Community Detection Algorithm Using Hypergraph Modularity

Bogumił Kamiński[1], Paweł Prałat[2(✉)], and François Théberge[3]

[1] Warsaw School of Economics, Warsaw, Poland
bkamins@sgh.waw.pl
[2] Ryerson University, Toronto, ON, Canada
pralat@ryerson.ca
[3] The Tutte Institute for Mathematics and Computing, Ottawa, ON, Canada
theberge@ieee.org

Abstract. We propose a community detection algorithm for hypergraphs. The main feature of this algorithm is that it can be adjusted to various scenarios depending on how often vertices in one community share hyperedges with vertices from other community.

Keywords: Community detection · Hypergraphs · Modularity

1 Motivation and Our Contribution

An important property of complex networks is their community structure, that is, the organization of vertices in clusters, with many edges joining vertices of the same cluster and comparatively few edges joining vertices of different clusters. In social networks, communities may represent groups by interest (practical application include collaborative tagging), in citation networks they correspond to related papers, and in the web communities are formed by pages on related topics. Being able to identify communities in a network could help us to exploit this network more effectively. Clusters in citation graphs may help to find similar scientific papers, discovering social network users with similar interests is important for targeted advertisement, etc.

Many networks that are currently modelled as graphs would be more accurately modelled as hypergraphs. This includes the collaboration network in which nodes correspond to researchers and hyperedges correspond to papers that consist of nodes associated with researchers that co-authorship a given paper. Unfortunately, the theory and tools are not sufficiently developed to allow most problems, including clustering, to be tackled directly within this context. Indeed, researchers and practitioners often create the 2-section graph of a hypergraph of interest (that is, replace each hyperedge with a clique). After moving to the 2-section graph, one clearly loses some information about hyperedges of size greater than two and so there is a common believe that one can do better by using the knowledge of the original hypergraph.

R. M. Benito et al. (Eds.): COMPLEX NETWORKS 2020, SCI 943, pp. 152–163, 2021.
https://doi.org/10.1007/978-3-030-65347-7_13

There are some recent attempts to deal with hypergraphs in the context of clustering. For example, Kumar et al. [6,7] still reduce the problem to graphs but use the original hypergraphs to iteratively adjust weights to encourage some hyperedges to be included in some cluster but discourage other ones (this process can be viewed as separating signal from noise). Moreover, in [4] a number of extensions of the classic null model for graphs are proposed that can potentially be used by true hypergraph algorithms. Unfortunately, there are many ways such extensions can be done depending on how often vertices in one community share hyperedges with vertices from other communities. This is something that varies between networks at hand and usually depends on the hyperedge sizes. Indeed, hyperedges associated with papers written by mathematicians might be more homogeneous and smaller in comparison with those written by medical doctors who tend to work in large and multidisciplinary teams. Moreover, in general, papers with a large number of co-authors tend to be less homogeneous. A good algorithm should be able to automatically decide which extension should be used.

In this paper, we propose a framework that is able to adjust to various scenarios mentioned above. We do it by generalizing and unifying all extensions of the graph modularity function to hypergraphs, and putting them into one framework in which the contribution from different "slices" is controlled by hyper-parameters that can be tuned for a given scenario (Sect. 2). We propose two prototype algorithms that show the potential of the framework, the so-called proof-of-concept (Sect. 3). In order to test the performance of algorithms in various scenarios, we introduce a synthetic random hypergraph model (Sect. 4) that may be of independent interest. We experiment with our prototypes as well as the two main competitors in this space, namely, the Louvain and Kumar et al. algorithms (Sect. 5). The experiments show that, after tuning hyper-parameters appropriately, the proposed prototypes work very well. Independently, we provide an evidence that such tuning can be done in an unsupervised way. Of course, more work and experiments need to be done before we are able to announce a scalable and properly tuned algorithm but at the end of this paper we reveal a bit more details to that effect (Sect. 6). *Spoiler alert*: the reader who wants to be surprised should avoid that section.

2 Modularity Functions

We start this section by recalling the classic definition of modularity function for graphs (Sect. 2.1). In order to deal with hypergraphs, one may reduce the problem to graphs by considering the corresponding 2-section graph (Sect. 2.2). Alternatively, one may generalize the modularity function to hypergraphs (Sect. 2.3) and then perform algorithms directly on hypergraphs. Such approach should presumably give better results as it preserves more information on the original network in comparison to the corresponding 2-section graphs. In this paper, we generalize the hypergraph modularity function even further that allows us to value various contributions to the modularity function differently (Sect. 2.4).

2.1 Modularity Function for Graphs

Before we define the modularity function, let us introduce some necessary notation and terminology. Let $G = (V, E)$ be a graph where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E is the set of edges. The edges are multisets of V of cardinality 2 (that is, with loops allowed). Throughout the paper, we will use $n = |V|$ for the number of vertices of G. For a given vertex $v \in V$, $\deg_G(v)$ is the *degree* of v in G (with a loop at v contributing 2 to the degree of v). For $A \subseteq V$, let the *volume* of A be $\mathrm{vol}_G(A) = \sum_{v \in A} \deg_G(v)$; in particular, the volume of the graphs is $\mathrm{vol}_G(V) = \sum_{v \in V} \deg_G(v) = 2|E|$.

The definition of modularity for graphs was first introduced by Newman and Girvan in [11]. Despite some known issues with this function such as the "resolution limit" reported in [3], many popular algorithms for partitioning vertices of large graphs use it [2,8,10] and perform very well. The modularity function favours partitions of the vertex set of a graph G in which a large proportion of the edges fall entirely within the parts (often called clusters), but benchmarks it against the expected number of edges one would see in those parts in the corresponding Chung-Lu random graph model which generates graphs with the expected degree sequence following exactly the degree sequence in G.

Formally, for a graph $G = (V, E)$ and a given partition $\mathbf{A} = \{A_1, A_2, \ldots, A_k\}$ of V, the *modularity function* is defined as follows:

$$q_G(\mathbf{A}) = \sum_{A_i \in \mathbf{A}} \frac{e_G(A_i)}{|E|} - \sum_{A_i \in \mathbf{A}} \left(\frac{\mathrm{vol}_G(A_i)}{\mathrm{vol}_G(V)} \right)^2, \tag{1}$$

where $e_G(A_i) = |\{\{v_j, v_k\} \in E : v_j, v_k \in A_i\}|$ is the number of edges in the subgraph of G *induced by* set A_i. The first term in (1), $\sum_{A_i \in \mathbf{A}} e_G(A_i)/|E|$, is called the *edge contribution* and it computes the fraction of edges that fall within one of the parts. The second one, $\sum_{A_i \in \mathbf{A}} (\mathrm{vol}_G(A_i)/\mathrm{vol}_G(V))^2$, is called the *degree tax* and it computes the expected fraction of edges that do the same in the corresponding random graph (the null model). The modularity measures the deviation between the two.

It is easy to see that $q_G(\mathbf{A}) \leq 1$. Also, if $\mathbf{A} = \{V\}$, then $q_G(\mathbf{A}) = 0$, and if $\mathbf{A} = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$, then $q_G(\mathbf{A}) = -\sum(\deg_G(v)/\mathrm{vol}_G(V))^2 < 0$. The maximum *modularity* $q^*(G)$ is defined as the maximum of $q_G(\mathbf{A})$ over all possible partitions $\mathbf{A}$ of V; that is, $q^*(G) = \max_{\mathbf{A}} q_G(\mathbf{A})$. In order to maximize $q_G(\mathbf{A})$ one wants to find a partition with large edge contribution subject to small degree tax. If $q^*(G)$ approaches 1 (which is the trivial upper bound), we observe a strong community structure; conversely, if $q^*(G)$ is close to zero (which is the trivial lower bound), there is no community structure. The definition in (1) can be generalized to weighted edges by replacing edge counts with sums of edge weights.

2.2 Using Graph Modularity for Hypergraphs

Given a hypergraph $H = (V, E)$, it is common to transform its hyperedges into complete graphs (cliques), the process know as forming the 2-section of

H, graph $H_{[2]}$ on the same vertex set as H. For each hyperedge $e \in E$ with $|e| \geq 2$ and weight $w(e)$, $\binom{|e|}{2}$ edges are formed, each of them with weight of $w(e)/(|e|-1)$. While there are other natural choices for the weights (such as the original weighting scheme $w(e)/\binom{|e|}{2}$ that preserves the total weight), this choice ensures that the degree distribution of the created graph matches the one of the original hypergraph H [6,7]. Moreover, let us also mention that it also nicely translates a natural random walk on H into a random walk on the corresponding $H_{[2]}$ [13]. As hyperedges in H usually overlap, this process creates a multigraph. In order for $H_{[2]}$ to be a simple graph, if the same pair of vertices appear in multiple hyperedges, the edge weights are simply added together.

2.3 Modularity Function for Hypergraphs

For the hypergraph $H = (V, E)$, each hyperedge $e \in E$ is a multiset of V of any cardinality $d \in \mathrm{N}$. Multisets in the context of hypergraphs are natural generalization of loops in the context of graphs. Even though H does not always contain multisets, it is convenient to allow them as they may appear in the random hypergraph that will be used to "benchmark" the edge contribution component of the modularity function. It will be convenient to partition the edge set E into $\{E_1, E_2, \ldots\}$, where E_d consists of hyperedges of size d. As a result, hypergraph H can be expressed as the disjoint union of *d-uniform hypergraphs* $H = \bigcup H_d$, where $H_d = (V, E_d)$. As for graphs, $\deg_H(v)$ is the degree of vertex v, that is, the number of hyperedges v is a part of (taking into account the fact that hyperedges are multisets). Finally, the volume of a vertex subset $A \subseteq V$ is $\mathrm{vol}_H(A) = \sum_{v \in A} \deg_H(v)$.

For edges of size greater than 2, several definitions can be used to quantify the edge contribution for a given partition $\mathbf{A}$ of the vertex set. As a result, the choice of hypergraph modularity function is not unique. It depends on how strongly one believes that a hyperedge is an indicator that some of its vertices fall into one community. The fraction of vertices of a given hyperedge that belong to one community is called its *homogeneity* (provided it is more than 50%). In one extreme case, all vertices of a hyperedge have to belong to one of the parts in order to contribute to the modularity function; this is the *strict* variant assuming that only homogeneous hyperedges provide information about underlying community structure. In the other natural extreme variant, the *majority* one, one assumes that edges are not necessarily homogeneous and so a hyperedge contributes to one of the parts if more than 50% of its vertices belong to it; in this case being over 50% is the only information that is considered relevant for community detection. All variants in between guarantee that hyperedges contribute to at most one part. Alternatively, a hyperedge could contribute to the part that corresponds to the largest fraction of vertices. However, this might not uniquely determine the part and it is more natural to classify such edges as "noise" that should not contribute to any part anyway. Once the variant is fixed, one needs to benchmark the corresponding edge contribution using the degree tax computed for the generalization of the Chung-Lu model to hypergraphs proposed in [4].

The framework introduced in [4] is more flexible but, for simplicity, let us concentrate only on the two extreme cases. For $d \in \mathrm{N}$ and $p \in [0,1]$, let $\mathrm{Bin}(d,p)$ denotes the binomial random variable with parameters d and p. The *majority-based modularity* function for hypergraphs is defined as

$$q_H^m(\mathbf{A}) = \sum_{A_i \in \mathbf{A}} \frac{e_H^m(A_i)}{|E|} - \sum_{d \geq 2} \frac{|E_d|}{|E|} \sum_{A_i \in \mathbf{A}} \mathrm{P}\left(\mathrm{Bin}\left(d, \frac{\mathrm{vol}_H(A_i)}{\mathrm{vol}_H(V)}\right) > \frac{d}{2}\right), \quad (2)$$

and the *strict-based modularity* as

$$\begin{aligned} q_H^s(\mathbf{A}) &= \sum_{A_i \in \mathbf{A}} \frac{e_H^s(A_i)}{|E|} - \sum_{d \geq 2} \frac{|E_d|}{|E|} \sum_{A_i \in \mathbf{A}} \left(\frac{\mathrm{vol}_H(A_i)}{\mathrm{vol}_H(V)}\right)^d \\ &= \sum_{A_i \in \mathbf{A}} \frac{e_H^s(A_i)}{|E|} - \sum_{d \geq 2} \frac{|E_d|}{|E|} \sum_{A_i \in \mathbf{A}} \mathrm{P}\left(\mathrm{Bin}\left(d, \frac{\mathrm{vol}_H(A_i)}{\mathrm{vol}_H(V)}\right) = d\right). \quad (3) \end{aligned}$$

In (2), $e_H^m(A_i)$ counts the number of hyperedges where the majority of vertices belong to part A_i while in (3), $e_H^s(A_i)$ counts the number of edges where all vertices are in part A_i. The goal is the same as for graphs. We search for a partition $\mathbf{A}$ that yields modularity as close as possible to the maximum *modularity* $q^*(H)$ which is defined as the maximum over all possible partitions of the vertex set. We can define weighted versions of the above functions (with weights on hyperedges) the same way as we did for graphs. Finally, note that if H consists only of hyperedges of size 2 (that is, H is a graph), then both (2) and (3) reduce to (1).

2.4 Unification and Generalization

In this section, we unify the definitions of modularity functions and put them into one common framework. This general framework is more flexible and can be tuned and applied to hypergraphs with hyperedges of different homogeneity.

In order to achieve our goal, we "dissect" the modularity function so that each "slice" can be considered independently. For each hyperedge size d, we will independently deal with contribution to the modularity function coming from hyperedges of size d with precisely c members from one of the parts, where $c > d/2$. For example, for $d = 7$ we get 4 slices corresponding to various values of c, namely, $c \in \{4,5,6,7\}$.

Let us first note that (2) can be rewritten as follows:

$$q_H^m(\mathbf{A}) = \sum_{A_i \in \mathbf{A}} \sum_{d \geq 2} \sum_{c=\lfloor d/2 \rfloor + 1}^{d} \left(\frac{e_H^{d,c}(A_i)}{|E|} - \frac{|E_d|}{|E|} \cdot \mathrm{P}\left(\mathrm{Bin}\left(d, \frac{\mathrm{vol}_H(A_i)}{\mathrm{vol}_H(V)}\right) = c\right)\right),$$

where $e_H^{d,c}(A_i)$ is the number of hyperedges of size d that have exactly c members in A_i. So $q_H^m(\mathbf{A})$ can be viewed as:

$$q_H^m(\mathbf{A}) = \sum_{d \geq 2} \sum_{c=\lfloor d/2 \rfloor + 1}^{d} q_H^{c,d}(\mathbf{A}),$$

where

$$q_H^{c,d}(\mathbf{A}) = \frac{1}{|E|} \sum_{A_i \in \mathbf{A}} \left(e_H^{d,c}(A_i) - |E_d| \cdot \mathrm{P}\left(\mathrm{Bin}\left(d, \frac{\mathrm{vol}(A_i)}{\mathrm{vol}(V)} \right) = c \right) \right).$$

Similarly, (3) can be viewed as:

$$q_H^s(\mathbf{A}) = \sum_{d \geq 2} q_H^{d,d}(\mathbf{A}).$$

Hence, in the majority-based modularity function q_H^m, each slice is weighted equally whereas for the strict-based definition q_H^s, only the slices with $c = d$ are considered.

In order to unify the definitions, our new modularity function is controlled by *hyper-parameters* $w_{c,d} \in [0,1]$ ($d \geq 2$, $\lfloor d/2 \rfloor + 1 \leq c \leq d$). For a fixed set of hyper-parameters, we simply define

$$q_H(\mathbf{A}) = \sum_{d \geq 2} \sum_{c = \lfloor d/2 \rfloor + 1}^{d} w_{c,d}\, q_H^{c,d}(\mathbf{A}). \tag{4}$$

This definition gives us more flexibility and allows as to value some slices more than others. In our experiments, we restricted ourselves to the following family of hyper-parameters that gave us enough flexibility but is controlled only by 3 variables. (In fact, we will argue later on that one of them, namely $\rho_{\max}$, can be set to one.) Let $\alpha \in [0, \infty)$, and $\rho_{\min}, \rho_{\max} \in (0.5, 1]$ such that $\rho_{\min} \leq \rho_{\max}$. Then,

$$w_{c,d} = \begin{cases} (c/d)^\alpha & \text{if } \lceil d\rho_{\min} \rceil \leq c \leq \lceil d\rho_{\max} \rceil. \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Parameters $\rho_{\min}$ and $\rho_{\max}$ are related to the assumption on the minimal and, respectively, maximal "pureness" of hyperedges and depends on the level of homogeneity of the network. In particular, $\rho_{\max}$ may be bounded away from one if one expects that "totally pure" (that is, occurring in a single community) hyperedges are unlikely to be observed in practice. Finally, parameter α governs the smooth transition between the relative informativeness between contributing hyperedges of different levels of "pureness".

As a result, after adjusting the hyper-parameters accordingly, (4) can be used for the two extreme cases (majority-based and strict-based) and anything in between. Moreover, (4) may well approximate the graph modularity for the corresponding 2-section graph $H_{[2]}$. Indeed, if c vertices of a hyperedge e of size d and weight $w(e)$ fall into one part of the partition $\mathbf{A}$, then the contribution to the graph modularity is $w(e)\binom{c}{2}/(|e|-1)$ (in the variant where the degrees are preserved) or $w(e)\binom{c}{2}/\binom{|e|}{2} \approx w(e)(c/|e|)^2$ (if the total weight is preserved). Hence, the hyper-parameters can be adjusted to reflect that. The only difference is that (4) does not allow to include contributions from parts that contain at most $d/2$ vertices which still contributes to the graph modularity of $H_{[2]}$. However, most of the contribution comes from large values of c and so the two corresponding measures are close in practice.

3 Algorithms

In this paper, we experiment with four clustering algorithms that can handle networks represented as hypergraphs. The last two of them are two prototypes of our hybrid and flexible framework under development. More advanced version will be presented in the forthcoming papers but some spoilers are provided in Sect. 6.

3.1 *Louvain*—Graph-Based Algorithm

As discussed in Sect. 2.2, in order to find communities in a hypergraph H, one may reduce the problem to graphs by considering its 2-section (weighted) graph $H_{[2]}$ and then try to find a partition that maximizes the graph modularity function (1) for $H_{[2]}$. One of the mostly used unsupervised algorithms for detecting communities in graphs is the *Louvain* algorithm [1]. It is a hierarchical clustering algorithm that tries to optimize the modularity function (modularity optimization phase), merge communities into single vertices (community aggregation phases), and then it recursively executes the modularity clustering on the condensed graphs until no increase in modularity is possible.

All clustering algorithms are heuristic in nature and only aim to find "good enough" partition without the hope of finding the best one. In particular, in order to be able to search different parts of the solution space, *Louvain* is a randomized algorithm that orders all vertices randomly before the modularity optimization phase takes place. Unfortunately, it means that the algorithm is not stable and outcomes of it may vary significantly between independent runs. In order to solve this issue, the ensemble clustering algorithm for graphs (*ECG*) [12] can be used instead. This algorithm, known to have good stability, is based on the *Louvain* algorithm and the concept of consensus clustering.

3.2 Kumar et al.—Refinement of Graph-Based Algorithm

The following refinement of Kumar et al. [6,7] generally gives better results than the original *Louvain* algorithm on several synthetic and real-world examples. However, this algorithm is not truly hypergraph-based but should rather be viewed as a refinement of a graph-based approach guided by the original hypergraph. In this algorithm, one first builds a degree-preserving weighted graph G based on the original hypergraph H. Then, the *Louvain* algorithm is applied to G that tries to maximize the graph modularity function (1). After that, hypergraph H is revisited and hyperedges are carefully re-weighted based on their measure of homogeneity between the obtained parts. These steps are repeated until convergence.

3.3 *LS* and *HA*—Our Prototypes

All successful algorithms based on graph modularity optimization (including Louvain, ECG, and Kumar et al. mentioned above) start the same way. Vertices are initially put into their own clusters, and a basic move is to consider

changing the cluster of a vertex to one of its neighbours' if it increases the modularity function. Unfortunately, trying to apply this strategy to hypergraphs is challenging. Indeed, if one starts from all vertices in their own community, then changing the cluster of only one vertex will likely have no positive effect on the modularity function unless edges of small size are present. For example, it takes several moves for a hyperedge of size $d \geq 4$ to have the majority of its vertices to fall into the same community.

In order to solve this problem, we propose to use the graph modularity function $q_G(\mathbf{A})$ defined in (1) to "lift the process from the ground" but then switch to the hypergraph counterpart $q_H(\mathbf{A})$ defined in (4). There are many ways to achieve it and one of them is mentioned in Sect. 6. For experiments provided in this paper, we consider two prototypes: the first one (*HA*) switches to hypergraphs as soon as possible whereas the second one (*LS*) stays with graphs for much longer. The first algorithm, that we call *HA* (for *hybrid algorithm*), works as follows:

1. Form small, tight clumps by running *ECG* using $q_G(\mathbf{A})$ on the degree-preserving graph G built from H. Prune edges below the threshold value of 70% (number of votes), and keep connected components as initial clumps.
2. Merge clumps (in a random order) if $q_H(\mathbf{A})$ improves. Repeat until no more improvement is possible.
3. Move one vertex at a time (in a random order) to a neighbouring cluster if it improves q_H. Repeat until convergence.

The second algorithm, that we call *LS* (for *last step*) runs Kumar et al. and only does the last step (step 3.) above.

Finally, recall that the hypergraph modularity function $q_H(\mathbf{A})$ is controlled by hyper-parameters $w_{c,d}$ but we restrict ourselves to a family of such parameters guided by parameters $\alpha, \rho_{\min}$, and $\rho_{\max}$; see (5). Hence, we will refer to the above algorithms as $HA(\alpha, \rho_{\min}, \rho_{\max})$ and, respectively, $LS(\alpha, \rho_{\min}, \rho_{\max})$.

4 Synthetic Random Hypergraph Model

In order to test various algorithms in a controlled, synthetic environment, we propose a simple model of a random hypergraph with communities. Such synthetic networks with an engineered *ground truth* are commonly used to evaluate the performance of clustering algorithms. There are many graph models of complex networks, including the well-known and widely used LFR benchmark graph [9] and our own ABCD [5]. On the other hand, very little has been done with hypergraphs. As we aim for a simple model and the degree distribution should not affect our exploratory experiments, we propose a model that is inspired by the classical stochastic block model. Designing more realistic model and performing experiments on it is left for future research.

A random hypergraph $\mathcal{H}$ consists of K *communities*; the kth community has n_k members so the total number of vertices in $\mathcal{H}$ is equal to $n = \sum_{k=1}^{K} n_k$. For $2 \leq d \leq M$, m_d is the number of hyperedges of size d; in particular, M is the

size of a largest hyperedge and $m = \sum_{d \geq 2} m_d$ is the total number of hyperedges. Hyperedges are partitioned into *community* and *noise* hyperedges. The expected proportion of noise edges is $\mu \in [0, 1]$, the parameter that controls the *level of noise*. Each community hyperedge will be assigned to one community. The expected fraction of hyperedges that are assigned to the kth community is p_k; in particular, $\sum_{k=1}^{K} p_k = 1$. Community hyperedges that are assigned to the kth community will have majority members from that community. On the other hand, noise hyperedges will be "sprinkled" across the whole hypergraph.

The hyperedges of $\mathcal{H}$ are generated as follows. For each edge size d, we independently generate m_d edges of size d. For each edge e of size d, we first decide if e is a community hyperedge or a noise. It is a noise with probability μ; otherwise, it is a community hyperedge. If e turns out to be a noise, then we simply choose its d vertices uniformly at random from the set of all sets of vertices of size d, regardless to which community they belong to. On the other hand, if e is a community edge, then we assign it to community k with probability p_k. Then, we fix the homogeneity value τ_e of hyperedge e that is the integer-valued random variable taken uniformly at random from the *homogeneity set* $\{\lceil \tau_{\min} d \rceil, \lceil \tau_{\min} d \rceil + 1, \ldots, \lceil \tau_{\max} d \rceil\}$. The homogeneity set depends on parameters $\tau_{\min}$ and $\tau_{\max}$ of the model that satisfy $0.5 < \tau_{\min} \leq \tau_{\max} \leq 1$, and is assumed to be the same for all edges. Finally, members of e are determined as follows: τ_e vertices are selected uniformly at random from the kth community, and the remaining vertices are selected uniformly at random from vertices outside of this community.

As mentioned above, the proposed model is aimed to be simple but it tries to capture the fact that many real-world networks represented as hypergraphs exhibit various levels of homogeneity or the lack of thereof. Moreover, some networks are noisy with some fraction of hyperedges consisting of vertices from different communities. Such behaviour can be controlled by parameters $\tau_{\min}$, $\tau_{\max}$, and μ. It gives us a tool to test the performance of our algorithms for various scenarios. A good algorithm should be able to adjust to any scenario in an unsupervised way.

5 Experiments

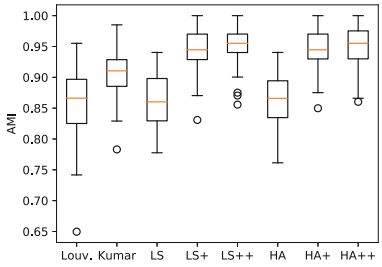
For our experiments we use the synthetic random hypergraph model introduced in Sect. 4. It contains 5 communities, each consisting of 40 vertices: $(n_1, n_2, \ldots, n_5) = (40, 40, \ldots, 40)$. The distribution of hyperedge sizes is as follows: $(m_1, m_2, \ldots, m_{11}) = (30, 30, 30, 30, 30, 30, 30, 20, 20, 20)$. The expected fraction of edges that belong to a given cluster is equal to 0.2: $(p_1, p_2, \ldots, p_5) = (0.2, 0.2, \ldots, 0.2)$. The lower bound for the homogeneity interval is fixed to be $\tau_{\min} = 0.65$. We performed experiments on four hypergraphs with the remaining two parameters fixed to: a) $(\mu, \tau_{\max}) = (0, 0.65)$, b) $(\mu, \tau_{\max}) = (0, 0.8)$, c) $(\mu, \tau_{\max}) = (0, 1)$, d) $(\mu, \tau_{\max}) = (0.1, 0.80)$. All of them lead to the same conclusion so we present figures only for hypergraph $\mathcal{H}$ that is obtained with parameters d).

We test the two known algorithms, Louvain and Kumar et al., as well as our two prototypes, LS and HA. For each prototype, we test three different sets of hyper-parameters. In the first variant, we include contribution to the hypergraph modularity function that comes from all slices, that is, we fix $\rho_{\min} = 0.5^{+} = 0.5 + \epsilon$ (for some very small $\epsilon > 0$ so that all "slices" of the modularity function are included) and $\rho_{\max} = 1$. For simplicity, we fix $\alpha = 1$. For convenient notation, let $LS = LS(1, 0.5^{+}, 1)$ and $HA = HA(1, 0.5^{+}, 1)$. For the second variant, we use the knowledge about the hypergraph (*ground truth*) and concentrate only on slices that are above the lower bound for the homogeneity set, that is, we fix $\rho_{\min} = \tau_{\min} = 0.65$ but keep $\rho_{\max} = 1$. Let $LS+ = LS(1, 0.65, 1)$ and $HA+ = HA(1, 0.65, 1)$. Finally, we use the complete knowledge about the generative process of our synthetic hypergraph and fix $\rho_{\min} = \tau_{\min} = 0.65$ and $\rho_{\max} = \tau_{\max} = 0.80$. The corresponding algorithms are denoted by $LS{+}{+} = LS(1, 0.65, 0.85)$ and $HA{+}{+} = HA(1, 0.65, 0, 85)$.

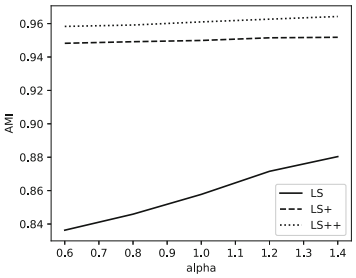
In the first experiment, we run each algorithm on $\mathcal{H}$ and measure its performance using the *Adjusted Mutual Information* (AMI). AMI is the information theory measure that allows us to quantify the similarity between two partitions of the same set of nodes, the partition returned by the algorithm and the *ground truth*. Since all algorithms involved are randomized, we run them 100 times and present a box-plot of the corresponding AMIs in Fig. 1(a). We see that LS and HA give comparable results as the original Louvain and Kumar et al. is consistently better. On the other hand, when our prototypes are provided with a knowledge about the homogeneity of $\mathcal{H}$, they perform very well, better than Kumar et al. There is a less difference between + and ++ variants of the two prototypes. This is a good and desired feature as "pure" hyperedges should not generally be penalized unless there is some known external hard constraint that prevents hyperedges to be homogeneous. On the other hand, if large hyperedges are non-homogeneous, then the quality of + and ++ should be similar as these "slices" barely contribute to the modularity function anyway. Note that this observation does not apply to small hyperedges; in the extreme situation when dealing with graphs with hyperedges of size 2, any choice of $\rho_{\max} \geq \rho_{\min}$ leads to exactly the same results.

The previous experiment shows that knowing some global statistics (namely, how homogeneous the network is) significantly increases the performance of our prototypes. However, typically such information is not available and the algorithm has to learn such global statistics in an unsupervised way. In our second experiment, we test if this is possible. We take a partition returned by Kumar et al. and investigate all hyperedges of $\mathcal{H}$. For each hyperedge e we check if at least $\tau \geq 0.5$ fraction of its vertices belong to some community. We compare it with the corresponding homogeneity value based on the *ground truth*. The two distributions are presented in Fig. 1(c) and are almost indistinguishable. This suggests that learning the right value of $\rho_{\min}$ should be possible in practice.

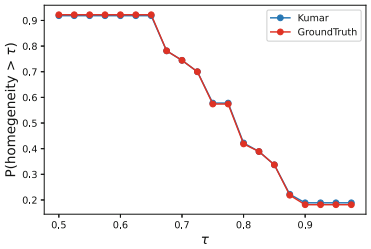
Finally, we tested the performance of our prototypes for various choices of parameter α. + and ++ variants turn out to be not too sensitive whereas LS and HA increase their performance as α increases—Figure 1(b). It is perhaps not



(a) AMI



(b) AMI as a function of α



(c) True/experimental homogeneity

Fig. 1. Experiments on hypergraph $\mathcal{H}$: $\mu = 0.1$, $\tau_{\min} = 0.65$, $\tau_{\max} = 0.8$.

too surprising as increasing α puts more weight to more homogeneous hyperedges which has similar effect to tuning parameter $\rho_{\min}$. More comprehensive experiments are to be performed.

6 Conclusions and Future Directions

In this paper, we propose two prototype algorithms and do some simple experiments that show their potential (of course, we experimented much more and most experiments are encouraging). Clearly more work needs to be done. For example, we showed that our prototypes work very well but only once proper tuning of the hyper-parameters is done. Initial experiments show that such tuning can be done in an unsupervised way but details need to be fixed. One important thing that we keep in mind is a potential risk of a solution to be overfitted.

We proposed two ways to solve a problem with initial phase of any algorithm based on the hypergraph modularity function, our two prototypes. Another option is to embed vertices of the 2-section graph in a geometric space such that nearby vertices are more likely to share an edge than those far from each other. Then, for example, the classical *k-means* algorithm with some large value of k may be used to find the initial partition and then one can switch to the hypergraph based algorithms optimizing the hypergraph modularity function.

Hyperedge re-weighting scheme proposed by Kumar et al. seems to work very well. This is an independent component that can be easily incorporated within our framework. We aim for a flexible framework that can mimic Louvain, ECG,

Kumar et al., and anything in between, but is additionally enhanced by the opportunities provided by the hypergraph modularity function.

The algorithm has to be scalable so that we may run it on large hypergraphs. The updates of the hypergraph modularity function can be done fast but it requires a proper design/usage of dedicated data structures and algorithms. We currently implement such a code in the Julia language.

Finally, on top of experimenting on large synthetic hypergraphs we plan to perform experiments on real-world networks represented as hypergraphs.

References

1. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theor. Exp. **10**, P10008 (2008)
2. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Phys. Rev. E **70**, 066111 (2004)
3. Fortunato, S., Barthelemy, M.: Resolution limit in community detection. Proc. Natl. Acad. Sci. U.S.A. **104**, 36–41 (2007)
4. Kamiński, B., Poulin, V., Prałat, P., Szufel, P., Théberge, F.: Clustering via hypergraph modularity. PLOS ONE **14**(11), e0224307 (2019)
5. Kamiński, B., Prałat, P., Théberge, F.: Artificial Benchmark for Community Detection (ABCD)—Fast Random Graph Model with Community Structure, arXiv:2002.00843
6. Kumar, T., Vaidyanathan, S., Ananthapadmanabhan, H., Parthasarathy, S., Ravindran, B.: A new measure of modularity in hypergraphs: theoretical insights and implications for effective clustering. In: International Conference on Complex Networks and Their Applications, Complex Networks 2019, pp. 286–297. Springer, Cham (2019)
7. Kumar, T., Vaidyanathan, S., Ananthapadmanabhan, H., Parthasarathy, S., Ravindran, B.: Hypergraph clustering by iteratively reweighted modularity maximization. Appl. Netw. Sci **5** (2020)
8. Lancichinetti, A., Fortunato, S.: Limits of modularity maximization in community detection. Phys. Rev. E **84**, 066122 (2011)
9. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E **78** (2008)
10. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**, 066133 (2004)
11. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**, 026–113 (2004)
12. Poulin, V., Théberge, F.: Ensemble clustering for graphs. In: Aiello, L., Cherifi, C., Cherifi, H., Lambiotte, R., Lió, P., Rocha, L. (eds.) Complex Networks and their Applications VII, COMPLEX NETWORKS 2018. Studies in Computational Intelligence, vol. 812. Springer, Cham (2018)
13. Théberge, F.: Summer School on Data Science Tools and Techniques in Modelling Complex Networks. https://github.com/ftheberge/ComplexNetworks2019/