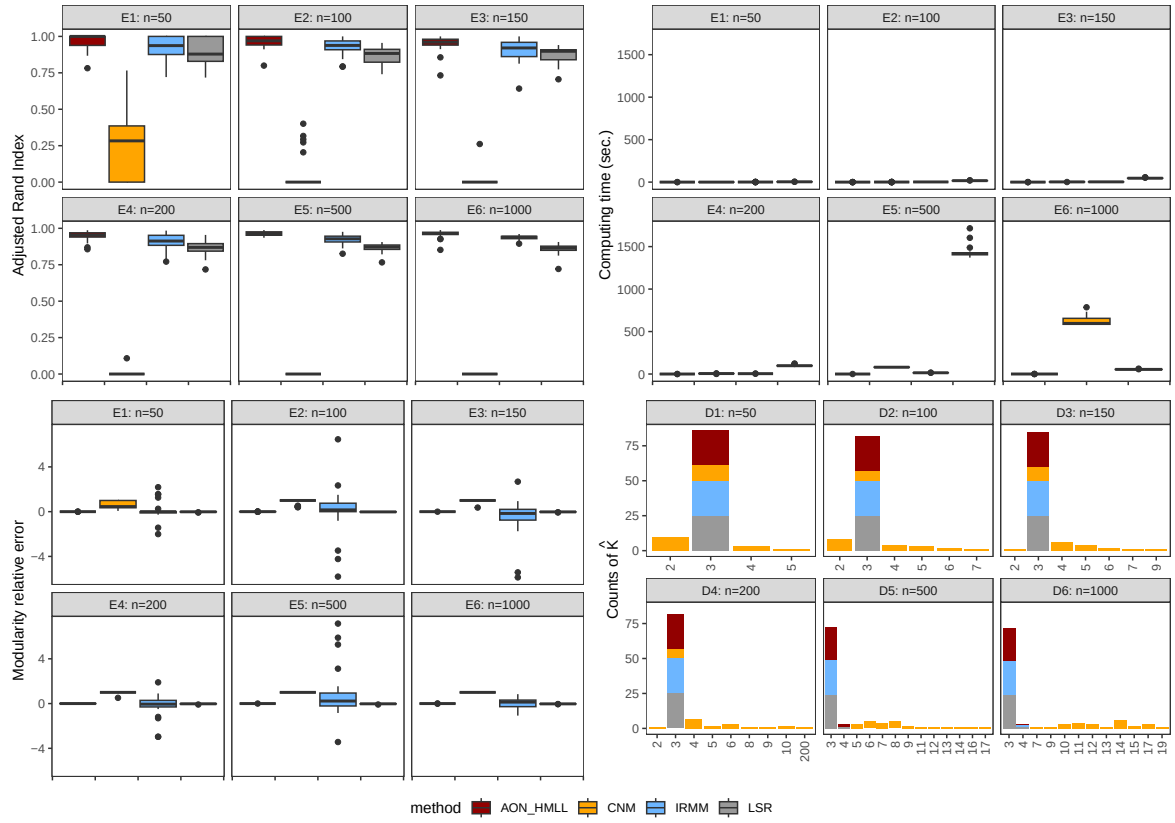

Figure 8: Datasets DCHSBM, scenarios E1 to E6. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). From the time plot (top right), values for LSR method in scenario E6 range between 15,833 and 20604 seconds and are not shown. Outlier points have been removed from the relative error plot (bottom left): 2 values concerning the `IRMM` method, one larger than 100 in scenario E3 and the other smaller than -21 in scenario E4.

Exploring possible bias from generating models. The bad results obtained by all methods on the datasets generated from scenarios A under h-ABCD model raised the question whether those hypergraphs are indeed modular. As we choose the settings of this simulation to mimic the observations obtained under HSBM and DCHSBM but did not completely succeed in that task, one could wonder whether our parameter choices make sense for this model. That is why we consider scenarios Z under h-ABCD, relying on the authors of the model default parameter choices. Note that we started at sample size $n = 100$ because $n = 50$ did not work. The results obtained on these datasets are presented in Figure 10. Here, we observe that again, none of the methods is able to recover the ground truth clusters (top left plot shows ARI values around 0 and bottom right plot shows difference between estimated

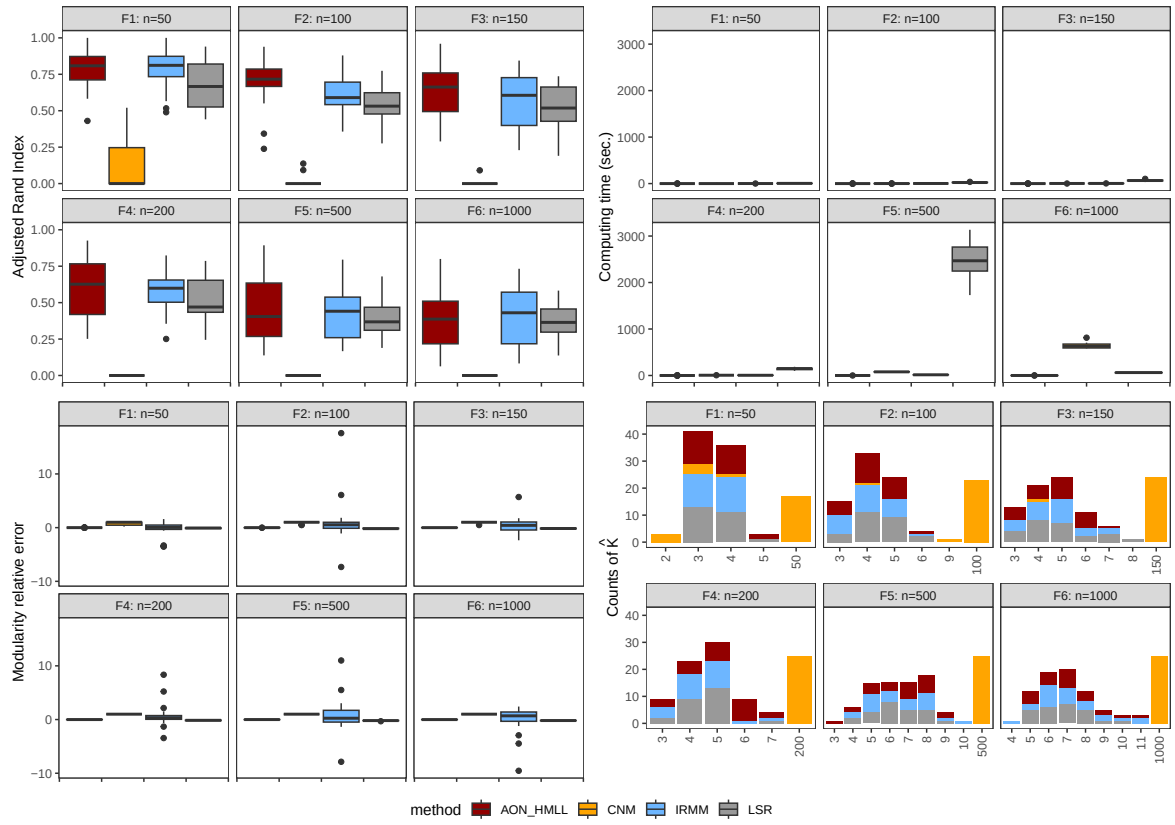

Figure 9: Datasets DCHSBM, scenarios F1 to F6. Comparison by increasing the number of nodes from $n \in \{50, 100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and estimated number of clusters (bottom right, true value is 3). From the time plot (top right), values for LSR method in scenario F6 range between 22,891 and 39,967 seconds and are not shown. Outlier points have been removed from the relative error plot (bottom left): 3 values concerning the `IRMM` method, with 2 values smaller than -40 and -680 and 1 value larger than 22 in scenarios F1, F4 and F6 respectively.

and true number of clusters quite large). This seems to indicate that h-ABCD is not an appropriate benchmark method to test community detection algorithms.

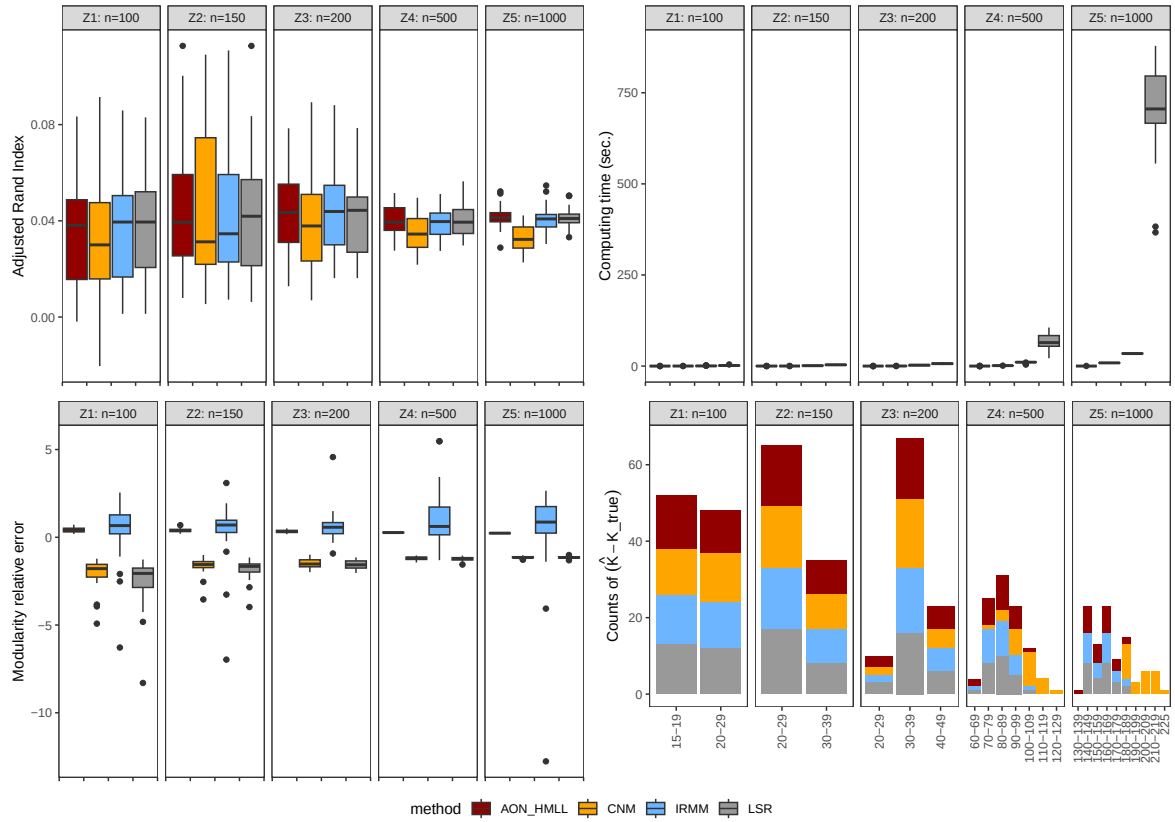

Figure 10: Datasets h-ABCD, scenarios Z1 to Z5. Comparison by increasing the number of nodes from $n \in \{100, 150, 200, 500, 1000\}$: Adjusted Rand Index (top left), time in seconds (top right), relative error on modularity (bottom left) and difference between estimated number of clusters and true value (bottom right). Outlier points have been removed: from the time plot (top right), 4 values above 900 seconds concerning the `LSR` method in scenario Z5 and from the relative error plot (bottom left), 1 value below -28 concerning the `IRMM` method in scenario Z4.

Overall, we could wonder whether the generating models DCHSBM and HSBM could be favoring the `AON-HMLL` method. This could be particularly the case for the DCHSBM model as this model and the `AON-HMLL` method both derived from the same article (Chodrow et al., 2021). However, we can argue against that claim that the modularities Q^{aon} and Q^{strict} maximized by the methods `AON-HMLL` and `CNM-like`, respectively (see summary in Table 1) both focus on contributions by within-clusters hyperedges only. More precisely, the difference in Q^{aon} and Q^{strict} lies only on adaptive weights included in the former, the latter appearing as a special choice of those weights. We thus conclude that our simulations that partly focused on the ratio of within-clusters over between-clusters hyperedges is not especially in favor of the `AON-HMLL` method.

As a final note, we notice that in our experiments, the `IRMM` method sometimes shows some very large values for the relative modularity error (points that we called "outliers" and

removed to preserve y-scales in the plots). Looking at Table 5 in the Appendix we observe that the ground truth modularity $Q^{\text{w-clique}}$ is close to zero, explaining this unstable behaviour of the relative error.

5 Discussion

Let us now summarize the main findings of this study:

- Globally, the best modularity-based approach is the `AON-HMLL`, as it often recovers the ground truth clusters and is among the fastest approaches;
- The `IRMM` algorithm has often good results at recovering ground truth clusters, but it is less fast than the `AON-HMLL`;
- Though the `LSR` algorithm is specifically designed to improve on the `IRMM`, it does not improve the clustering problem at stake;
- The `CNM-like` algorithm does not recover the ground truth clusters in any simulation setting;
- We did not observe any algorithm for which the modularity Q would be correctly maximized (relative error in modularity close to zero) while clusters would not be recovered (low ARI values). Nonetheless, the modularity Q^{strict} from Kamiński et al. (2019a) is not fully maximized by the `CNM-like` method, which leaves open the question of whether it is able to capture communities in hypergraphs.

In the following, we concentrate on commenting the results of the "working methods", namely the `AON-HMLL, IRMM` and the `LSR`:

- The working methods tend to have better results when the densities of the hypergraphs increase, though still in a sparse setting (i.e., when the number of hyperedges increases, see scenarios B);
- The methods are sensitive to the balance in the cluster sizes, with better results when clusters are balanced (see scenarios C);
- The methods tend to have better results when we observe a larger proportion of larger-size hyperedges (i.e., when $|\mathcal{E}_3|$ becomes larger than $|\mathcal{E}_2|$, see scenarios D);
- The methods are sensitive to the ratio ρ_s of within-cluster over between-cluster size-s hyperedges, with better results when this ratio is larger (thus the hypergraph is more modular, see scenarios E and F).

Another conclusion from our study is that the h-ABCD benchmark model (Kamiński et al., 2023b) does not seem appropriate to generate modular hypergraphs, or at least that none of the current modularity-based approaches is able to detect the simulated clusters in those hypergraphs.

Our work is a first building block in gaining a better understanding of modularity in hypergraphs, yet it comes with certain limitations that warrant attention in future research. One constraint arises from computational limitations in both the generating models and

modularity maximization methods, restricting our exploration to relatively small graphs (with a number of nodes $n \leq 1,000$). Consequently, we constrained ourselves to a limited number of clusters ($K \leq 3$), as larger values might lead to clusters too small for effective detection. Our focus was on binary hypergraphs, which already encompass a vast array of higher-order interactions. However, weighted hypergraphs are also of significant interest. Additionally, our approach relied on simulated hypergraphs with characteristics dictated by methodological constraints (e.g., the number of nodes, number of clusters) and others chosen to align with what we believe to be realistic (e.g., sparse hypergraphs, $|\mathcal{E}_2| \gg |\mathcal{E}_3|, \dots$). Lee et al. (2021) examined 13 real-world hypergraphs with heterogeneous sparsity (ratios $|\mathcal{E}|/n$ ranging from as small as 0.5 to around 50) and an average hyperedge size s generally less than 3.9, with two exceptions (hypergraphs related to drug chemicals). Despite this attempt, the literature still lacks a large-scale study on the characteristics of real-world hypergraphs that could inform and support simulations.

There remain numerous unresolved questions that extend beyond the scope of the present contribution. Realistic characteristics, influenced by parameter choices in the generating models, are intricately tied to the issue of detectability thresholds. Specifically, under what circumstances is it possible to effectively recover clusters in a hypergraph? While this question has garnered attention for uniform hypergraphs (Angelini et al., 2015; Chien et al., 2019; Stephan and Zhu, 2022; Zhang and Tan, 2023), real-world hypergraphs, which are non-uniform, remain largely unexplored in this context. Furthermore, moving beyond clustering recovery, it would be valuable to investigate the discriminative power of modularities. Specifically, understanding how discriminative each proposed modularity measure is could provide insights on their design. Examining the distribution of modularity values across a diverse set of hypergraphs, including non-modular ones, holds significant importance. In a similar vein, whether hypergraph modularities are unimodal or not is an important question. Characterizing the behavior of modularities across the entire spectrum of node clusterings would aid in designing suitable modularity-based methods for community detection in hypergraphs.

6 Availability

All the material for reproducing the simulations can be found online at `https://github.com/veronicapoda/modularity/`.

Acknowledgements

The authors would like to thank Professor Veronica Vinciotti for fruitful discussions and her comments on earlier versions of this work.

References

Angelini, M. C., F. Caltagirone, F. Krzakala, and L. Zdeborová (2015). Spectral detection on sparse hypergraphs. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 66–73.

Battiston, F., G. Cencetti, I. Iacopini, V. Latora, M. Lucas, A. Patania, J.-G. Young, and

G. Petri (2020). Networks beyond pairwise interactions: Structure and dynamics. *Phys Rep 874*, 1–92.

Bick, C., E. Gross, H. A. Harrington, and M. T. Schaub (2023). What are higher-order networks? *SIAM Review 65*(3), 686–731.

Blondel, V. D., J.-L. Guillaume, R. Lambiotte, and E. Lefebvre (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*(10), P10008.

Brusa, L. and C. Matias (2022a). Model-based clustering in simple hypergraphs through a stochastic blockmodel. Technical report, arXiv:2210.05983.

Brusa, L. and C. Matias (2022b). `HyperSBM`: Stochastic blockmodel for hypergraphs. R package, `https://github.com/LB1304/HyperSBM`.

Cafieri, S., P. Hansen, and L. Liberti (2010). Loops and multiple edges in modularity maximization of networks. *Phys Rev E 81*, 046102.

Chelaru, M. I., S. Eagleman, A. R. Andrei, R. Milton, N. Kharas, and V. Dragoi (2021). High-order correlations explain the collective behavior of cortical populations in executive, but not sensory areas. *Neuron 109*(24), 3954–3961.

Chien, I. E., C.-Y. Lin, and I.-H. Wang (2019). On the minimax misclassification ratio of hypergraph community detection. *IEEE Transactions on Information Theory 65*(12), 8095–8118.

Chodrow, P., N. Eikmeier, and J. Haddock (2023). Nonbacktracking spectral clustering of nonuniform hypergraphs. *SIAM Journal on Mathematics of Data Science 5*(2), 251–279.

Chodrow, P. S., N. Veldt, and A. R. Benson (2021). Generative hypergraph clustering: From blockmodels to modularity. *Science Advances 7*(28), eabh1303.

Chodrow, P. S., N. Veldt, and A. R. Benson (2022). *HyperModularity*. Julia package, `https://github.com/nveldt/HyperModularity.jl`.

Chung, F. and L. Lu (2002). Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics 6*, 125–145.

Clauset, A., M. E. J. Newman, and C. Moore (2004). Finding community structure in very large networks. *Physical Review E 70*(6), 066111.

Flamm, C., B. M. Stadler, and P. F. Stadler (2015). Chapter 13 - generalized topologies: Hypergraphs, chemical reactions, and biological evolution. In S. C. Basak, G. Restrepo, and J. L. Villaveces (Eds.), *Advances in Mathematical Chemistry and Applications*, pp. 300–328. Bentham Science Publishers.

Ghoshdastidar, D. and A. Dukkipati (2017). Consistency of spectral hypergraph partitioning under planted partition model. *Ann Statist 45*(1), 289 – 315.

Hubert, L. and P. Arabie (1985). Comparing partitions. *J Classif 2*(1), 193–218.

Kamiński, B., P. Prałat, and F. Théberge (2021). Community detection algorithm using hypergraph modularity. In R. M. Benito, C. Cherifi, H. Cherifi, E. Moro, L. M. Rocha, and M. Sales-Pardo (Eds.), *Complex Networks & Their Applications IX*, pp. 152–163.

Kamiński, B., V. Poulin, P. Prałat, P. Szufel, and F. Théberge (2019a). Clustering via hypergraph modularity. *PLoS ONE 14*(11), e0224307.

Kamiński, B., V. Poulin, P. Prałat, P. Szufel, and F. Théberge (2019b). *Clustering via hypergraph modularity - Companion source code and files*. Python code, `https://gist.github.com/pszufe`.

Kamiński, B., P. Prałat, and F. Théberge (2023a). *Artificial Benchmark for Hypergraphs Community Detection (ABCDH) - A Random Hypergraph Model with Community Structure*. Julia code, `https://github.com/bkamins/ABCDHypergraphGenerator.jl`.

Kamiński, B., P. Prałat, and F. Théberge (2023b, 08). Hypergraph Artificial Benchmark for Community Detection (h–ABCD). *Journal of Complex Networks 11*(4), cnad028.

Kumar, T., S. Vaidyanathan, H. Ananthapadmanabhan, S. Parthasarathy, and B. Ravindran (2020). Hypergraph clustering by iteratively reweighted modularity maximization. *Appl. Netw. Sci. 5*(1), 52.

Lancichinetti, A., S. Fortunato, and F. Radicchi (2008). Benchmark graphs for testing community detection algorithms. *Phys. Rev. E 78*, 046110.

Lee, G., M. Choe, and K. Shin (2021). How do hyperedges overlap in real-world hypergraphs? - patterns, measures, and generators. In *Proceedings of the Web Conference 2021*, WWW '21, New York, NY, USA, pp. 3396–3407. Association for Computing Machinery.

Massen, C. P. and J. P. K. Doye (2005). Identifying communities within energy landscapes. *Phys Rev E 71*, 046101.

Muyinda, N., B. De Baets, and S. Rao (2020). Non-king elimination, intransitive triad interactions, and species coexistence in ecological competition networks. *Theor Ecol 13*, 385–397.

Newman, M. E. J. (2016). Equivalence between modularity optimization and maximum likelihood methods for community detection. *Phys. Rev. E 94*, 052315.

Newman, M. E. J. and M. Girvan (2004). Finding and evaluating community structure in networks. *Physical Review E 69*(2), 026113.

Ng, T. and T. Murphy (2022). Model-based clustering for random hypergraphs. *Adv Data Anal Classif 16*, 691–723.

PNNL, L. (2023). *HyperNetX*. Python library (v2.0.3), `https://pnnl.github.io/HyperNetX/index.html`.

Roy, S. and B. Ravindran (2015). Measuring network centrality using hypergraphs. In *Proceedings of the Second ACM IKDD Conference on Data Sciences*, CoDS '15, pp. 59–68.

Ruggeri, N., M. Contisciani, F. Battiston, and C. D. Bacco (2023). Community detection in large hypergraphs. *Science Advances 9*(28), eadg9159.

Simmel, G. (1950). *The sociology of Georg Simmel*. The free press, New York.

Squartini, T. and D. Garlaschelli (2011). Analytical maximum-likelihood method to detect patterns in real networks. *New J Phys 13*(8), 083001.

Stephan, L. and Y. Zhu (2022). Sparse random hypergraphs: Non-backtracking spectra and community detection. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 567–575.

Torres, L., A. S. Blevins, D. Bassett, and T. Eliassi-Rad (2021). The why, how, and when of representations for complex systems. *SIAM Rev 63*(3), 435–485.

Yang, Z., R. Algesheimer, and C. Tessone (2016). A comparative analysis of community detection algorithms on artificial networks. *Sci Rep 6*, 30750.

Zhang, Q. and V. Y. F. Tan (2023). Exact recovery in the general hypergraph stochastic block model. *IEEE Trans Inf Theory 69*(1), 453–471.

A A symmetric hypergraph modularity

Chodrow et al. (2021) define a *symmetric* modularity, where for any partition $\mathcal{C}$ of the set of nodes, the contribution of a hyperedge $e \in \mathcal{E}$ to the modularity of this partition is characterized only by the vector $\mathbf{p}$ whose entries p_k count the number of nodes in e belonging to the k-th largest part in $e \cap \mathcal{C}$. More precisely, fix $\mathcal{C} = (C_1, \dots, C_K)$ a partition of V into K parts and consider $e \subset V$ a subset of nodes (here, e is not necessarily a hyperedge). Nodes in e are partitioned by $\mathcal{C}$ into parts, namely $e \cap \mathcal{C} = (e_1, \dots, e_{J_e})$ for some $J_e \leq K$ (empty parts are discarded). Sorting these parts from largest to smallest, we obtain $e_{(1)}, \dots, e_{(J_e)}$ with $|e_{(1)}| \geq \cdots \geq |e_{(J_e)}| \geq 1$. In other words, $e_{(k)}$ is the k-th largest non-empty part in $e \cap \mathcal{C}$. Then we let $p_k = |e_{(k)}|$ be the sizes of these size-ordered parts. The vector $\mathbf{p} = (p_1, \dots, p_{J_e})$ belongs to the set of partition vectors

$$\mathcal{P} = \{\mathbf{p} = (p_1, \dots p_J); p_1 \geq \cdots \geq p_J \geq 1, \text{ for some } J \geq 1\}.$$

More generally, for any partition $(e_1, \dots, e_J)$ of a subset of nodes $e \subset V$, we consider its ordering $(e_{(1)}, \dots, e_{(J)})$ in size-decreasing order, and let

$$\phi(e_1, \dots, e_J) = (|e_{(1)}|, \dots, |e_{(J)}|) \in \mathcal{P}.$$

Note that $\|\phi(e_1, \dots, e_J)\|_1 = \sum_k |e_{(k)}| = \sum_k |e_k| = |e|$ and we say that the partition vector $\phi(e_1, \dots, e_J)$ has size $|e|$. For any partition $\mathcal{C}$ of the nodes, any subset of nodes $e \subset V$ and any partition vector $\mathbf{p} \in \mathcal{P}$, we say that e is *split* by $\mathcal{C}$ into $\mathbf{p}$ whenever $\phi(e \cap \mathcal{C}) = \mathbf{p}$. In the following, the modularity of a partition $\mathcal{C}$ will focus on how many hyperedges in H are split into different partition vectors $\mathbf{p} \in \mathcal{P}$.

In order to fully understand the links between all the modularities, we introduce a set of nodes subsets as follows. For any partition $\mathcal{C} = (C_1, \dots, C_K)$ of the set of nodes V and any partition vector $\mathbf{p} \in \mathcal{P}$, with $\|\mathbf{p}\|_0 = J \leq K$ and $\|\mathbf{p}\|_1 = s \leq n$, we let

$$C_{\mathbf{p}} = \{e = \{v_1, \dots, v_s\} \subset V; \phi(e \cap \mathcal{C}) = \mathbf{p}\},$$

the set of s-tuples of nodes that are split into the partition vector $\mathbf{p}$ by the partition $\mathcal{C}$. With a slight abuse of notation, we can extend the definitions $e_H(\cdot)$ and $\mathrm{Vol}_H(\cdot)$ to this set of nodes subsets (while those functions were originally defined on a unique subset of nodes). Thus we let

$$e_H(C_{\mathbf{p}}) = \sum_{e \in \mathcal{E}} w(e) 1\{e \in C_{\mathbf{p}}\} = \sum_{e \in \mathcal{E}} w(e) 1\{\phi(e \cap \mathcal{C}) = \mathbf{p}\},$$

and

$$\begin{aligned} \mathrm{Vol}_H(C_{\mathbf{p}}) &:= \sum_{e = \{v_1, \dots, v_s\} \subset V} 1\{\phi(e \cap \mathcal{C}) = \mathbf{p}\} \times \prod_{i=1}^{s} \mathrm{Vol}_H(\{v_i\} \cap \mathcal{C}) \\ &= \sum_{\substack{l_1, \dots, l_J \in \{1, \dots, K\} \\ l_j \text{distinct}}} \prod_{j=1}^{J} \mathrm{Vol}_H(C_{l_j})^{p_j}. \end{aligned} \tag{6}$$

Note that while the quantity $e_H(C_{\mathbf{p}})$ is obtained as a direct generalization of the function e_H defined on a unique subset of nodes, the quantity $\mathrm{Vol}_H(C_{\mathbf{p}})$ is a bit more involved. It is a sum-product of volumes over all distinct clusters labels $(l_1, \dots, l_J)$ that induce the same

partition $\mathbf{p}$. These quantities are the generalizations of the former $\mathrm{Vol}_H(C_k)^s$ that appeared when considering only size-s hyperedges with all nodes belonging to a unique cluster C_k.

Let us consider an example to better understand $\mathrm{Vol}_H(C_{\mathbf{p}})$. We fix a partition $\mathcal{C}$ of the nodes with $K \geq 3$ parts. Then, the only partition vectors of size $s = 2$ are $\mathbf{p} = (2)$ and $\mathbf{p} = (1, 1)$. Similarly, the only partition vectors of size $s = 3$ are $\mathbf{p} = (3); (2, 1); (1, 1, 1)$. Now the volumes $\mathrm{Vol}_H(C_{\mathbf{p}})$ are:

$$\mathrm{Vol}_H(C_{(2)}) = \sum_{k=1}^{K} \mathrm{Vol}_H(C_k)^2; \qquad \mathrm{Vol}_H(C_{(1,1)}) = \sum_{k \neq l \in \{1,\dots,K\}} \mathrm{Vol}_H(C_k)\mathrm{Vol}_H(C_l);$$

$$\mathrm{Vol}_H(C_{(3)}) = \sum_{k=1}^{K} \mathrm{Vol}_H(C_k)^3; \quad \mathrm{Vol}_H(C_{(2,1)}) = \sum_{k \neq l \in \{1,\dots,K\}} \mathrm{Vol}_H(C_k)^2\mathrm{Vol}_H(C_l);$$

$$\mathrm{Vol}_H(C_{(1,1,1)}) = \sum_{\substack{k,l,m \in \{1,\dots,K\} \\ k,l,m \text{ distinct}}} \mathrm{Vol}_H(C_k)\mathrm{Vol}_H(C_l)\mathrm{Vol}_H(C_m).$$

Note that for any partition $\mathcal{C}$, any size s and any integer $c \in \{\lfloor s/2 \rfloor + 1, \dots, s\}$, we have the relation

$$\sum_{k=1}^{K} e_H^{s,c}(C_k) = \sum_{\mathbf{p} \in \mathcal{P}; \|\mathbf{p}\|_1 = s, p_1 = c} e_H(C_{\mathbf{p}}).$$

In other words, the quantity $e_H^{s,c}(C_k)$ counts the number of hyperedges of size s having the majority of their nodes (c of them) in part C_k under partition $\mathcal{C}$. When summing this over all possible k, we get all possible splits of size-s hyperedges into partition vectors $\mathbf{p}$ such that the majority part has exactly c elements ($p_1 = c$).

Chodrow et al. (2021) also introduce a general affinity function $\Omega : \mathcal{P} \to \mathbb{R}$ that modulates the weight of the contribution of each partition vector $\mathbf{p}$. They define the symmetric modularity as:

$$\begin{aligned} Q^{\mathrm{sym}}(H, \mathcal{C}) &= \sum_{\mathbf{p} \in \mathcal{P}} \Big(\sum_{e \in \mathcal{E}} w(e) 1\{\phi(e \cap \mathcal{C}) = \mathbf{p}\} \log(\Omega(\mathbf{p})) - \mathrm{Vol}_H(C_{\mathbf{p}})\Omega(\mathbf{p}) \Big) \\ &= \sum_{\mathbf{p} \in \mathcal{P}} \Big(e_H(C_{\mathbf{p}}) \log(\Omega(\mathbf{p})) - \mathrm{Vol}_H(C_{\mathbf{p}})\Omega(\mathbf{p}) \Big). \end{aligned}$$

A first term in this modularity counts how many (weighted) hyperedges are split by $\mathcal{C}$ into the different partition vectors $\mathbf{p} \in \mathcal{P}$, while a second term is related to the generalized volume $\mathrm{Vol}_H(C_{\mathbf{p}})$. The extra weights $\log(\Omega(\mathbf{p}))$ and $\Omega(\mathbf{p})$ might not seem natural at first. In fact, they appear as the result of an approximate maximum likelihood approach in a specific degree-corrected hypergraph stochastic blockmodel (DCSHBM), in the same way as Newman (2016) did in a graph context. As a result, these extra weights $\log(\Omega(\mathbf{p}))$ and $\Omega(\mathbf{p})$ modulate the influence of the 2 terms whose differences are summed. Also, (up to the scaling factors $\log(\Omega(\mathbf{p}))$ and $\Omega(\mathbf{p})$), while the first term $e_H(C_{\mathbf{p}})$ in each sum still corresponds to an observed number of specific hyperedges, the second term $\mathrm{Vol}_H(C_{\mathbf{p}})$ is not explicitly its expected value under some probabilistic null model. Nonetheless it is a correction term naturally arising from the consideration of a specific probabilistic model (the DCHSBM).

In practice, the affinity function Ω is estimated from an initial partition $\mathcal{C}^{\text{init}}$ through the (weighted) number of hyperedges split into each partition vector by $\mathcal{C}^{\text{init}}$, normalized by the volume of this partition vector under $\mathcal{C}^{\text{init}}$:

$$\forall \mathbf{p} \in \mathcal{P}, \quad \widehat{\Omega}(\mathbf{p}) = \frac{\sum_{e \in \mathcal{E}} w(e) 1\{\phi(e \cap \mathcal{C}^{\text{init}}) = \mathbf{p}\}}{\text{Vol}_H(C^{\text{init}}_{\mathbf{p}})} = \frac{e_H(C^{\text{init}}_{\mathbf{p}})}{\text{Vol}_H(C^{\text{init}}_{\mathbf{p}})},$$

and the modularity becomes

$$\widehat{Q}^{\text{sym}}(H, \mathcal{C}) = \sum_{\mathbf{p} \in \mathcal{P}} \Big(e_H(C_{\mathbf{p}}) \log(\widehat{\Omega}(\mathbf{p})) - \text{Vol}_H(C_{\mathbf{p}}) \widehat{\Omega}(\mathbf{p}) \Big). \tag{7}$$

The modularity $\widehat{Q}^{\text{sym}}$ represents a compromise between the 2 extremes $Q^{\text{w-clique}}$ and Q^{strict}, that goes beyond the one proposed by Q^{wsc}: all hyperedges play a role in this modularity, with weights depending on which partition vector $\mathbf{p}$ they are split in by partition $\mathcal{C}$. This is at the cost of estimating many extra affinity parameter $\Omega(\mathbf{p})$.

Currently, the code provided by Chodrow et al. (2022) does not contain an implementation of an estimation of a general affinity function $\widehat{\Omega}$. Indeed, such a general affinity function requires as many parameters as the total number of possible partitions of a size-s tuple of nodes for any $s \in \{2, \dots, S\}$ with $S = \max_{e \in \mathcal{E}} |e|$, which is huge. Note that their `demos` directory only contains in the file `parameterized-affinities.ipynb` an *experimental* version of a very specific parametrized affinity function.

B All-or-nothing modularity revisited

With the previous notation at stake, we may give a different presentation of Q^{aon}. The all-or-nothing affinity function Ω is defined by

$$\forall \mathbf{p} \in \mathcal{P} \text{ such that } \|\mathbf{p}\|_1 = s, \quad \Omega^{\text{aon}}(\mathbf{p}) = \begin{cases} \omega_{s1} & \text{if } \|\mathbf{p}\|_0 = 1, \\ \omega_{s0} & \text{else.} \end{cases} \tag{8}$$

Here, the ℓ_1 norm of an integer vector $\mathbf{p} = (p_1, \dots, p_J)$ is the sum of its entries $\|\mathbf{p}\|_1 = \sum_j p_j$ and its ℓ_0 norm is the number of its entries $\|\mathbf{p}\|_0 = J$. In other words, a partition vector $\mathbf{p}$ has AON affinity

$$\Omega^{\text{aon}}(\mathbf{p}) = \sum_{s \geq 2} \Big(\omega_{s1} 1\{\mathbf{p} = (s)\} + \omega_{s0} 1\{\|\mathbf{p}\|_0 \geq 2; \|\mathbf{p}\|_1 = s\} \Big).$$

For each size s of a partition vector, this affinity function is parametrized by only two different values ω_{s1}, ω_{s0}. These parameters will modulate differently the contributions of hyperedges e depending on their size $s = |e|$ and on whether all their nodes belong to the same cluster (i.e., $\|\phi(e \cap \mathcal{C})\|_0 = 1$) or belong to at least 2 different clusters (i.e., $\|\phi(e \cap \mathcal{C})\|_0 \geq 2$). In particular, the volumes (6) computed for partition vectors $\mathbf{p}$ such that $\|\mathbf{p}\|_0 = 1$ write

$$\text{Vol}_H(C_{(s)}) = \sum_{k=1}^{K} \text{Vol}_H(C_k)^s.$$

The AON affinity function will in practice be estimated from an initial partition $\mathcal{C}^{\text{init}}$ through

$$\forall s \geq 2, \quad \widehat{\omega}_{s1} = \frac{e_H(C^{\text{init}}_{(s)})}{\sum_{k=1}^K \text{Vol}_H(C^{\text{init}}_k)^s}; \quad \widehat{\omega}_{s0} = \frac{\sum_{e \in \mathcal{E}_s} w(e) 1\{\|\phi(e \cap \mathcal{C}^{\text{init}})\|_0 \geq 2\}}{\sum_{\mathbf{p} \in \mathcal{P}; \|\mathbf{p}\|_0 \geq 2, \|\mathbf{p}\|_1 = s} \text{Vol}_H(C^{\text{init}}_{\mathbf{p}})}.$$

Finally, inserting that affinity function inside Q^{sym} and after some algebra, the resulting modularity becomes (up to additional constants that do not depend on $\mathcal{C}$ and are thus neglected)

$$\begin{aligned} Q^{\text{aon}}(H, \mathcal{C}) &= -\sum_{s \geq 2} \hat{\beta}_s \Big(\text{cut}_s(\mathcal{C}) + \hat{\gamma}_s \sum_{k=1}^K (\text{Vol}_H(C_k))^s \Big) + c \\ &= \sum_{s \geq 2} \hat{\beta}_s \Big(e_H(C_{(d)}) - \hat{\gamma}_s \sum_{k=1}^K (\text{Vol}_H(C_k))^s \Big) + c \\ &= \sum_{k=1}^K \sum_{s \geq 2} \hat{\beta}_s \Big(\sum_{C'_k \subset C_k; |C'_k| = s} e_H(C'_k) - \hat{\gamma}_s (\text{Vol}_H(C_k))^s \Big) + c, \end{aligned} \tag{9}$$

where $\hat{\beta}_s = \log \hat{\omega}_{s1} - \log \hat{\omega}_{s0}$ and $\hat{\gamma}_s = \hat{\beta}_s^{-1}(\hat{\omega}_{s1} - \hat{\omega}_{s0})$, the cut terms $\text{cut}_s(\mathcal{C})$ count the (weighted) number of hyperedges of size s that contain nodes in two or more distinct clusters; namely, $\text{cut}_s(\mathcal{C}) = |\mathcal{E}_s| - e_H(C_{(s)})$. The first line in Equation (9) is the expression of Q^{aon} given in Chodrow et al. (2021) while the following lines correspond to our rewriting, showing the similarities with the other previously defined modularities. While in general we may expect that $\hat{\omega}_{s1} > \hat{\omega}_{s0}$ so that both $\hat{\beta}_s, \hat{\gamma}_s > 0$, we then recover in this expression a sum of difference terms between a count of specific hyperedges (those entirely included in a cluster) and a correcting volume term.

C Linking parameters in HSBM and DCHSBM-like

If we consider a DCHSBM-like model generating multiset hypergraphs with equal cluster proportions, then we get using twice Baye's rule,

$$\begin{aligned} p_s &= \frac{\mathbb{P}(e = \{v_1, \ldots, v_s\} \in \mathcal{E}_s; \exists 1 \leq k \leq K, \phi(e \cap \mathcal{C}^{\text{true}}) \subset C^{\text{true}}_k)}{|\mathcal{E}_s|/n^s} \\ &= \frac{\alpha_s K (K/n)^s}{|\mathcal{E}_s|/n^s} = \frac{\alpha_s K^{s+1}}{|\mathcal{E}_s|}. \end{aligned}$$

On the other way round, if we consider a HSBM generating simple hypergraphs with cluster proportions π_k, then we get using twice Baye's rule,

$$\alpha_s = \frac{p_s[\alpha_s \sum_k \pi_k^s + \beta_s(1 - \sum_k \pi_k^s)]}{\sum_k \pi_k^s},$$

and in the particular case of equal cluster proportions, namely $\pi_k = 1/K$, then this leads to

$$\alpha_s = \frac{p_s \beta_s (K^{s-1} - 1)}{1 - p_s}.$$

D Values for ground truth modularity

Table 5 shows mean and standard deviations values of the modularities at the ground truth clustering for each method (each one focusing on a specific modularity definition, as summarized in Table 1) and each scenario. We observe that the `IRMM` algorithm that maximizes $Q^{\text{w-clique}}$ has a ground truth modularity close to 0.

scenario / method	AON-HMLL	CNM	IRMM	LSR
ScenA-HSBM				
A1: $n = 50$	-1440.28(253.46)	0.31(0.04)	-0.01(0.02)	0.23(0.04)
A2: $n = 100$	-3281.04(321.45)	0.34(0.02)	-0.01(0.02)	0.27(0.02)
A3: $n = 150$	-5089.41(349.39)	0.35(0.02)	0(0.01)	0.27(0.02)
A4: $n = 200$	-7007.19(431.96)	0.36(0.02)	0(0.01)	0.28(0.02)
A5: $n = 500$	-19151.93(657.66)	0.36(0.01)	0(0.01)	0.28(0.01)
ScenA-DCHSBM				
A1: $n = 50$	-1507.05(140.8)	0.38(0.03)	-0.02(0.02)	0.3(0.03)
A2: $n = 100$	-3214.43(259.46)	0.37(0.02)	0(0.02)	0.29(0.02)
A3: $n = 150$	-5055.25(307.23)	0.37(0.02)	-0.01(0.01)	0.29(0.02)
A4: $n = 200$	-6819.27(297.63)	0.37(0.02)	0(0.01)	0.29(0.02)
A5: $n = 500$	-18619.63(568.68)	0.36(0.01)	0(0.01)	0.29(0.01)
A6: $n = 1000$	-39925.19(782.67)	0.36(0.01)	0(0.01)	0.29(0.01)
ScenA-hABCD				
A1: $n = 50$	-177.69(39.81)	0.27(0.1)	-0.02(0.07)	0.24(0.09)
A2: $n = 100$	-439.39(58.81)	0.34(0.07)	-0.01(0.04)	0.3(0.06)
A3: $n = 150$	-727.05(63.2)	0.36(0.04)	-0.01(0.03)	0.31(0.03)
A4: $n = 200$	-1040.2(69.62)	0.38(0.04)	-0.01(0.03)	0.33(0.04)
A5: $n = 500$	-3453.7(312.77)	0.42(0.01)	0(0.02)	0.36(0.01)
A6: $n = 1000$	-8578.86(633.97)	0.44(0.01)	0(0.01)	0.37(0.01)
ScenB-DCHSBM				
B1: $n = 50$	-4476.24(286.51)	0.38(0.02)	-0.01(0.01)	0.3(0.02)
B2: $n = 100$	-9895.18(438.98)	0.37(0.01)	-0.01(0.01)	0.29(0.01)
B3: $n = 150$	-15436.71(407.27)	0.37(0.01)	0(0.01)	0.29(0.01)
B4: $n = 200$	-21146.49(497.66)	0.36(0.01)	0(0.01)	0.29(0.01)
B5: $n = 500$	-57396.92(1005.76)	0.37(0)	0(0)	0.29(0)
B6: $n = 1000$	-122253.16(1071.73)	0.36(0)	0(0)	0.29(0)
ScenC-HSBM				
C1: $n = 50$	-1692.96(341.87)	0.22(0.04)	-0.01(0.02)	0.16(0.04)
C2: $n = 100$	-3818.35(526.14)	0.23(0.03)	-0.01(0.01)	0.16(0.03)
C3: $n = 150$	-6006.44(720.38)	0.24(0.03)	0(0.01)	0.17(0.03)
C4: $n = 200$	-8411.32(678.46)	0.24(0.02)	0(0.01)	0.17(0.02)
C5: $n = 500$	-23301.14(1133.75)	0.24(0.02)	0(0.01)	0.17(0.02)
ScenD-DCHSBM				
D1: $n = 50$	-3268.27(172.98)	0.47(0.03)	-0.01(0.02)	0.28(0.02)
D2: $n = 100$	-7377.67(213.42)	0.45(0.02)	0(0.01)	0.27(0.01)
D3: $n = 150$	-11692.36(325.51)	0.45(0.02)	-0.01(0.01)	0.27(0.01)

Continued on next page

Table 5 – continued from previous page

scenario / method	AON-HMLL	CNM	IRMM	LSR
D4: $n = 200$	-15946(274.87)	0.45(0.01)	0(0.01)	0.27(0.01)
D5: $n = 500$	-43675.15(431.82)	0.45(0.01)	0(0.01)	0.27(0.01)
D6: $n = 1000$	-92887.89(1079.8)	0.45(0.01)	0(0)	0.27(0.01)
ScenE-DCHSBM				
E1: $n = 50$	-1496.2(102.41)	0.41(0.03)	-0.01(0.02)	0.33(0.02)
E2: $n = 100$	-3210.91(190.09)	0.41(0.02)	0(0.01)	0.33(0.02)
E3: $n = 150$	-5117.84(249)	0.41(0.02)	-0.01(0.01)	0.32(0.02)
E4: $n = 200$	-6902.04(311.24)	0.4(0.02)	-0.01(0.01)	0.32(0.01)
E5: $n = 500$	-19069.69(632.72)	0.4(0.01)	0(0.01)	0.32(0.01)
E6: $n = 1000$	-40020.25(699.77)	0.4(0)	0(0)	0.32(0)
ScenF-DCHSBM				
F1: $n = 50$	-1401.74(122.37)	0.33(0.03)	-0.02(0.02)	0.26(0.02)
F2: $n = 100$	-3045.16(240.8)	0.32(0.02)	0(0.02)	0.25(0.02)
F3: $n = 150$	-4990.03(275.21)	0.32(0.02)	0(0.01)	0.25(0.01)
F4: $n = 200$	-6623.92(208.33)	0.33(0.02)	0(0.01)	0.26(0.02)
F5: $n = 500$	-18654.39(359.67)	0.32(0.01)	0(0.01)	0.25(0.01)
F6: $n = 1000$	-39341.82(941.03)	0.32(0.01)	0(0)	0.25(0.01)
ScenZ-hABCD				
Z1: $n = 100$	-529.92(164.11)	0.28(0.07)	-0.02(0.04)	0.26(0.08)
Z2: $n = 150$	-782.78(209.52)	0.33(0.05)	-0.02(0.04)	0.31(0.05)
Z3: $n = 200$	-961.22(128.09)	0.35(0.04)	-0.01(0.02)	0.34(0.03)
Z4: $n = 500$	-2457.49(132.12)	0.39(0.02)	0(0.01)	0.41(0.02)
Z5: $n = 1000$	-5027.19(177.84)	0.41(0.01)	0(0.01)	0.43(0.01)

Table 5: Mean value (and standard deviation) of ground truth modularities $Q^{\text{true}} = Q(H, \mathcal{C}^{\text{true}})$ for the 25 hypergraphs generated under each method and scenario.