


RESEARCH

Open Access



Hypergraph clustering by iteratively reweighted modularity maximization

Tarun Kumar^{1,2*} , Sankaran Vaidyanathan^{1,2,3}, Harini Ananthapadmanabhan^{2,4}, Srinivasan Parthasarathy⁵ and Balaraman Ravindran^{1,2}

*Correspondence:

tkumar@cse.iitm.ac.in

¹Robert Bosch Centre for Data Science and AI, Chennai, India

²Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India
Full list of author information is available at the end of the article

Abstract

Learning on graphs is a subject of great interest due to the abundance of relational data from real-world systems. Many of these systems involve higher-order interactions (super-dyadic) rather than mere pairwise (dyadic) relationships; examples of these are co-authorship, co-citation, and metabolic reaction networks. Such super-dyadic relations are more adequately modeled using hypergraphs rather than graphs. Learning on hypergraphs has thus been garnering increased attention with potential applications in network analysis, VLSI design, and computer vision, among others. Especially, hypergraph clustering is gaining attention because of its enormous applications such as component placement in VLSI, group discovery in bibliographic systems, image segmentation in CV, etc. For the problem of clustering on graphs, modularity maximization has been known to work well in the pairwise setting. Our primary contribution in this article is to provide a generalization of the modularity maximization framework for clustering on hypergraphs. In doing so, we introduce a null model for graphs generated by hypergraph reduction and prove its equivalence to the configuration model for undirected graphs. The proposed graph reduction technique preserves the node degree sequence from the original hypergraph. The modularity function can be defined on a thus reduced graph, which can be maximized using any standard modularity maximization method, such as the Louvain method. We additionally propose an iterative technique that provides refinement over the obtained clusters. We demonstrate both the efficacy and efficiency of our methods on several real-world datasets.

Keywords: Hypergraph clustering, Hypergraph modularity, Null model

Introduction

The graph clustering problem involves dividing a graph into multiple sets of nodes, such that the similarity of nodes within a cluster is higher than the similarity of nodes belonging to different clusters (Schaeffer 2007; Sankar et al. 2015; Wang et al. 2017; Satuluri and Parthasarathy 2009). While most graph clustering approaches assume pairwise relationships between entities, many real-world network systems involve entities that engage in more complex, multi-way relations. In such systems, modeling all relations as pairwise can lead to a loss of information. The representational power of pairwise graph models

is insufficient to capture higher-order information and present it for analysis or learning tasks.

These systems can be more precisely modeled using *hypergraphs* where nodes represent the interacting components, and hyperedges capture higher-order interactions (Bretto and et al. 2013; Klamt et al. 2009; Satchidanand et al. 2014; Lung et al. 2018). A hyperedge can capture a multi-way relation; for example, in a co-authorship network, where nodes represent authors, a hyperedge could represent a group of authors who collaborated for a common paper. If this were modeled as a graph, we would be able to see which two authors are collaborating, but would not see if multiple authors worked on the same paper. This suggests that the hypergraph representation is not only more information-rich but is also conducive to higher-order learning tasks by virtue of its structure. Indeed, there is a recently expanding interest in research in learning on hypergraphs (Zhang et al. 2018; Kumar et al. 2020; Zhao et al. 2018; Saito et al. 2018; Feng et al. 2018; Chodrow and Mellor 2019).

Analogous to the graph clustering task, *Hypergraph clustering* seeks to discover densely connected components within a hypergraph (Schaeffer 2007). This has been the subject of several research works by various communities with applications to various problems such as VLSI placement (Karypis and Kumar 1998), discovering research groups (Kamiński et al. 2019), image segmentation (Kim et al. 2011), de-clustering for parallel databases (Liu and Wu 2001) and modeling eco-biological systems (Estrada and Rodriguez-Velazquez 2005), among others. A few early works on hypergraph clustering (Leordeanu and Sminchisescu 2012; Bulo and Pelillo 2013; Agarwal et al. 2005; Shashua et al. 2006; Liu et al. 2010) are confined to k -uniform hypergraphs where each hyperedge connects exactly k number of nodes. However, most of the real-world hypergraphs have arbitrary-sized hyperedges, which makes these methods unsuitable for several practical applications. Within the machine learning community, Zhou et al. (2007), were among the earliest to look at learning on non-uniform hypergraphs. They sought to support *spectral clustering* methods (for example see Shi and Malik (2000); Ng et al. (2002)) on hypergraphs and defined a suitable hypergraph Laplacian for this purpose. This effort, like many other existing methods for hypergraph learning, makes use of a reduction of the hypergraph to a graph (Agarwal et al. 2006) and has led to follow-up work (Louis 2015). Spectral based methods involve expensive computations to determine the eigenvector (multiple eigenvectors in case of multiple clusters), which makes these methods less suitable for large hypergraphs.

An alternative methodology for clustering on simple graphs (those with just dyadic relations) is *modularity maximization* (Newman 2006). This class of methods, in addition to providing a useful metric for evaluating cluster quality through the *modularity* function, also returns the number of clusters automatically and avoids the expensive eigenvector computation step - typically associated with other popular methods such as spectral clustering. In practice, a greedy optimization algorithm known as the Louvain method (Blondel et al. 2008) is commonly used, as it is known to be fast and scalable and can operate on large graphs.

Kindly note that this paper is a significantly extended version of our work titled *A New Measure of Modularity in Hypergraphs: Theoretical Insights and Implications for Effective Clustering* (Kumar et al. 2019), presented at *The 8th International Conference on Complex Networks and their Applications*.

However, extending the modularity function to hypergraphs is a non-trivial task, as a node-degree preserving null model would be required, analogous to the graph setting. A straightforward procedure would be to leverage clique reduction, to reduce a hypergraph to a simple graph and then apply a conventional modularity-based solution. Such an approach ignores the underlying super-dyadic nature of interactions and thus loses critical information. Additionally, a clique reduction method would not preserve the node degree sequence of the original hypergraph, which is vital for the null model that modularity maximization techniques are typically based on.

Recently, there have been several attempts to define the null models on the hypergraphs. Chodrow (2019) proposed a Monte Carlo Markov Chain based method, in which random hypergraphs are generated by pairwise reshuffling the edges in the bipartite projection. A more recent study involves the generalization of the celebrated Chung-Lu random graph model (Chung and Lu 2002) to hypergraphs, and employs it to solve the problem of hypergraph clustering (Kamiński et al. 2019). The hypergraph modularity objective proposed by Kamiński et al. (2019) only counts the participation of hyperedges completely contained inside a cluster. Though this assumption enables the analytic tractability of the solution, it limits its applicability to real world hypergraphs where hyperedges can be of arbitrary size. There exists a parallel line of inquiry where hypergraphs are viewed as simplicial complexes, and null models are defined through the preservation of topological features of interest (Giusti et al. 2016; Courtney and Bianconi 2016; Young et al. 2017). Such models make a strong assumption - that of subset-inclusion¹, which may not hold often in real-world data.

Unlike edges in graphs, there are different ways to cut a hyperedge. Depending on where a hyperedge is cut, the proportion and assignments of nodes on different sides of the cut will change, influencing the resultant clustering (Veldt et al. 2020). One way of incorporating the information from the hypergraph's structural properties is to introduce weights along hyperedges. These weights can be determined based on a measure or a function of the input data. For example researchers (Satchidanand et al. 2015) have used the Hellinger distance to weight hyperedges for transductive inference tasks. While this is a supervised metric, one can also consider unsupervised hyperedge weighting schemes that incorporate hyperedge information. One way of incorporating information based on properties of hyperedges or their vertices, is to introduce hyperedge weights based on a metric or function of the data. Building on this idea, we make the following contributions in this work :

- (“Hypergraph modularity” section): We define a null model for graphs generated by hypergraph reduction that preserves the hypergraph node degree sequence. Using this null model and the proposed reduction, we define a modularity function that can be used in conjunction with the popular Louvain method to find hypergraph clusters.
- (“Iterative hyperedge reweighting” section): We propose a generic iterative refinement procedure for hypergraph clustering. This refinement is done by reweighting hyperedges and operates natively on the hypergraph structure.

¹Subset inclusion assumes that, for each hyperedge, any subset of the nodes is also a hyperedge. For example, if authors (A, B, C, D) publish a paper together and form a hyperedge in a co-authorship hypergraph, subset-inclusion would also include all possible subsets such as (A, B), (B, C, D), etc. as observed hyperedges, which may not hold in the real-world datasets.

- (“[Evaluation on ground truth](#)” section): We perform extensive experiments with the resultant algorithm, titled *Iteratively Reweighted Modularity Maximization* (IRMM), on a wide range of real-world datasets and demonstrate both its efficacy and efficiency over state-of-the-art methods. We empirically establish that the hypergraph based methods perform better than their graph-based counterparts.
- (“[Results and analysis](#)” section): We investigate the effect of the reweighting procedure and show that the proposed refinements indeed help us to achieve balanced hyperedge cuts. Furthermore, the experimental results demonstrate that the proposed iterative scheme helps achieve better results over their equivalent non-iterative methods on all datasets.
- (“[Results and analysis](#)” section): We examine the scalability of the hypergraph modularity maximization algorithm using synthetic data.

Background

Hypergraphs

Let V be a finite set of nodes and E be a collection of subsets of V that are collectively exhaustive. For a $w \in \mathbb{R}_+^{|E|}$, $G = (V, E, w)$ is a hypergraph, with vertex set V and hyperedge set E . Each hyperedge e has a positive weight $w(e)$ associated with it. The number of vertices can be denoted by $n = |V|$ and the number of hyperedges can be denoted by $m = |E|$.

While a traditional graph edge has just two nodes, a hyperedge can connect multiple nodes. For a vertex v , we can write its degree as $d(v) = \sum_{e \in E, v \in e} w(e)$. The degree of a hyperedge e is the count of nodes it contains; we can write this as $\delta(e) = |e|$.

The hypergraph incidence matrix H is given by $h(v, e) = 1$ if vertex v is in hyperedge e , and 0 otherwise. W , D_v and D_e are the hyperedge weight matrix, vertex degree matrix and edge degree matrix respectively; W and D_e are diagonal matrices of size $m \times m$, and D_v is a diagonal matrix of size $n \times n$.

Clique Reduction: For a given hypergraph, one can compute its *clique reduction* (Hadley et al. 1992) by substituting each hyperedge with a clique induced by its node-set. For a hypergraph with incidence matrix H , the adjacency matrix of its clique reduction can be written as:

$$A^{clique} = HWH^T$$

To remove the self-loops, we may subtract D_v from the above expression. The resultant clique reduction becomes $A^{clique} = HWH^T - D_v$.

Modularity

When clustering graphs, it is desirable to cut as few edges (or edges with lesser weights in case of weighted graphs) within a cluster as possible. Modularity is a metric of clustering quality that measures whether the number of within-cluster edges is greater than its expected value. In Newman (2006) the modularity function is defined as:

$$\begin{aligned} Q &= \frac{1}{2m} \sum_{ij} [A_{ij} - P_{ij}] \delta(g_i, g_j) \\ &= \frac{1}{2m} \sum_{ij} B_{ij} \delta(g_i, g_j) \end{aligned} \quad (1)$$

Here, $\delta(\cdot)$ is the Kronecker delta function, and g_i, g_j are the clusters to which vertices i and j belong. The $\frac{1}{2m}$ will be dropped for the remainder of this work because it is a constant (number of edges) for a given graph and doesn't affect the maximization of Q . $B_{ij} = A_{ij} - P_{ij}$ is called the modularity matrix. A_{ij} denotes the actual, and P_{ij} denotes the expected number of edges between node i and node j , given by a *null model*. For graphs, the *configuration model* (Newman 2010) is used, where edges are drawn randomly while keeping the node-degree preserved. For two nodes i and j , with (weighted) degrees k_i and k_j respectively, the expected number of edges between them is hence given by:

$$P_{ij} = \frac{k_i k_j}{\sum_{j \in V} k_j}$$

Since the total number of edges in a given network is fixed, maximizing the number of within-cluster edges is the same as minimizing the number of between-cluster edges. This suggests that clustering can be achieved by *modularity maximization*. Kindly note that in this article, we focussed on modularity as defined by Newman (2006). Other definitions of modularity (Courtney and Bianconi 2016) are not in the scope of this work.

Hypergraph modularity

One possible way to define hypergraph modularity is to introduce a hypergraph null model and utilize it to define a modularity function. Kaminski et al. (Kamiński et al. 2019) follow this approach and use a generalized version of the Chung-Lu model (Chung and Lu 2002) to define hypergraph modularity. The proposed modularity function only counts the participation of hyperedges entirely contained inside a cluster. Moreover, the modularity function requires separate processing of hypergraphs induced by hyperedges with different cardinalities. Though such assumptions can provide the analytic tractability of the solution, they limit its applicability to real-world hypergraphs where the hypergraphs can be of very large size with varying hyperedge cardinalities.

Another possible way to define hypergraph modularity is to convert the hypergraph to an appropriate graph and then define modularity on the resultant graph. Such an approach can get benefits from the already existing tools for graphs. In this section, we will follow the latter approach to introduce hypergraph modularity.

To introduce the hypergraph modularity, we start by proposing a null model on the graphs generated by reducing hypergraphs. In a reduced graph, we desire the nodes to possess the same degree as that of the original hypergraph. In a thus reduced graph, the expected number of edges connecting nodes i and j can be given as

$$P_{ij}^{hyp} = \frac{d(i) \times d(j)}{\sum_{v \in V} d(v)} \quad (2)$$

The proposed null model can be interpreted as a mechanism to generate random graphs where the node degree sequence of a given hypergraph is preserved irrespective of the count and cardinality of hyperedges. In order to define a modularity matrix, we need to obtain a graph reduction where the node degree sequence should remain preserved. One straightforward way could be to use a clique reduction of the original hypergraph. However, during clique reduction, the degree of a node in the resultant graph does not remain the same as its degree in the original hypergraph, as verified below.

Lemma 1 For the clique reduction of a hypergraph with incidence matrix H , the degree of a node i in the reduced graph is given by:

$$k_i = \sum_{e \in E} H(i, e)w(e)(\delta(e) - 1)$$

where $\delta(e)$ and $w(e)$ are the degree and weight of a hyperedge e respectively.

Proof For the clique reduction, the adjacency matrix of the resultant graph is given by

$$A^{clique} = HWH^T$$

$$(HWH^T)_{ij} = \sum_{e \in E} H(i, e)w(e)H(j, e)$$

In the resultant graph, each node has a self-loop that can be removed, since they are not cut during the clustering process. This is achieved by explicitly setting $A_{ii}^{clique} = 0$ for all i . Considering this, the degree of a node i in the resultant graph can be written as:

$$\begin{aligned} k_i &= \sum_j A_{ij}^{clique} \\ &= \sum_j \sum_{e \in E} H(i, e)w(e)H(j, e) \\ &= \sum_{e \in E} H(i, e)w(e) \sum_{j: j \neq i} H(j, e) \\ &= \sum_{e \in E} H(i, e)w(e)(\delta(e) - 1) \end{aligned}$$

□

From the above lemma, we can infer that in the clique reduction of a hypergraph, the degree of a node is not preserved and for each hyperedge e , it is overcounted by a factor of $(\delta(e) - 1)$. We can hence scale down the node degree in the reduced graph by a factor of $(\delta(e) - 1)$. This results in the following reduction equation,

$$A^{hyp} = HW(D_e - I)^{-1}H^T \quad (3)$$

We can now verify that the above adjacency matrix preserves the hypergraph node degree.

Proposition 1 For the reduction of a hypergraph given by the adjacency matrix $A^{hyp} = HW(D_e - I)^{-1}H^T$, the degree of a node i in the reduced graph (denoted k_i) is equal to its degree $d(i)$ in the original hypergraph.

Proof We have,

$$(HW(D_e - I)^{-1}H^T)_{ij} = \sum_{e \in E} \frac{H(i, e)w(e)H(j, e)}{\delta(e) - 1}$$

Following a similar argument from the previous theorem, we can explicitly set $A_{ii}^{hyp} = 0$ for all i . The degree of a node in the reduced graph can be written as

$$\begin{aligned} k_i &= \sum_j A_{ij}^{hyp} \\ &= \sum_{e \in E} \frac{H(i, e)w(e)}{\delta(e) - 1} \sum_{j: j \neq i} H(j, e) \\ &= \sum_{e \in E} H(i, e)w(e) \\ &= d(i) \end{aligned}$$

□

With Eq. 3, we can reduce a given hypergraph to a weighted graph and zero out its diagonals by explicitly setting the diagonal entries to zero. The hypergraph modularity matrix can subsequently be written as,

$$B_{ij}^{hyp} = A_{ij}^{hyp} - P_{ij}^{hyp}$$

This new modularity matrix can be used in Eq. 1 to obtain an expression for the hypergraph modularity and can then be used in conjunction with a Louvain-style algorithm.

$$Q^{hyp} = \frac{1}{2m} \sum_{ij} B_{ij}^{hyp} \delta(g_i, g_j) \quad (4)$$

Fundamental observations:

- B^{hyp} exhibits all spectral properties of an undirected weighted graph's modularity matrix (Bolla et al. 2015; Fasino and Tudisco 2016).
- As with any undirected weighted graph (Blondel et al. 2008), Q^{hyp} ranges from -1 to $+1$.
- A negative value of Q^{hyp} indicates a clustering assignment, where a node pair (i, j) from the same cluster participates in lesser than the expected number of hyperedges. This situation may arise when the number of within-cluster edges is lower than the number of across cluster edges.
- A positive value of Q^{hyp} indicates a clustering assignment, where a node pair (i, j) from the same cluster participates in more than the expected number of hyperedges. In graphs, typically, a modularity value higher than 0.3 is considered to be significant (Clauset et al. 2004).
- $Q^{hyp} = 0$ indicates a clustering assignment, where a node pair (i, j) from the same cluster participates in the expected number of hyperedges. This situation can occur because of the random assignment of nodes to the clusters.

In the rest of the section, we will analyze the properties of the proposed modularity function. We will relate the graph reduction equation to the random walk model for hypergraphs. The relation establishes the link with earlier works on hypergraph clustering, where the random walk strategies were employed (Zhou et al. 2007).

Connection to random walks:

Consider the clique reduction of the hypergraph. We can distribute the weight of each hyperedge uniformly among the edges in its associated clique. All nodes within a single hyperedge are assumed to contribute equally; a given node would receive a fraction of the weight of each hyperedge it belongs to. The number of edges each node is connected to from a hyperedge e is $\delta(e) - 1$. Hence by dividing each hyperedge weight by the number of edges in the clique, we obtain the normalized weight matrix $W(D_e - I)^{-1}$. Introducing this in the weighted clique formulation results in the proposed reduction $A = HW(D_e - I)^{-1}H^T$.

Another way of interpreting this reduction is to consider a random walk on the hypergraph in the following manner -

- pick a start node i
- select a hyperedge e containing i , proportional to its weight $w(e)$
- select a new node from e uniformly (there are $\delta(e) - 1$ choices)

The behaviour described above is captured by the following random walk transition model -

$$P_{ij} = \sum_{e \in E} \frac{w(e)h(i,e)}{d(i)} \frac{h(j,e)}{\delta(e) - 1}$$

$$\implies P = D_v^{-1}HW(D_e - I)^{-1}H^T$$

By comparing the above with the random walk probability matrix for graphs ($P = D^{-1}A$) we can recover the reduction $A = HW(D_e - I)^{-1}H^T$.

Iterative hyperedge reweighting

When clustering graphs, it is desired that edges within clusters are greater in number than edges between clusters. Hence when trying to improve clustering, we look at minimizing the number of between-cluster edges that get cut. For a hypergraph, this would be done by minimizing the total volume of the hyperedge cut (Zhou et al. 2007). Consider the two-clustering problem, where the task is to divide the set V into two clusters S and S^c . Zhou et al. (2007) observed that the volume of the cut ∂S is directly proportional to $\sum_e w(e)|e \cap S||e \cap S^c|$, for a hypergraph whose vertex set is partitioned into two sets S and S^c . For a hyperedge e , which has its vertices in both S and S^c , the product $|e \cap S||e \cap S^c|$ can be interpreted as the number of cut sub-edges within a clique reduction. It can be seen that this product is maximized when the cut is balanced and there are an equal number of vertices in S and S^c . In such a case, there will be $\left(\frac{\delta(e)}{2}\right)^2$ sub-edges getting cut. On the other hand, when all vertices of e go into one partition and the other partition is left empty, the product is zero. Similarly, if one of the vertices of e go into one partition and the other partition contains all $\delta(e) - 1$ vertices, then the product is $\delta(e) - 1$. A min-cut algorithm would favor cuts that are as unbalanced as possible, as a consequence of the minimization of $|e \cap S||e \cap S^c|$. In the sequel, we will present the intuition behind our proposed iterative re-weighting technique followed by its mathematical formulation.

Intuition: While clustering in graphs, when an edge gets cut between two clusters, one of its nodes becomes a member of the first cluster, and the other node becomes part of the second cluster. But in hypergraphs, a hyperedge can get cut in multiple ways. When a hyperedge gets cut, if the majority of its vertices go into the first cluster c_1 and only a

smaller fraction of vertices go into the second cluster c_2 , then it is more likely that the vertices going into second cluster are similar to the rest and should be drawn into the first cluster. On the other hand, if a hyperedge gets cut equally across clusters, then its vertices are equally likely to be part of any cluster; hence it is less informative than a hyperedge that gets an unbalanced cut. Building on this idea, we would want to cut the less informative hyperedges (the ones getting balanced cut), and more informative hyperedges that got unbalanced cut to be left uncut.

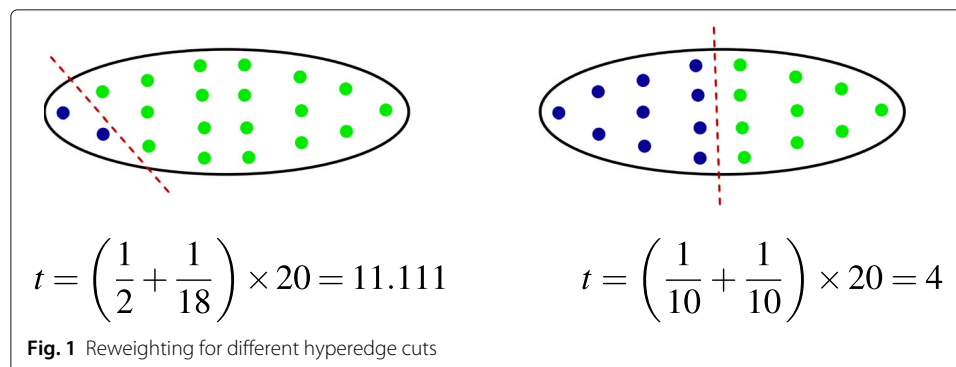
This can be done by increasing the weights of hyperedges that get unbalanced cuts, and (relatively) decreasing the weights of hyperedges that get more balanced cuts. We know that an algorithm that tries to minimize the volume of the hyperedge boundary would try to cut as few heavily weighted hyperedges as possible. Since the hyperedges that had more unbalanced cuts get a higher weight, they are less likely to be cut after reweighting, and instead would reside inside a cluster. Hyperedges that had more balanced cuts get a lower weight, and on reweighting, continue to get balanced cuts. Thus after reweighting and clustering, we would observe fewer hyperedges between clusters, and more hyperedges pushed into clusters. Moreover, after reweighting, we expect that the hyperedges getting cut between clusters should get balanced cuts. In the remaining section, we will formally present the solution mentioned above. Its effectiveness can be seen in the example shown in Fig. 2.

Now, we formally develop a reweighting scheme that satisfies the properties described above - increasing weight for a hyperedge that received a more unbalanced cut, and decreasing weight for a hyperedge that received a more balanced cut. Considering the case where a hyperedge gets partitioned into two clusters with k_1 and k_2 nodes in each partition ($k_1, k_2 \neq 0$), the following equation operationalizes the above mentioned scheme -

$$t = \left(\frac{1}{k_1} + \frac{1}{k_2} \right) \times \delta(e) \quad (5)$$

Here the multiplicative coefficient, $\delta(e)$, seeks to keep t independent of the number of vertices in the hyperedges. Note that for a hyperedge e with two partitions, $\delta(e) = k_1 + k_2$. Figure 1 illustrates an example where t takes two different values depending on the cut.

To see why this satisfies our desired property, note that t is minimized when k_1 and k_2 are equal. It can be verified by the following proposition.



Proposition 2 In the function, $t = \left(\frac{1}{k_1} + \frac{1}{k_2}\right) \times \delta(e)$, the minimum value of $t = 4$, and it is achieved when $k_1 = k_2 = \frac{\delta(e)}{2}$. Here, for a hyperedge e , $\delta(e)$ is its cardinality and k_i represents the number of nodes in the i^{th} partition.

Proof Let $k_i \in \mathbb{Z}^+$

Then,

$$t = \left(\frac{1}{k_1} + \frac{1}{k_2}\right) \times \delta(e)$$

(by substituting $\delta(e) = k_1 + k_2$)

$$\begin{aligned} &= \frac{k_1^2}{k_1 k_2} + \frac{k_2^2}{k_1 k_2} + 2 \\ &= \frac{k_1}{k_2} + \frac{k_2}{k_1} + 2 + (2 - 2) \\ &= \left(\sqrt{\frac{k_1}{k_2}} - \sqrt{\frac{k_2}{k_1}}\right)^2 + 4 \end{aligned}$$

$\left(\sqrt{\frac{k_1}{k_2}} - \sqrt{\frac{k_2}{k_1}}\right)^2$ is minimized when $k_1 = k_2$ and the resultant value of $t = 4$. \square

Note: It can be observed that Eq. 5 coincides with the ratio between arithmetic mean (AM) and harmonic mean (HM) of the two numbers k_1 and k_2 . More precisely, we can write

$$t = 4 \frac{\text{AM}(k_1, k_2)}{\text{HM}(k_1, k_2)}$$

By using the fact that $\text{AM}(k_1, k_2) \geq \text{HM}(k_1, k_2)$, and $\text{AM}(k_1, k_2) = \text{HM}(k_1, k_2)$ only when $k_1 = k_2$, we can obtain the similar result to Proposition 2.

We can then generalize Eq. 5 to c partitions as follows -

$$w'(e) = \frac{1}{m} \sum_{i=1}^c \frac{1}{k_i + 1} [\delta(e) + c] \quad (6)$$

Here, $+1$ term in the denominator accounts for the cases when $k_i = 0$. To compensate for this extra $+1$, $+c$ has been added to the numerator. Additionally, m is the number of hyperedges, and the division by m is added to normalize the weights (Fig. 1). During the first iteration of the algorithm, we find clusters in the hypergraph using its default weights. At the end of the first iteration, we find the updated weights using the Eq. 6. It can be seen that for a hyperedge e if it does not get balanced cut, the $w'(e)$ will not be minimized, and its value will be proportional to the extent to which it gets unbalanced cut. Thus, updating hyperedge weights by Eq. 6 suffices our purpose.

At step $t + 1$, let $w_t(e)$ be the weight of hyperedge e till the previous iteration. Using Eq. 6, $w'(e)$ can be computed for the current iteration. The weight update equation can be written as,

$$w_{t+1}(e) = \alpha w_t(e) + (1 - \alpha) w'(e) \quad (7)$$

Here, α is a hyperparameter which decides the importance to be given to newly calculated weights over the current weights of hyperedges. The complete algorithm for modularity maximization on hypergraphs with iterative reweighting, entitled *Iteratively Reweighted Modularity Maximization (IRMM)*, is described in Algorithm 1. In rest of the

section, we will demonstrate the effectiveness of the hyperedge reweighting scheme by using a toy example.

Algorithm 1: Iteratively Reweighted Modularity Maximization (IRMM)

input : Hypergraph incidence matrix H , vertex degree matrix D_v , hyperedge degree matrix D_e , hyperedge weights W

output: Cluster assignments $cluster_ids$, number of clusters c

```

1 // Initialize weights as  $W \leftarrow I$  if the hypergraph is unweighted
2 repeat
3   // Compute reduced adjacency matrix
4    $A \leftarrow HW(D_e - I)^{-1}H^T$ 
5   // Zero out the diagonals of A
6    $A \leftarrow zero\_diag(A)$ 
7   // Return number of clusters and cluster assignments
8    $cluster\_ids, c = LOUVAIN\_MOD\_MAX(A)$ 
9   // Compute new weight for each hyperedge
10  for  $e \in E$  do
11    // Compute the number of nodes in each cluster
12    for  $i \in [1, \dots, c]$  do
13      // Set of nodes in cluster  $i$ 
14       $C_i \leftarrow cluster\_assignments[i]$ 
15       $k_i = |e \cap C_i|$ 
16    end
17    // Compute new weight
18     $w'(e) = \frac{1}{m} \sum_{i=1}^c \frac{1}{k_i+1} (\delta(e) + c)$ 
19    // Take moving average with previous weight
20     $W_{prev}(e) \leftarrow W(e)$ 
21     $W(e) = \frac{1}{2} (w'(e) + W_{prev}(e))$ 
22  end
23 until  $\|W - W_{prev}\| < threshold$ 

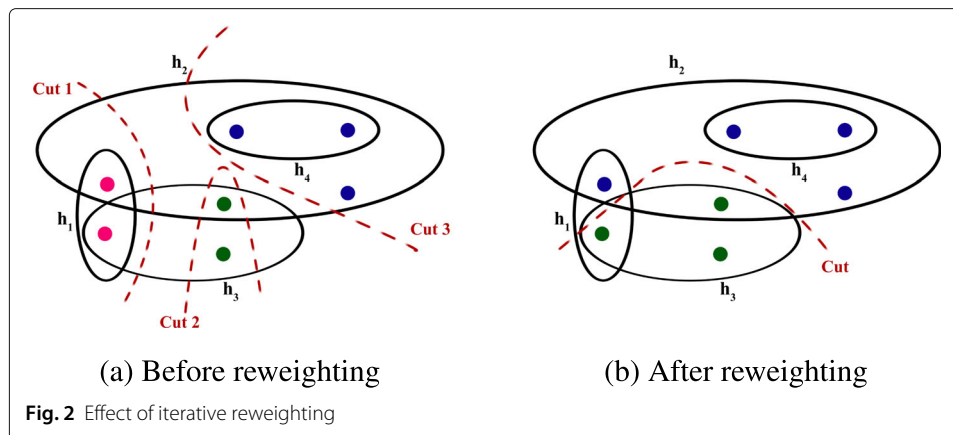
```

A simple example

Figure 2 illustrates the change in hyperedge cuts on a toy hypergraph for a single iteration.

Initially when clustering this hypergraph by modularity maximization, the hypergraph had two highly unbalanced cuts. In Fig. 2a, hyperedge h_2 gets splitted by Cut 1, Cut 2 and Cut 3 in 1 : 4, 1 : 4 and 2 : 3 ratios respectively. Similarly, hyperedge h_3 gets cut by both Cut 1 and Cut 2 in ratio 1 : 2. After applying one iteration hyperedge reweighting, hyperedge h_1 gets split in a 1 : 1 ratio and h_2 gets cut in a 1 : 4 ratio (Fig. 2b). In this case, hyperedge reweighting procedure decreases the number of cuts and leaves two desired clusters. With the intital clustering, there were single nodes left out from hyperedges h_2 and h_3 , which are pulled back into the larger clusters after reweighting. This example illustrates that the reweighting scheme exhibits the desired behavior, as discussed earlier in this section.

We are now in a position to evaluate our ideas empirically.



Evaluation on ground truth

In this section, we will present the experiments conducted to validate the proposed methods. We used the Rand Index, average F1 measure (Yang and Leskovec 2012) and purity, three popular metrics to evaluate the clustering quality. We will start with a brief introduction to the Louvain method, followed by details on the experimental setup and datasets used.

The Louvain method: The Louvain method is a greedy optimization method for detecting communities in large networks (Blondel et al. 2008). The method works on the principle of grouping the nodes that maximize the overall modularity. Since checking all possible cluster assignments is impractical, the Louvain algorithm uses a heuristic that is known to work well on real-world graphs. The method starts by assigning each node to its own cluster and merging those clusters, resulting in the highest modularity gain. Merged clusters are treated as single nodes, and again those cluster-pairs merge that result in the highest modularity gain. If there are no cluster pairs left that will further increase the overall network modularity, the algorithm stops and returns the clusters.

Fixing the number of clusters: We use the Louvain algorithm to maximize the hypergraph modularity as per Eq. 4. Since this method uses a node-degree-preserving graph reduction, we refer to it as *NDP-Louvain* (Node Degree Preserving Louvain). Louvain algorithm automatically returns the number of clusters. To get a predefined number of clusters c , we use agglomerative clustering (Ding and He 2002) on the top of clusters obtained by the Louvain algorithm. For the linkage criterion, we use the average linkage. It is a bottom-up hierarchical clustering method. The algorithm constructs a dendrogram that exhibits pairwise similarity among clusters. At each step, two clusters with the shortest distance are merged into a single cluster. The distance between any two clusters c_i and c_j is taken to be the average distance of all distances $d(x, y)$, where node $x \in c_i$ and node $y \in c_j$.

The proposed methods are shown in the results table as *NDP-Louvain* and *IRMM*.

Settings for IRMM

We investigate the effect of the hyperparameter α using a grid search over the set $[0.1, 0.9]$ with a step size of 0.1. We did not observe any difference in the resultant Rand Index, purity, and F1 scores. While tuning the α , we witnessed a very minimal difference in the

convergence rate, over a wide range of values (for example, 0.3 to 0.9 on the TwitterFootball dataset). It can be noted that α is a scalar value in a moving average; it will not cause any significant variation in the resulting weights. In our experiments, we decided to set it at $\alpha = 0.5$. We stop the iterations if the difference between the mod of two subsequent weight assignments is less than a set threshold. In our experiments, we set chose to set this threshold at $threshold = 0.01$

Compared methods

To evaluate the performance of our proposed methods, we compared the following baselines.

Clique Reductions: We reduced the original hypergraph using a clique reduction ($A = HWH^T$) and then applied the Louvain method and Spectral Clustering.

Hypergraph-based Spectral Clustering: We use the hypergraph-based spectral clustering method, as defined in Zhou et al. (2007). The given hypergraph is reduced to a graph $\left(A = D_v^{-\frac{1}{2}} H W D_e^{-1} H^T D_v^{-\frac{1}{2}}\right)$ and its Laplacian is calculated. The top k eigenvectors of the Laplacian are found and clustered by the bisecting-k-means clustering procedure. In the results table, this method is referred to as *Zhou-Spectral*.

PaToH² and hMETIS³: These are popular hypergraph partitioning algorithms that work on the principles of coarsening the hypergraph before partitioning. The coarsened hypergraph is partitioned using expensive heuristics. In our experiments, we used the original implementations from the corresponding authors.

Datasets

Dataset statistics are furnished in Table 1. For all datasets, we use the largest connected component of the hypergraph for our experiments. All the datasets are classification datasets, where the class labels accompany the data points. We use these class labels as the proxy for clusters. The detailed description of the hypergraph construction is given below:

MovieLens⁴: This is a multi-relational dataset provided by GroupLens research, where movies are represented by nodes. We construct a co-director hypergraph by using the *director* relationship to represent hyperedges. A hyperedge would connect a group of nodes if the same individual directed them. Here, the genre of a movie represents the class of the corresponding node.

Cora and Citeseer: These are bibliographic datasets, where the nodes represent papers. In each dataset, a set of nodes is connected by a hyperedge if they involve the same set of words (after removing low frequency and stop words). Different disciplines were used as clusters (Sen et al. 2008).

TwitterFootball: This is a social network taken from the Twitter dataset (Greene et al. 2012). This dataset involves players of 20 football clubs (classes) of the English Premier League. Here, the nodes represent players, and if a set of players are co-listed, then the corresponding nodes are connected by a hyperedge.

Arnetminer: This is a large bibliographic dataset (Tang et al. 2008). Here, the nodes represent papers, and a set of nodes are connected if the corresponding papers are

²<http://bmi.osu.edu/umit/software.html>

³<http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download>

⁴<http://ir.ii.uam.es/hetrec2011/datasets.html>

Table 1 Dataset description

Dataset	# nodes	# hyperedges	Avg. hyperedge degree	Avg. node degree	# classes
TwitterFootball	234	3587	15.491	237.474	20
Cora	2708	2222	3.443	2.825	7
Citeseer	3264	3702	27.988	31.745	6
MovieLens	3893	4677	79.875	95.961	2
Arnetminer	21375	38446	4.686	8.429	10

co-cited. The nodes in the hypergraph are accompanied by Computer Science sub-disciplines. Different sub-disciplines were used as clusters.

Experiments

For the different datasets, we compare the Rand Index (Rand 1971), purity (Manning et al. 2008), and average F1 scores (Yang and Leskovec 2013) on all the methods discussed earlier. The number of clusters was first set to that returned by the Louvain method, in an unsupervised fashion. This is what would be expected in a real-world setting, where the number of clusters is not given apriori. Table 2 shows the results of this experiment.

Secondly, we ran the same set of methods with the number of ground truth classes set as the number of clusters. In the case of Louvain method, the clusters obtained are merged

Table 2 Rand Index, Purity and Average F1 scores against ground truth; the number of clusters for hMETIS, PaToH, Spectral, and Zhou-Spectral is set to the number of clusters returned by the IRMM method are 13, 79, 8, 18, and 1358 for Citeseer, Cora, Movielens, TwitterFootball, and Arnetminer, respectively

	Citeseer	Cora	MovieLens	TwitterFootball	Arnetminer
(a) Rand Index scores against ground truth.					
hMETIS	0.6504	0.7592	0.4970	0.7639	0.0416
PaToH	0.6612	0.6919	0.4987	0.7553	0.0052
Spectral	0.7164	0.2478	0.4806	0.7486	0.0610
Zhou-Spectral	0.8210	0.5743	0.4977	0.9016	0.0628
Louvain	0.7361	0.7096	0.4898	0.6337	0.0384
NDP-Louvain	0.7899	0.8238	0.4988	0.9056	0.0821
IRMM	0.7986	0.8646	0.5091	0.9448	0.0967
(b) Purity scores against ground truth.					
hMETIS	0.5894	0.6596	0.6893	0.2556	0.6831
PaToH	0.6271	0.5912	0.7017	0.3176	0.3928
Spectral	0.4629	0.3897	0.6832	0.8114	0.9216
Zhou-Spectral	0.5287	0.4145	0.7118	0.8325	0.9378
Louvain	0.7190	0.6836	0.7189	0.8054	0.9138
NDP-Louvain	0.7307	0.7597	0.7245	0.8829	0.9691
IRMM	0.7659	0.8138	0.7291	0.8948	0.9765
(c) Average F1 scores against ground truth.					
hMETIS	0.1087	0.1075	0.1291	0.3197	0.0871
PaToH	0.0532	0.1171	0.1104	0.1132	0.0729
Spectral	0.1852	0.1291	0.1097	0.4496	0.0629
Zhou-Spectral	0.2774	0.2517	0.118	0.5055	0.0938
Louvain	0.1479	0.2725	0.1392	0.2238	0.1378
NDP-Louvain	0.2782	0.3248	0.1447	0.5461	0.1730
IRMM	0.4019	0.3709	0.1963	0.5924	0.1768

Louvain, NDP-Louvain, and IRMM return the number of clusters on their own
Best performance in each column is boldfaced

using the post-processing technique explained earlier. The results of this experiment are given in Table 3. On some datasets, the *Louvain* method and IRMM return fewer clusters than the number of ground truth classes. In such cases, we do not report the results and leave the entries as “-”.

We also plotted the results for varying number of clusters using the same methodology described above, to assess our method’s robustness. The results are shown in Fig. 3. In all datasets but *Arnetminer*, we set the number of clusters to a minimum value such as two and then increase it by a factor of two. For *Arnetminer*, since the IRMM method returns a very large number of clusters, we set the initial number of clusters to ten and increase it by a factor of ten. For all datasets, the maximum number of clusters is set to the number of clusters returned by the IRMM method. On some datasets, *Louvain* and *NDP-Louvain* methods return a fewer number of clusters than IRMM. In such cases, the corresponding curves in Fig. 3 are left truncated.

Results and analysis

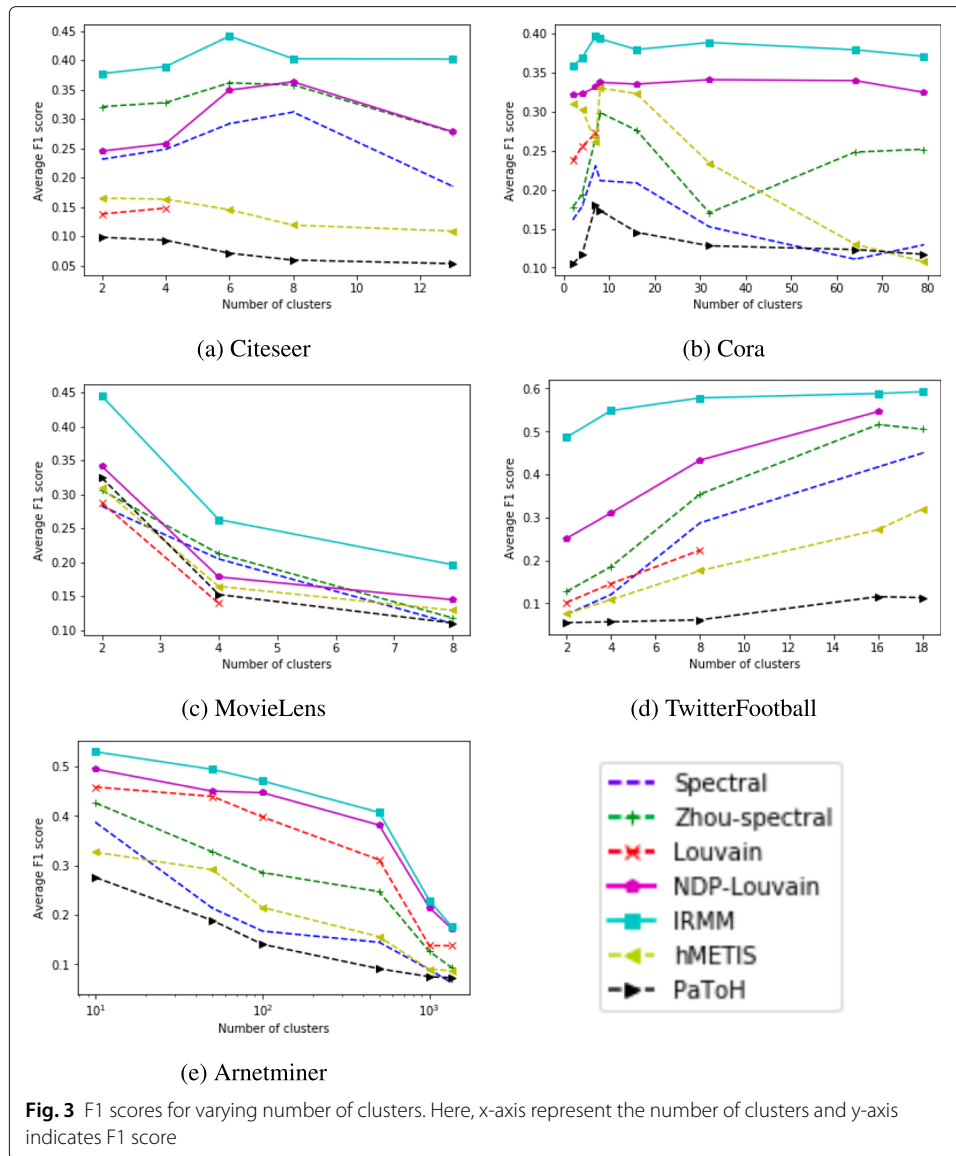
We show that the proposed methods - *NDP-Louvain* and IRMM perform consistently better on all the datasets (except on one dataset with RI measure). To test the robustness

Table 3 Rand Index, Purity and Average F1 scores against ground truth; the number of clusters is set to the number of ground truth classes

	Citeseer	Cora	MovieLens	TwitterFootball	Arnetminer
(a) Rand Index scores; number of clusters set to the number of ground truth classes					
hMETIS	0.6891	0.7853	0.5028	0.7697	0.3116
PaToH	0.7312	0.7208	0.4984	0.7618	0.1820
Spectral	0.7369	0.3117	0.4812	0.7765	0.3762
Zhou-Spectral	0.8267	0.5845	0.5006	0.9112	0.3851
Louvain	-	0.7096	0.4982	-	0.4198
NDP-Louvain	0.8197	0.8441	0.5119	-	0.5359
IRMM	0.8245	0.889	0.5347	-	0.5506
(b) Cluster purity scores; number of clusters set to the number of ground truth classes					
hMETIS	0.5249	0.6359	0.6914	0.2354	0.2984
PaToH	0.5724	0.6498	0.7139	0.2419	0.2391
Spectral	0.4839	0.5819	0.7294	0.7815	0.5169
Zhou-Spectral	0.5374	0.6115	0.742	0.8191	0.5827
Louvain	-	0.7136	0.7364	-	0.4837
NDP-Louvain	0.7495	0.7441	0.7429	-	0.5968
IRMM	0.7732	0.779	0.7737	-	0.6173
(c) Average F1 scores; number of clusters set to the number of ground truth classes					
hMETIS	0.1451	0.2611	0.4445	0.3702	0.3267
PaToH	0.071	0.1799	0.3239	0.1036	0.2756
Spectral	0.2917	0.2305	0.2824	0.4345	0.387
Zhou-Spectral	0.3614	0.2672	0.3057	0.5377	0.4263
Louvain	-	0.2725	0.2874	-	0.4587
NDP-Louvain	0.3491	0.3314	0.3411	-	0.4948
IRMM	0.441	0.3966	0.4445	-	0.5299

Citeseer, Cora, Movielens, TwitterFootball, and Arnetminer have 6, 7, 2, 20, and 10 classes, respectively. On some datasets, the *Louvain* and IRMM method return fewer clusters than the number of ground truth classes. In such cases, we do not report the results and leave the entries as “-”.

Best performance in each column is boldfaced



of the proposed method, we vary the number of clusters and report the results in the latter half of the section. To investigate the effect of the reweighting scheme, we report the distribution of the sizes of hyperedges getting cut. This is followed by testing the scalability of the proposed algorithm against one of the competitive baseline. We will start by discussing the empirical evaluation of the proposed methods.

From the Tables 2 and 3, it is evident that IRMM gives the highest cluster purity scores and average F1 scores across all the datasets and the highest Rand Index scores are obtained on all except *Citeseer* dataset. Besides the fact that IRMM significantly outperforms over other methods, we want to emphasize on the following two observations:

Superior performance of hypergraph based methods:

It is evident that hypergraph based methods perform consistently better than their clique based equivalents. Results indicate that *Zhou-Spectral* and *NDP-Louvain* are better than *Spectral* and *Louvain* respectively. Hence, preserving the super-dyadic structure helps in getting a better cluster assignment.

The proposed iterative reweighting scheme helps to boost up the performance:

The proposed hyperedge reweighting scheme aids in the performance across all datasets. It must be noted that the first iteration of IRMM is the *NDP-Louvain* and IRMM performance is consistently better than the *NDP-Louvain* method, which shows that balancing the hyperedge cut enhances the cluster quality.

Effect of reweighting on hyperedge cuts

Consider a hyperedge that is cut; its nodes partitioned into different clusters. Looking at Eq. 6, we can see that $w'(e)$ is minimized when all the partitions are of equal size, and maximized when one of the partitions is much larger than the other. The iterative reweighting procedure is designed to increase the number of hyperedges with balanced partitioning, and decrease the number of hyperedges with unbalanced partitioning. As iterations pass, hyperedges that are more unbalanced should be pushed into neighbouring clusters, and the hyperedges that lie between clusters should be more balanced.

We analyze the effect of hyperedge reweighting in Fig. 4. For each hyperedge, we find the relative proportion of the biggest partition and add them in the bins with interval size = 0.1. The plot illustrates the variation in the size of each bin over along with iterations.

$$\text{relative size}(e) = \max_i \frac{\text{number of nodes in cluster } i}{\text{number of nodes in the hyperedge } e}$$

If a hyperedge is a balanced cut, then the proportion of its largest partition is low; we call such hyperedges as *fragmented*. On the other hand, if a hyperedge has a very high proportion of its largest partition, then the hyperedge is not a balanced cut; we call such hyperedges as *dominated*.

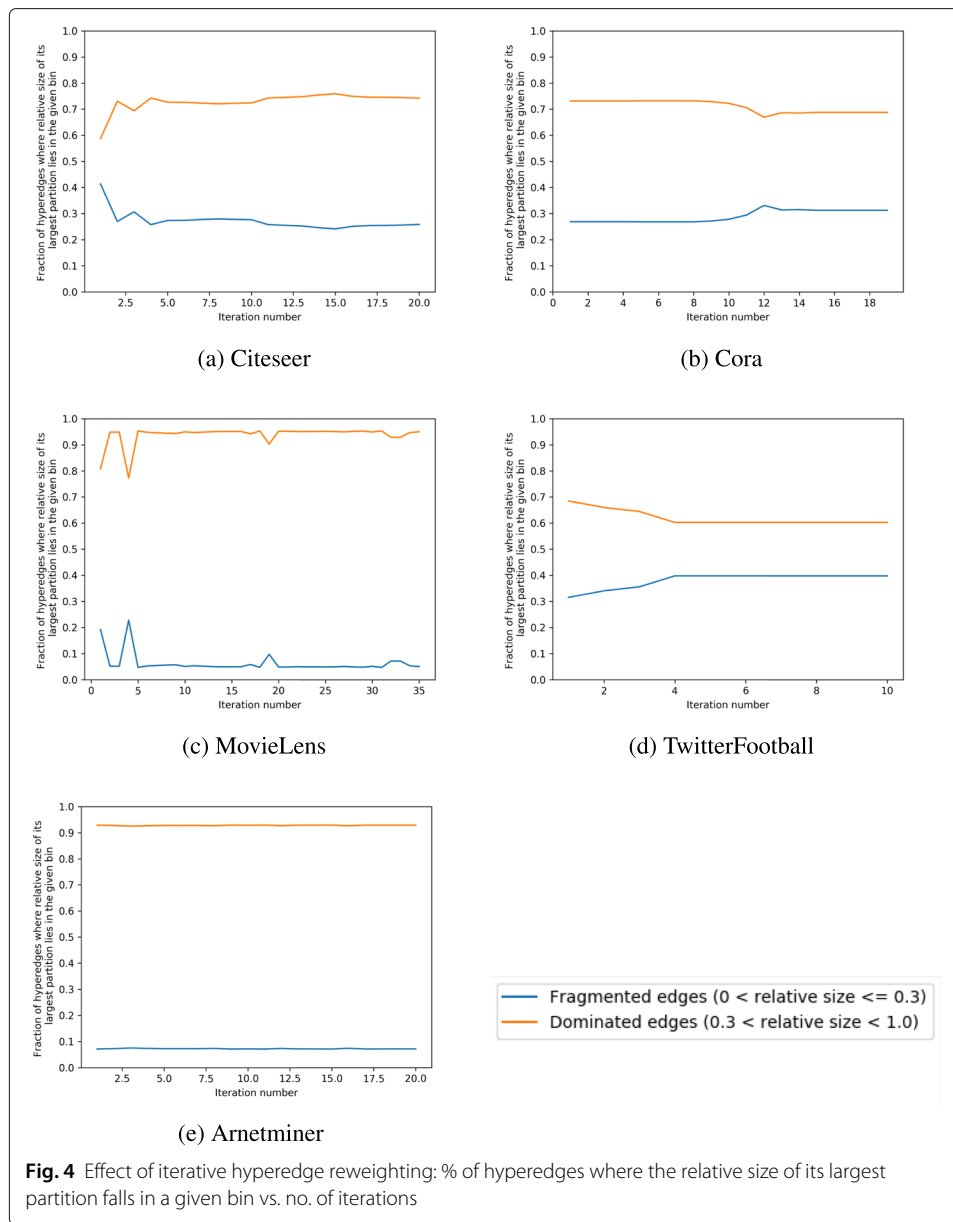
On *TwitterFootball* dataset, the effect of reweighting is distinctly visible as the number of fragmented edges increases with iterations. This behavior confirms our intuition of achieving more balanced cuts with the proposed reweighting procedure. After four iterations, the method converges as we don't observe any change in the hyperedge distribution.

A similar trend is observed with the *Cora* dataset. Here, the number of fragmented edges fluctuate before their final convergence.

In the case of *Arnetminer* dataset, the change in fragmented and dominated edges is very minimal. One possible reason for such behavior could be its significantly large size as compared to the number of ground truth clusters.

In the case of *Citeseer* and *Movielens* datasets, we could not see the convergence in the change of hyperedge weights in a pre-fixed number of iterations. Though the number of hyperedges seems to fluctuate with iterations, the algorithm tries to find the best clustering at each step by using the *NDP-Louvain* algorithm. This results in the improved performance of the overall algorithm after following the refinement procedure.

Both in *Citeseer* and *Movielens* datasets, IRMM returns lesser number of clusters than *NDP-Louvain*. *NDP-Louvain* returns 16 clusters for *Citeseer* and 13 clusters for *Movielens* dataset. These number of clusters are reduced to 13 and 8 for *Citeseer* and *Movielens* datasets respectively. Thus, the refinement procedure tends to minimize the cut value along with cut-balancing.



Scalability of the *NDP-Louvain* method

To further motivate the extension of modularity maximization methods to the hypergraph clustering problem, we look at the scalability of the *NDP-Louvain* method against the strongest baseline, *Zhou-Spectral*. Table 4 shows the CPU times⁵ for the *NDP-Louvain* and *Zhou-Spectral* on the real-world datasets. We see that while the difference is less pronounced on a smaller dataset like *TwitterFootball*, it is much greater on the larger datasets. In particular, the runtime on Arnetminer for *NDP-Louvain* is lower by a significant margin, not having to compute an expensive eigendecomposition.

⁵The runtime of IRMM is not reported as it is highly dependent on the number of iterations. For some datasets such as *TwitterFootball* and *Cora*, our method converged in 4 and 13 iterations, respectively. For remaining datasets, we experimented with the number of iterations set to 20.

Table 4 CPU times (in seconds) for the hypergraph clustering methods on all datasets

	Citeseer	Cora	MovieLens	TwitterFootball	Arnetminer
Zhou-Spectral	84.16	41.44	155.8	3.88	34790
NDP-Louvain	41.21	24.23	35.9	3.32	4311.2

Note: To compute the eigenvectors for spectral clustering based method, we use of the *eig(.)* function from MATLAB. The *eig(.)* function makes use of orthogonal similarity transformations to convert the matrix into upper Hessenberg matrix followed by QR algorithm to find its eigenvectors.

Analysis on synthetic hypergraphs: On the real-world data, modularity maximization showed improved scalability as the dataset size increased. To evaluate this trend, we compared the CPU times for the *Zhou-Spectral* and *NDP-Louvain* methods on synthetic hypergraphs of different sizes. For each hypergraph, we first ran *NDP-Louvain* and found the number of clusters returned, then ran the *Zhou-Spectral* method with the same number of clusters.

Following the hypergraph generation method used in EDRW: Extended Discriminative Random Walk⁶ (Satchidanand et al. 2015), we generated hypergraphs with 2 classes and a homophily of 0.4 (40% of the hyperedges deviate from the expected class distribution). The hypergraph followed a modified power-law distribution, where 75% of its hyperedges contained less than 3% of the nodes, 20% of its hyperedges contained 3%-50% of the nodes, and the remaining 5% contained over half the nodes in the dataset. To generate a hypergraph, we first set the number of hyperedges to 1.5 times the number of nodes. For each hyperedge, we sampled its size k from the modified power-law distribution and chose k different nodes based on the homophily of the hypergraph. We generated hypergraphs of sizes ranging from 1000 nodes up to 10000 nodes, at intervals of 500 nodes.

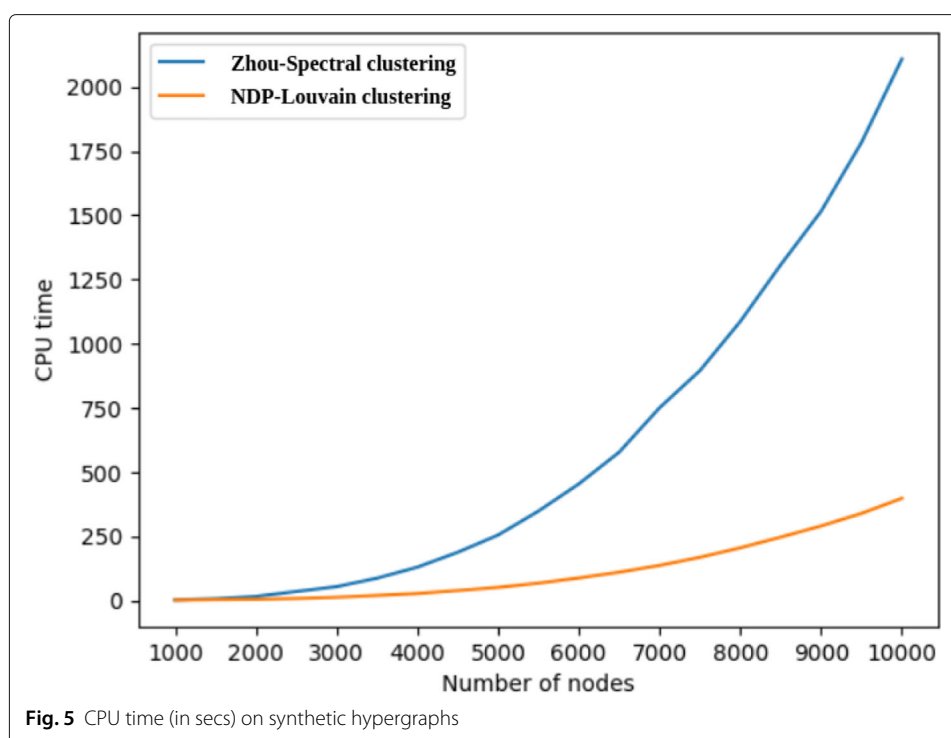
Figure 5 shows how the CPU time varies with the number of nodes, on the synthetic hypergraphs generated as given above.

While *NDP-Louvain* is shown to run consistently faster than *Zhou-Spectral* for the same number of nodes, the difference increases as the hypergraph grows larger. In Fig. 5, this is shown by the widening in the gap between the two curves as the number of nodes increases.

Conclusion and future directions

In this paper, we have defined the problem of clustering on hypergraphs and state challenges involved to solve it. We start with defining a null model for the graphs generated by the hypergraph reduction and theoretically show its equivalence to the configuration model defined for weighted undirected graphs. Our proposed graph reduction technique preserves the node degree sequence of the actual hypergraph. After reducing the hypergraph to a graph, we apply the Louvain algorithm to find clusters. We have motivated the problem of balancing the hypergraph cuts and provided an iterative solution for the same. Our extensive set of experiments demonstrates the supremacy of the proposed methods over state-of-the-art approaches. The promising results confirm the need for hypergraph modeling and open up new directions for further research. The proposed graph reduction technique can be used for different tasks such as node classification, link prediction, node representation learning, etc., that are left as the avenues of future research.

⁶<https://github.com/HariniA/EDRW>



Abbreviations

IRMM: Iteratively reweighted modularity maximization; NDP-Louvain: Node degree preserving Louvain; EDRW: Extended discriminative random walk

Acknowledgements

We would like to thank Manikandan Narayanan and Deepak Maurya for their valuable suggestions.

Authors' contributions

TK, SV, SP and BR contributed to the mathematical analysis, design of algorithms and the experimental design. TK with assistance from SV, led the implementation. HA contributed to the inception of the graph reduction and hyperedge reweighting methods. All the authors contributed to the discussions. TK and SV wrote the initial manuscript and all authors edited, read and approved the final manuscript.

Funding

This work was partially supported by Intel research grant RB/18-19/CSE/002/INTI/BRAV to BR. SP would like to acknowledge the United States National Science Foundation under grants: SES-1949037 CCF-1629548 and EAR-1520870; and the United States National Institutes of Health under the grant: NIH-1R01 HD088545-01A1. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

Availability of data and materials

The code and sample data is available at github.com/tarunkumariitm/IRMM.

Competing interests

The authors declare that they have no competing interests.

Author details

¹Robert Bosch Centre for Data Science and AI, Chennai, India. ²Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India. ³Present Address: Currently at University of Massachusetts Amherst, Amherst, Massachusetts, USA. ⁴Present Address: Currently at Google, Bangalore, India. ⁵Department of Computer Science and Engineering, The Ohio State University, Columbus, USA.

Received: 17 March 2020 Accepted: 4 August 2020

Published online: 20 August 2020

References

Agarwal S, Branson K, Belongie S (2006) Higher order learning with graphs. In: ICML'06: Proceedings of the 23rd International Conference on Machine Learning. Association for Computing Machinery, New York, pp 17–24. <https://doi.org/10.1145/1143844.1143847>

- Agarwal S, Lim J, Zelnik-Manor L, Perona P, Kriegman D, Belongie S (2005) Beyond pairwise clustering. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 2. IEEE Computer Society, USA. pp 838–8452. <https://doi.org/10.1109/CVPR.2005.89>
- Blondel VD, Guillaume J-I, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. *J Stat Mech Theory Exp* 2008(10):10008. <https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/meta>
- Bolla M, Bullins B, Chaturapruek S, Chen S, Friedl K (2015) Spectral properties of modularity matrices. *Linear Algebra Appl* 473:359–376
- Bretto A, et al. (2013) *Hypergraph Theory: An Introduction*. Springer Publishing Company, Incorporated. <https://dl.acm.org/doi/book/10.5555/2500991>
- Bulo SR, Pelillo M (2013) A game-theoretic approach to hypergraph clustering. *IEEE Trans Pattern Anal Mach Intell* 35(6):1312–1327
- Chodrow PS (2019) Configuration models of random hypergraphs and their applications. arXiv preprint arXiv:1902.09302
- Chodrow P, Mellor A (2019) Annotated hypergraphs: Models and applications. arXiv preprint arXiv:1911.01331
- Chung F, Lu L (2002) Connected components in random graphs with given expected degree sequences. *Ann Comb* 6(2):125–145
- Clauset A, Newman ME, Moore C (2004) Finding community structure in very large networks. *Phys Rev E* 70(6):066111
- Courtney OT, Bianconi G (2016) Generalized network structures: The configuration model and the canonical ensemble of simplicial complexes. *Phys Rev E* 93(6):062311
- Ding C, He X (2002) Cluster merging and splitting in hierarchical clustering algorithms. In: 2002 IEEE International Conference on Data Mining, 2002. Proceedings. IEEE Computer Society, USA. pp 139–146. <https://dl.acm.org/doi/10.5555/844380.844756>
- Estrada E, Rodriguez-Velazquez JA (2005) Complex networks as hypergraphs. arXiv preprint physics/0505137
- Fasino D, Tudisco F (2016) Generalized modularity matrices. *Linear Algebra Appl* 502:327–345
- Feng F, He X, Liu Y, Nie L, Chua T-S (2018) Learning on partial-order hypergraphs. In: Proceedings of the 2018 World Wide Web Conference. WWW '18. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland. pp 1523–1532
- Giusti C, Ghrist R, Bassett DS (2016) Two's company, three (or more) is a simplex. *J Comput Neurosci* 41(1):1–14
- Greene D, Sheridan G, Smyth B, Cunningham P (2012) Aggregating content and network information to curate twitter user lists. In: Proceedings of the 4th ACM RecSys Workshop on Recommender Systems and the Social Web. RSWeb '12. ACM, New York. pp 29–36
- Hadley SW, Mark BL, Vannelli A (1992) An efficient eigenvector approach for finding netlist partitions. *IEEE Trans Comput Aided Des Integr Circ Syst* 11(7):885–892
- Kamiński B, Poulin V, Prałat P, Szufel P, Théberge F (2019) Clustering via hypergraph modularity. *PLoS ONE* 14(11):e0224307. <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0224307>
- Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput* 20(1):359–392
- Kim S, Nowozin S, Kohli P, Yoo CD (2011) Higher-order correlation clustering for image segmentation. In: Advances in Neural Information Processing Systems. Curran Associates Inc., Red Hook. pp 1530–1538
- Klamt S, Haus U-U, Theis F (2009) Hypergraphs and cellular networks. *PLoS Comput Biol* 5(5):e1000385. <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000385>
- Kumar T, Darwin K, Parthasarathy S, Ravindran B (2020) HPRA: Hyperedge prediction using resource allocation. In: 12th ACM Conference on Web Science. Association for Computing Machinery, New York. pp 135–143. <https://doi.org/10.1145/3394231.3397903>
- Kumar T, Vaidyanathan S, Ananthapadmanabhan H, Parthasarathy S, Ravindran B (2019) A new measure of modularity in hypergraphs: Theoretical insights and implications for effective clustering. In: International Conference on Complex Networks and Their Applications. Springer International Publishing, Cham. pp 286–297. https://link.springer.com/chapter/10.1007/978-3-030-36687-2_24
- Leordeanu M, Sminchisescu C (2012) Efficient hypergraph clustering. In: Lawrence ND, Girolami M (eds). Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 22. PMLR, La Palma. pp 676–684
- Liu H, Latecki LJ, Yan S (2010) Robust clustering as ensembles of affinity relations. In: Advances in Neural Information Processing Systems
- Liu D-R, Wu M-Y (2001) A hypergraph based approach to declustering problems. *Distrib Parallel Databases* 10(3):269–288
- Louis A (2015) Hypergraph markov operators, eigenvalues and approximation algorithms. In: Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing. STOC '15. ACM, New York. pp 713–722
- Lung RI, Gaskó N, Suciu MA (2018) A hypergraph model for representing scientific output. *Scientometrics* 117(3):1361–1379
- Manning CD, Schütze H, Raghavan P (2008) *Introduction to information retrieval*. Cambridge University Press, USA
- Newman ME (2006) Modularity and community structure in networks. *Proc Natl Acad Sci* 103(23):8577–8582
- Newman ME (2010) *Networks: An Introduction*. Oxford University Press, Inc., New York
- Ng AY, Jordan MI, Weiss Y (2002) On spectral clustering: Analysis and an algorithm. In: Advances in Neural Information Processing Systems. MIT Press, Cambridge. pp 849–856
- Rand WM (1971) Objective criteria for the evaluation of clustering methods. *J Am Stat Assoc* 66(336):846–850
- Saito S, Mandic D, Suzuki H (2018) Hypergraph p-laplacian: A differential geometry view. In: AAAI Conference on Artificial Intelligence. AAAI Press. <http://dblp.unitrier.de/db/conf/aaai/aaai2018.html#SaitoMS18>
- Sankar V, Ravindran B, Shivashankar S (2015) Ceil: A scalable, resolution limit free approach for detecting communities in large networks. In: Twenty-Fourth International Joint Conference on Artificial Intelligence. AAAI Press, Buenos Aires. <https://dl.acm.org/doi/10.5555/2832415.2832540>
- Satchidanand SN, Ananthapadmanabhan H, Ravindran B (2015) Extended discriminative random walk: a hypergraph approach to multi-view multi-relational transductive learning. In: Twenty-Fourth International Joint Conference on Artificial Intelligence. AAAI Press, Buenos Aires. <https://dl.acm.org/doi/10.5555/2832747.2832778>

- Satchidanand SN, Jain SK, Maurya A, Ravindran B (2014) Studying indian railways network using hypergraphs. In: 2014 Sixth International Conference on Communication Systems and Networks (COMSNETS). IEEE. pp 1–6. <http://dblp.unitrier.de/db/conf/comsnets/comsnets2014.html#SatchidanandJMR14>
- Saturluri V, Parthasarathy S (2009) Scalable graph clustering using stochastic flows: applications to community discovery. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009. Association for Computing Machinery, New York. pp 737–746. <https://doi.org/10.1145/1557019.1557101>
- Schaeffer SE (2007) Graph clustering. *Comput Sci Rev* 1(1):27–64
- Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T (2008) Collective classification in network data. *AI Mag* 29(3):93
- Shashua A, Zass R, Hazan T (2006) Multi-way clustering using super-symmetric non-negative tensor factorization. In: Proceedings of the 9th European Conference on Computer Vision - Volume Part IV. ECCV'06. Springer, Berlin. pp 595–608
- Shi J, Malik J (2000) Normalized cuts and image segmentation. *IEEE Trans Pattern Anal Mach Intell* 22(8):888–905
- Tang J, Zhang J, Yao L, Li J, Zhang L, Su Z (2008) Arnetminer: Extraction and mining of academic social networks. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '08. Association for Computing Machinery, New York. pp 990–998
- Veldt N, Benson AR, Kleinberg J (2020) Hypergraph cuts with general splitting functions. *arXiv preprint arXiv:2001.02817*
- Wang C, Pan S, Long G, Zhu X, Jiang J (2017) Mgae: Marginalized graph autoencoder for graph clustering. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17. Association for Computing Machinery, New York. pp 889–898
- Yang J, Leskovec J (2012) Defining and evaluating network communities based on ground-truth. In: Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics. MDS '12. ACM, New York. pp 3–138
- Yang J, Leskovec J (2013) Overlapping community detection at scale: a nonnegative matrix factorization approach. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. Association for Computing Machinery, New York. pp 587–596. <https://doi.org/10.1145/2433396.2433471>
- Young J-G, Petri G, Vaccarino F, Patania A (2017) Construction of and efficient sampling from the simplicial configuration model. *Phys Rev E* 96(3):032312
- Zhang M, Cui Z, Jiang S, Chen Y (2018) Beyond link prediction: Predicting hyperlinks in adjacency space. In: AAAI Conference on Artificial Intelligence. AAAI Press. <http://dblp.unitrier.de/db/conf/aaai/aaai2018.html#ZhangCJC18>
- Zhao X, Wang N, Shi H, Wan H, Huang J, Gao Y (2018) Hypergraph learning with cost interval optimization. In: AAAI Conference on Artificial Intelligence
- Zhou D, Huang J, Schölkopf B (2007) Learning with hypergraphs: Clustering, classification, and embedding. In: Advances in Neural Information Processing Systems. MIT Press, Cambridge. pp 1601–1608

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)