Hypergraph Simultaneous Generators

Bahman Pedrood
George Mason University

Carlotta Domeniconi
George Mason University

Kathryn B. Laskey
George Mason University

Abstract

Generative models for affiliation networks condition the edges on the membership of their nodes to communities. The problem of community detection under these models is addressed by inferring the membership parameters from the network structure. Current models make several unrealistic assumptions to make the inference feasible, and are mostly designed to work on regular graphs that cannot handle multi-way connections between nodes. While the models designed for hypergraphs attempt to capture the latter, they add further strict assumptions on the structure and size of hyperedges and are usually computationally intractable for real data. This paper proposes an efficient probabilistic generative model for detecting overlapping communities that process hyperedges without any changes or restrictions on their size. Our model represents the entire state space of the hyperedges, which is exponential in the number of nodes. We develop a mathematical computation reduction scheme that reduces the inference time to linear in the volume of the hypergraph without sacrificing precision. Our experimental results validate the effectiveness and scalability of our model and demonstrate the superiority of our approach over state-of-the-art community detection methods.

1 INTRODUCTION

Graph theory enables the modeling of connections, relations, and dependencies among entities. It has been extensively used to study and analyze complex networks in a wide variety of domains, such as computer science, biology, telecommunications, digital circuits, and social sciences. Graphs are characterized by a set of *nodes*, which represent individuals or entities, and *edges*, which represent connections between pairs of nodes. Graphs are appropriate to model problems in which connections are between pairs of nodes. However, some problems are characterized by larger units of connection. For instance, in a co-authorship network, the nodes represent authors and the connections represent papers written by any number of authors. To capture this kind of connection in a graph, the n-way connection is typically reduced to pairwise connections among the authors. A *hypergraph* is an extension of a regular graph that preserves higher-order connection information by utilizing *hyperedges* that can connect any number of nodes. As such, for the co-authorship graph, hypergraphs enable representing a paper as a single hyperedge, thus avoiding information loss. Note that one can represent a hypergraph using a bipartite graph by associating each hyperedge with a new node of type "hyper edge" and connecting all the hypergraph nodes adjacent to the hyperedge to the new node. Since the hypergraph representation fits better to our context, and to avoid any confusions with a specific bipartite graph that will be discussed later in this paper, we use the former representation.

Community detection is an important and widely studied problem in networks. A community is typically defined as a collection of densely connected nodes. Different objective functions have been introduced to guide the discovery of communities (Fortunato and Hric, 2016), leading to fast, reliable, and scalable methods (Yang et al., 2013; Yang and Leskovec, 2013, 2014; Abbe, 2017; Grover and Leskovec, 2016). Community detection has also been studied in hypergraphs, but progress has been limited. Most methods either implicitly or explicitly transform a hypergraph into a regular graph (Agarwal et al., 2005; Hagen and Kahng, 1992; Agarwal et al., 2006; Veldt et al., 2020), or formulate a set of relaxations to define a simplified hypergraph in terms of its graph counterpart (Zhou et al., 2007; Hein et al., 2013; Yang et al., 2019; Huang et al., 2019; Li and Milenkovic, 2017; Chodrow, 2019; Chodrow and Mellor, 2020), and then find commu-

Proceedings of the 25$^{\text{th}}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

nities therein using popular graph-based community detection methods. Other approaches constrain the size of hyperedges to some fixed value and apply a graph community detection technique to the hypergraph's adjacency d-tensor (Bulò and Pelillo, 2009; Ghoshdastidar and Dukkipati, 2014; Kim et al., 2017; Leordeanu and Sminchisescu, 2012). Given the aforementioned limitations, existing hypergraph methods are not widely applied to real-world data.

In this paper, we introduce *Hypergraph Simultaneous Generators* (HySGen), a probabilistic generative model for discovering overlapping communities in hypergraphs. Unlike previous work on community detection in hypergraphs, HySGen directly leverages the higher order relations captured by the hyperedges, without enforcing constraints that may distort the connection information. Our generative model follows the assumption of affiliation networks (Lattanzi and Sivakumar, 2009) for the causality relation between communities and [hyper]edges, and our inference method estimates the degree of affiliation of the nodes to the overlapping communities. The proposed generative model and inference method are inspired by Yang and Leskovec (2013), but include several key distinctions. Although we define the affiliation of a node to a community directly as a probability measure, unlike LDA-based approaches, we do not impose any constraint that limits the membership to a community based on the memberships to other communities. This enables the inference process to find any degree of overlap among communities. Furthermore, the model accepts input hypergraphs with hyperedges of any size. While this makes the complexity of the model exponential in the number of nodes, we propose an algebraic reduction scheme that provides a solution with complexity proportional to the sum of the hyperedges' degrees. Our experiments on synthetic and real-world networks demonstrate the effectiveness of our model in discovering ground-truth communities and its superiority compared to popular baseline methods.

2 RELATED WORK

Generative models allow modeling complex assumptions about an underlying distribution of data in a clear and interpretable way. Although they have been extensively employed for community detection in graphs (Jin et al., 2021; Abbe, 2017), progress in developing such models for hypergraphs has been limited. One reason might be the computational burden of inference algorithms, especially given the exponential size of the space of hyperedges. For this reason, most previous work has placed strict constraints on hyperedge size, and/or made substantial relaxations and approximations (Angelini et al., 2015; Ke et al., 2019). For example, Kim et al. (2018) proposed a method that only works on k-uniform, sparse, assortative hypergraphs, and only when there are two equally sized communities. Recently, Chodrow et al. (2021) tackled the computational intensity of their inference by converting the procedure to maximizing a modularity measure (Chodrow et al., 2021). Further, the authors limit the hyperedge size in their experiments to a maximum of 4, and their modularity maximization approach includes reducing the hypergraph to a clique transformed graph. Moreover, almost all the current generative approaches for community detection in hypergraphs formulate their model by extending a variation of *Stochastic Block Models (SBM)* for regular graphs, so they can only partition the nodes, rather than discovering overlapping communities.

Another class of generative models for graphs is based on affiliation network models (Lattanzi and Sivakumar, 2009). Affiliation networks provide a random model for real-world social networks. They consist of a bipartite graph that shows the connections between *actors* and communities, and of a second graph that shows the connection between pairs of actors; the two graphs evolve together in the original model. One of the most influential graph community detection methods based on affiliation networks is BigCLAM (Yang and Leskovec, 2013). BigCLAM introduces a generative model on the bipartite graph to discover overlapping communities based on the connections in the actors' graph. Despite its advantages, this model makes ad-hoc choices (e.g. edge probability formulation), which result in poor interpretability and weak theoretical underpinning. To the best of our knowledge, HySGen is the first community detection method for hypergraphs based on an affiliation network model, and it does not suffer from the aforementioned limitations of BigCLAM. Unlike any other hypergraph generative models for community detection, HySGen is not based on SBMs, it does not constrain the size of hyperedges, and its generative model and inference algorithm do not have any kind of lossy relaxations or approximations. Nonetheless, it performs the iterations for inferring $N \times C$ node membership parameters as fast as $O(C \times vol_{\mathcal{H}})$, where N is the number of nodes, C is the number of communities, and $vol_{\mathcal{H}}$ is the sum of degrees of the hyperedges. We explain the details of HySGen in the next section.

3 GENERATIVE MODEL

A network under our model is an undirected, unweighted hypergraph $\mathcal{H}(V, E)$ with N nodes $V = \{v_1, ..., v_N\}$ and M hyperedges $E = \{e_1, ..., e_M\}$. A hyperedge $e \in E$ represents a connection between a set of m nodes, where $m \geq 2$. We define the degree of a hy-

peredge $\deg(e)$ as the number of nodes in e, and the degree of a node $\deg(v)$ as the number of hyperedges that have v as a member. The *volume* of the hypergraph $vol_{\mathcal{H}}$ is the sum of the degrees of all the hyperedges or the nodes: $vol_{\mathcal{H}} = \sum_{e \in E} deg(e) = \sum_{v \in V} deg(v)$. A hyperedge space Υ in our model is the set of all *potential hyperedges* $\Upsilon = \{e | e \subseteq V, |e| \geq 2\}$. While E is a subset of Υ that exist in $\mathcal{H}$, the set of *non-hyperedges* $\bar{E}$ is defined as the subset of Υ that do not exist in $\mathcal{H}$, which can be seen as the complement of E with respect to Υ: $\bar{E} = \{\bar{e} | \bar{e} \in \Upsilon, \bar{e} \notin E\}$. To improve the clarity of formulations later in this section, for any node v we also define the set $E_v = \{e | e \in E, v \in e\}$ as the subset of edges that include node v, and similarly the set $\bar{E}_v = \{\bar{e} | \bar{e} \in \bar{E}, v \in \bar{e}\}$ as the subset of non-hyperedges that include v. An *Extended* hyperedge space Υ' is also defined similarly by adding the isolated nodes in the hyperedge space $\Upsilon' = \{x | x \subseteq V, |x| \geq 1\} = \{\Upsilon \cup V\}$, and will be used later in section 5.

For each $\varepsilon \in \Upsilon$, we define an indicator variable h_ε, which equals 1 when $\varepsilon \in E$ and 0 otherwise. We assume there are C communities $[C] = \{1, ..., C\}$ in the network, and the primary goal in this paper is to discover those communities. Each node $v \in V$ may be simultaneously affiliated to any community $c \in [C]$ with probability $S_{vc} \in \mathbb{R}^{[0,1]}$, which also specifies the strength or weight of this affiliation. $S \in \mathbb{R}^{[0,1]}_{N \times C}$ is the set of all the community membership parameters, where $S_{vc} = 0$ shows no affiliation and $S_{vc} = 1$ shows the strongest affiliation. The proposed generative model assumes that the joint affiliation of any subset ε of nodes to a community c is positively associated with their tendency to connect together and form a hyperedge. The probability of community c generating a hyperedge for ε is given by:

$$\pi_{\varepsilon c} = \prod_{v \in \varepsilon} S_{vc} \tag{1}$$

With this formulation, for a community c to have a high chance of generating a hyperedge for ε, all the nodes in ε must have a high affiliation to c. The probability in (1) decreases dramatically if at least one of the nodes has a low weight. Big hyperedges are discouraged by this function, but large clusters of nodes with strong enough affiliation to a community are still likely to create a hyperedge.

We assume the nodes can be members of more than one community at the same time. We develop our generative model in a way to allow many communities to simultaneously contribute in generating a hyperedge. This means that in order for a hyperedge to connect the nodes in ε, it must be generated by at least one of the communities. A direct formulation of the overall probability of generating a hyperedge based on this assumption requires incorporating $2^C - 1$ possible cases where each community does or does not contribute in generating the hyperedge. Instead, we formulate this probability with only C terms considering the fact that the probability of *not* generating a hyperedge corresponds to the joint probability that *none* of the communities contribute. Incorporating the independence assumption for the effect of each community, this probability is computed as:

$$P(h_\varepsilon = 1) = 1 - \prod_{c=0}^{C} (1 - \pi_{\varepsilon c}) \tag{2}$$

This kind of formulation has been previously used in the context of Bayesian networks, where it is known as noisy-OR model (Heckerman, 1993). The formulation in (2) also includes a background probability for generating random hyperedges that is represented here as a *null community* with index $c = 0$, to which every node has a fixed small affiliation. This allows some hyperedges to be generated without having an affiliation to any of the target communities.

Figure 1 is a graphical representation of our hypergraph model. We also assumed a *truncated gamma* prior distribution for the parameters in S with shape α and scale λ^{-1}, and we set the random variable upper bound to 1. The generative process for creating the hyperedges is specified as follows:

1. **Repeat:** Do for *every* node v and community $c \in \{1, ...C\}$: Choose $S_{vc} \sim \text{TruncatedGamma}(\alpha, \lambda^{-1} | S_{vc} \leq 1)$.
2. **Repeat:** Do for *every* potential hyperedge $\varepsilon \in \Upsilon$:
   (a) Compute $P_\varepsilon = P(h_\varepsilon = 1)$ as specified in (2).
   (b) Choose hyperedge $e \sim \text{Bernouli}(P_\varepsilon)$.

Given a hypergraph $\mathcal{H}$, the problem of discovering the communities will be to infer the parameters in S, which is discussed in the next section.

4 INFERENCE

Exact inference of the posterior distribution $P(S|E)$ in a fully Bayesian setting is intractable. As we discuss in section 5, even a point estimate solution for S based on a direct formulation of the problem is not scalable. Nonetheless, we start developing our inference algorithm with a direct formulation of the log-posterior distribution of the model to obtain a Maximum A-Posteriori (MAP) solution for S. We set the shape parameter of the prior distribution α to 1, and to avoid any randomness bias toward certain hyperedges, we give equal background probabilities to all the nodes ($\forall v \in V : S_{v0} = S_0$). Additionally, since learning

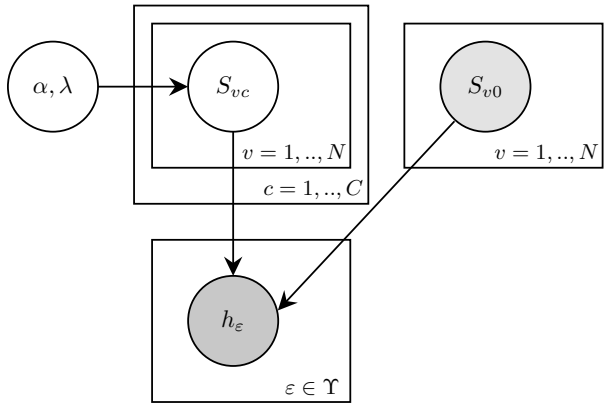

Figure 1: Plate model displays a graphical representation of the proposed generative model for a given hypergraph. Shaded circles are the observations and fixed parameters, and white circles are the model parameters. For each potential hyperedge ε, the community membership weights S_{vc} of the nodes $v \in \varepsilon$ specify the probability of generating a hyperedge for ε. The parameters S_{v0} show the affiliation of the nodes to a null community that represents a background probability for generating random hyperedges.

S_0 from the data interprets the null community as a target community and captures some shared pattern among the nodes, we set it as a fixed parameter. This parameter's precise value assignment will be explored in detail in appendix C. Our solution is determined by maximizing the log-posterior distribution of S:

$$\hat{S} = \underset{0 \leq S \leq 1}{\operatorname{argmax}} \left\{ \log P(E|S) - \lambda \sum_{v \in V} \sum_{c=1}^{c} S_{vc} + f(\lambda, S_0) \right\} \tag{3}$$

where $f(\lambda, S_0)$ is the part of the prior $\log P(S|\lambda)$ that does not depend on the target parameters and can be removed without affecting the solution. The term with the λ coefficient corresponds to the prior distribution of the hyperedges. Our choice of prior formulation has made it equivalent to applying a L_1-regularization on the values of S that encourages sparsity of the solution. We define $\mathcal{L}_{\mathcal{H}}(S)$ as the part of the log posterior that is used in the computation for optimizing (3):

$$\begin{aligned} \mathcal{L}_{\mathcal{H}}(S) = &\sum_{e \in E} \log \left(1 - \prod_{c=0}^{C} \left(1 - \prod_{v \in e} S_{vc} \right) \right) \\ &+ \sum_{\bar{e} \in \bar{E}} \sum_{c=0}^{C} \log \left(1 - \prod_{v \in \bar{e}} S_{vc} \right) - \lambda \sum_{v \in V} \sum_{c=1}^{C} S_{vc} \end{aligned} \tag{4}$$

$\mathcal{L}_{\mathcal{H}}(S)$ is concave in each vector $S_{v.}$, assuming the values of S as constant for the rest of the nodes. As such, we apply gradient ascent on each $S_{v.}$ one node at a time, as part of an iterative Block Coordinate Ascent scheme. Under this scheme, we can update a $S_{v.}$ vector by maximizing a subproblem of (4) that only includes the terms with components of $S_{v.}$:

$$\begin{aligned} \mathcal{L}_{\mathcal{H}}(S_{v.}) = &\sum_{\substack{\{e \in E \\ |v \in e\}}} \log \left(1 - \prod_{c=0}^{C} \left(1 - \prod_{u \in e} S_{uc} \right) \right) \\ &+ \sum_{\substack{\{\bar{e} \in \bar{E} \\ |v \in \bar{e}\}}} \sum_{c=0}^{C} \log \left(1 - \prod_{u \in \bar{e}} S_{uc} \right) - \lambda \sum_{c=1}^{C} S_{vc} \end{aligned} \tag{5}$$

where $\mathcal{L}_{\mathcal{H}}(S_{v.})$ is the part of $\mathcal{L}_{\mathcal{H}}$ that depends on $S_{v.}$.

To apply gradient ascent on $S_{v.}$, we compute the gradient vectors $\nabla \mathcal{L}_{\mathcal{H}}(S_{v.}) = [\frac{\partial \mathcal{L}_{\mathcal{H}}(S_{v1})}{\partial S_{v1}}, ..., \frac{\partial \mathcal{L}_{\mathcal{H}}(S_{vC})}{\partial S_{vC}}]$ for node v by calculating the partial derivatives of $\mathcal{L}_{\mathcal{H}}(S_{v.})$ against S_{vc}, for each $c \in [C]$:

$$\begin{aligned} &\frac{\partial \mathcal{L}_{\mathcal{H}}(S_{vc})}{\partial S_{vc}} = \\ &\sum_{\substack{\{e \in E \\ |v \in e\}}} \frac{\prod_{m \in (e - \{v\})} S_{mc} \prod_{b \in ([C] - \{c\})} \left(1 - \prod_{u \in e} S_{ub}\right)}{1 - \prod_{b=0}^{C} \left(1 - \prod_{u \in e} S_{ub}\right)} \\ &- \sum_{\substack{\{\bar{e} \in \bar{E} \\ |v \in \bar{e}\}}} \frac{\prod_{m \in (\bar{e} - \{v\})} S_{mc}}{1 - \prod_{u \in \bar{e}} S_{uc}} - \lambda I_1(S_{vc}) \end{aligned} \tag{6}$$

where $I_1(x)$ is an indicator function which gives the value 1 if $S_{vc} \geq 1$ and 0 if its value is 0. Having (6) computed for all communities, the updating formula for optimizing the rows of S is as follows:

$$S_{v.}^{new} = S_{v.}^{old} + \alpha \Big(\nabla \mathcal{L}_{\mathcal{H}}(S_{v.}) \Big) \tag{7}$$

where α is a learning rate parameter for optimization. Although the updating formula in (7) could potentially be used for the inference, in practice it is computationally too expensive. In fact, any conventional deterministic or approximate inference technique for the parameters in our model needs at least one round of calculations for each hyperedge $e \in E$ and each non-hyperedge $\bar{e} \in \bar{E}$, for a total processing of $2^N - (N+1)$ potential hyperedges. Next we propose a solution to this problem to make our approach scalable for large hypergraphs.

5 COMPUTATION REDUCTION

A one-time computation for any of (4) to (6) includes enumerating the complete hyperedge space Υ and it

Table 1: The list of sets defined for Section 5.

Symbol definition	Description
$\Upsilon_c = \{\varepsilon \vert \varepsilon \in \Upsilon, \forall u \in \varepsilon : S_{uc} > 0\}$	Hyperedge space within community c
$\Upsilon_{vc} = \{\varepsilon \vert \varepsilon \in \Upsilon_c, v \in \varepsilon\}$	Members of Υ_c that have node v
$\overline{\Upsilon}_{vc} = \{\varepsilon \vert \varepsilon \in \Upsilon_c, v \notin \varepsilon\}$	Members of Υ_c that do not have node v
$\Upsilon' = \{x \vert x \subseteq V, \vert x\vert \geq 1\} = \{\Upsilon \cup V\}$	Extended hyperedge space: Includes isolated nodes
$\Upsilon'_c = \{\varepsilon \vert \varepsilon \in \Upsilon', \forall u \in \varepsilon : S_{uc} > 0\}$	Extended hyperedge space within community c
$\Upsilon'_{vc} = \{\varepsilon \vert \varepsilon \in \Upsilon'_c, v \in \varepsilon\}$	Members of Υ'_c that have node v
$\overline{\Upsilon'}_{vc} = \{\varepsilon \vert \varepsilon \in \Upsilon'_c, v \notin \varepsilon\}$	Members of Υ'_c that do not have node v

takes $O(C^2 \times N^2 \times 2^{N-2})$ computations. Each of those equations consists of two summations, one for the hyperedges E and the other for the non-hyperedges $\bar{E}$. Since a hypergraph is very sparse in practice ($M \ll |\Upsilon|$), it is the summation over $\bar{e} \in \bar{E}$ that makes the computational complexity exponential. To address this complication, in this section, we reformulate the summations over $\bar{E}$ to reduce the exponential enumeration. With the new formulations, a value is initialized for those summations once in $O(1)$ and then will be updated, again in $O(1)$ at each optimization iteration. The new formulation is virtually lossless and does not sacrifice any precision. We also apply an initialization-update routine for the observed hyperedges to further reduce the worst-case complexity. This routine, along with a detailed complexity analysis discussion, is explained in appendix B, where we show that these reformulations reduce the overall complexity of our inference algorithm to $O(C \times vol_{\mathcal{H}})$.

5.1 Reformulations

Our goal in this section is to find an alternative formulation for the $\bar{e} \in \bar{E}$ summations in (5) and (6) so that a tractable product of S_{vc} can be factorized out of the summation. With that, we can update the value of those summations in a tractable time after a new S_{vc} value is determined as discussed in section 5.2. We extend our notation in table 1 for a more clear formulation in this section. We start by working toward finding a transformation for (5), then we develop a transformation for (6) in a similar fashion.

We define the function $\Psi(A, c)$ to use for developing our reformulations:

$$\Psi(A, c) = \sum_{\varepsilon \in A} \log\left(1 - \prod_{u \in \varepsilon} S_{uc}\right) \tag{8}$$

where A can be any given subset of the nodes. Using (8) to represent the summation over non-hyperedges, (5) can be rewritten as:

$$\mathcal{L}_{\mathcal{H}}(S_{v\cdot}) = \sum_{\substack{\{e \in E \\ \vert v \in e\}}} \log\left(1 - \prod_{c=0}^{C}\left(1 - \prod_{u \in e} S_{uc}\right)\right) + \sum_{c=0}^{C} \Psi(\tbinom{\{\bar{e} \in \bar{E}}{\vert v \in \bar{e}\}}, c) - \lambda_S \sum_{c=1}^{C} S_{vc} \tag{9}$$

As $\Psi(\binom{\{\bar{e} \in \bar{E}}{\vert v \in \bar{e}\}}, c) = \Psi(\Upsilon_{vc}, c) - \Psi(\binom{\{e \in E}{\vert v \in e\}}, c)$, (5) can be computed in a tractable time if we have a tractable way to compute $\Psi(\Upsilon_{vc}, c)$.

Since $\prod_{u \in \varepsilon} S_{uc}$ is bounded in the range $[0, 1]$, if we prevent it from becoming exactly 1, we can replace $\log\left(1 - \prod_{u \in \varepsilon} S_{uc}\right)$ in (8) with its Taylor expansion around zero (Maclaurin expansion). Note that the only situation in which $\prod_{u \in \hat{\varepsilon}} S_{uc}$ for some $\hat{\varepsilon}$ becomes 1 is if all the nodes in $\hat{\varepsilon}$ have a membership weight of 1 to the community c. But under our model, this can only happen if all subsets of $\hat{\varepsilon}$ with size 2 or more have a corresponding hyperedge in the hypergraph $\mathcal{H}$. Since Ψ is only computed to be used for non-hyperedge summations, in that case we can dismiss including such $\hat{\varepsilon}$ and its subsets from the computations.

Replacing $\log\left(1 - \prod_{u \in \varepsilon} S_{uc}\right)$ in (8) with its equivalent Taylor expansion, we have:

$$\Psi(A, c) = \sum_{\varepsilon \in A} \sum_{n=1}^{n_\infty} \frac{-1}{n}\left(\prod_{u \in \varepsilon} S_{uc}\right)^n = \sum_{n=1}^{n_\infty} \psi_n(A, c) \tag{10}$$

where n_∞ is an upper bound for the order of the series. It theoretically goes to ∞, but in practice the terms in the series converge to 0 quickly and a very accurate estimate is yielded by setting n_∞ to a finite $O(1)$ number. We show the n^{th} term of the expansion in (10) with the function $\psi_n(A, c) = \frac{-1}{n} \sum_{\varepsilon \in A} \left(\prod_{u \in \varepsilon} S_{uc}\right)^n$.

Applying the transformation in (10) to $\Psi(\Upsilon_{vc}, c)$ we have $\Psi(\Upsilon_{vc}, c) = \sum_{n=1}^{n_\infty} \psi_n(\Upsilon_{vc}, c)$. Having the transformed formulation, S_{vc} can be completely factorized out of $\psi_n(\Upsilon_{vc}, c)$:

$$\psi_n(\Upsilon_{vc}, c) = {S_{vc}}^n \psi_n(\overline{\Upsilon'}_{vc}, c) \tag{11}$$

It is trivial to show that for every node v and community c the following equality holds:

$$\psi_n(\Upsilon'_c, c) = \psi_n(\overline{\Upsilon'}_{vc}, c) + \psi_n(\Upsilon_{vc}, c) - \frac{1}{n}{S_{vc}}^n \tag{12}$$

where the last term is added to count for the case of $\varepsilon' = \{v\}$, which is counted in $\psi_n(\Upsilon'_c, c)$, but is neither in $\psi_n(\overline{\Upsilon'}_{vc}, c)$ nor in $\psi_n(\Upsilon_{vc}, c)$. Using (11) to replace the $\psi_n(\overline{\Upsilon'}_{vc}, c)$ in (12) by its equivalent in terms of $\psi_n(\Upsilon_{vc}, c)$, the following equivalency can be derived:

$$\psi_n(\Upsilon_{vc}, c) = \frac{{S_{vc}}^n}{{S_{vc}}^n + 1}(\psi_n(\Upsilon'_c, c) + \frac{1}{n}{S_{vc}}^n) \tag{13}$$

This means by having one value of $\psi_n(\Upsilon'_c, c)$ for each community c, we can compute $\psi_n(\Upsilon_{vc}, c)$ and consequently $\Psi(^{\{\bar{e}\in\bar{E}}_{|v\in\bar{e}\}}, c)$ for every $v \in V$ in $O(1)$. As we show in section 5.2, our formulation provides an $O(1)$ access time to the updated value of $\psi_n(\Upsilon'_c, c)$ for any community c, which lets the computation of the likelihood in (5) to be done in $O(1)$.

We follow a similar procedure to reduce the computation for (6). As explained in appendix A, the non-hyperedge summation for any input in (6) can be computed in $O(1)$ from the same $\psi_n(\Upsilon'_c, c)$ that is calculated for (13) and no additional updating during the iterations are required for it.

5.2 Initialization and Updating

Given the reformulations in section 5.1, we only need a tractable access to $\psi_n(\Upsilon'_c, c)$ to overcome the exponential computations. Our overall scheme to solve this problem is to *initialize* these values in the beginning, store them, and *update* them after any S_{vc} value is updated during the inference iterations. In this section, we provide $O(1)$ solutions to both initialization and updating steps. To avoid terminology confusion in this section, we use derivations of the word *initialize* to refer to initializing $\psi_n(\Upsilon'_c, c)$ values, and *preset* in reference to initializing the S_{vc} values.

The under-determined solution space of our model makes a good preset an integral part of a successful community inference. Our inference method allows any arbitrary preset for the community memberships values S_{vc}. Still, if an informed preset is not available, one can start the learning iterations by randomly assigning nodes to the communities and giving the members an equal arbitrary membership weight.

Any community preset partitions the network into member and non-member nodes for each community c. Then every potential hyperedge ε would have one of the two following states with respect to the community according to (8):

1. $\exists u \in \varepsilon : S_{uc} = 0$, then the term corresponding to ε in $\psi_n(\Upsilon'_c, c)$ collapses to zero.
2. $\forall u \in \varepsilon : S_{uc} > 0$, then the term corresponding to ε in $\psi_n(\Upsilon'_c, c)$ becomes nonzero.

So the initialization of $\psi_n(\Upsilon'_c, c)$ values for each c includes only the possible hyperedges that all the including nodes have a non-zero membership to community c. That means a direct computation of $\psi_n(\Upsilon'_c, c)$ includes summing over all subsets of the nodes in c. To find an $O(1)$ analytical solution, even if the informed community preset gives different membership weights to the nodes, we start with an equal value of S_0 for all $\{S_{vc} | v \in c\}$. Then we can run a round of updating as explained later in (15) to replace S_0 with the specific preset S_{vc} values for each node. Taking this initial value $S_0 > 0$, for each $n \geq 1$ we have:

$$\begin{aligned}\psi_n(\Upsilon'_c,c)^{(0)} &= \frac{-1}{n} \sum_{\{\varepsilon' \in \Upsilon'_c\}} \left(S_0{}^{|e'|}\right)^n \\ &= \frac{-1}{n}\left[\sum_{m=0}^{N_c} \binom{N_c}{m} (S_0{}^n)^m - \binom{N_c}{0}(S_0{}^n)^0\right] \\ &= \frac{-1}{n}\left[(S_0{}^n + 1)^{N_c} - 1\right]\end{aligned} \tag{14}$$

where N_c is the number of nodes in community c.

The next task is to develop an updating formula for $\psi_n(\Upsilon'_c, c)$. Our approximate inference algorithm iteratively updates the values of S_{vc}. At each iteration t, after a new value ${S_{vc}}^{(t)}$s is computed in (7), we need to update $\psi_n(\Upsilon'_c, c)$ with the new values for each $n = 1, \ldots, n_\infty$. Taking $S_{vc}^{(t-1)}$ as the membership value from previous iteration and $S_{vc}^{(t)}$ as the updated value that is computed in current iteration, our goal is to isolate the terms with $S_{vc}^{(t-1)}$ in $\psi_n(\Upsilon'_c, c)$ and replace them with the new values $S_{vc}^{(t)}$ to get the updated value of $\psi_n^{(t)}(\Upsilon'_c, c)$. Constructing on (11) and (12), and the definition of $\psi_n(A, c)$ in (10), the updating formula for $\psi_n(\Upsilon'_c, c)$ is derived as:

$$\begin{aligned}\psi_n^{(t)}(\Upsilon'_c, c) = &\frac{{S_{vc}^{(t)}}^n + 1}{{S_{vc}^{(t-1)}}^n + 1}\left(\psi_n^{(t-1)}(\Upsilon'_c, c)\right. \\ &\left. + \frac{1}{n}{S_{vc}^{(t-1)}}^n\right) - \frac{1}{n}{S_{vc}^{(t)}}^n\end{aligned} \tag{15}$$

6 EXPERIMENTAL RESULTS

6.1 Experimental Design

We implemented[1] our algorithm using *C++* and extended the *SNAP* open-source *C++* library (Leskovec and Sosič, 2016). We also implemented an efficient data structure for hypergraphs to be added to the standard Graph library in SNAP. We show the performance of our model by running it on both synthetic and real-world data. To compare against graph community detection methods, we created a graph for each hypergraph by turning every hyperedge e into a $\deg(e)$-clique. We evaluate the performance of our model against popular community detection methods

[1]Full implementation of HySGen and the data used in our experiments can be accessed at https://github.com/bpedrood/HySGen.

Table 2: Real world datasets used in the experiments. **N**: Number of nodes, **H**: Number of hyperedges, **MH**: Maximum hyperedge size, **E**: Number of edges in the regular graph converted from the hypergraph, **C**: Number of ground truth communities.

Dataset	N	H	MH	E	C
NSF	8,167	6,221	14	38,206	309
Scratch	5,985	5,842	40	139,570	1,174
DBLP	6,129	6,261	9	9,708	14

by measuring the similarity of discovered communities from each method against the ground truth in real-world networks. To quantify the similarities, we use the following performance measure, as done by Yang and Leskovec (2013); Yang et al. (2013):

$$F(\delta, C, C^*) = \frac{1}{2|C^*|} \sum_{C_i^* \in C^*} \max_{C_j \in C} \delta(C_i^*, C_j) + \frac{1}{2|C|} \sum_{C_j \in C} \max_{C_i^* \in C^*} \delta(C_i^*, C_j) \tag{16}$$

where C^* represents the ground truth communities and C the detected communities. $\delta(C_i^*, C_j)$ is the F1 score or Jaccard index between the two sets C_i^* and C_j. Other similarity measures can also be used.

Our model has a large number of parameters; as such, the solution space is under-determined and our inference model is prone to converging to local maxima. A good community initialization helps the inference procedure to find a better solution. Our inference method allows any arbitrary initialization method to determine the initial community membership weights. To avoid any bias and advantage over the baseline methods, we use locally minimized neighborhoods (Gleich and Seshadhri, 2012), which is the same initialization method used in BigCLAM (Yang and Leskovec, 2013). Further details on our optimization process and parameter assignments are given in appendix C.

6.2 Synthetic Data Experiment

The goal of this experiment is to show how HySGen is capable of leveraging the hypergraph structure when no graph community detection method can perform well. We synthesized a hypergraph with 87 nodes and 55 hyperedges for an imaginary example of a school, where the nodes are individuals and hyperedges represent the meetings they attended. There are two ground truth communities: Computer Science and History students, shown as nodes with shades of blue and red, respectively, in fig. 2a. In all the sub-figures of fig. 2, the membership weights of nodes are shown with a heat-map that displays larger weights with darker colors and smaller weights with lighter colors. The yellow nodes in fig. 2a are not students and do not belong to any community. Out of 22 members of the blue community, 15 are senior students with strong affiliation to their community and are shown with dark blue, while 7 are freshmen who had very little activity.The same situation exists with old and new members in the red community. The hyperedges of the network are shown as colored contours around the nodes in fig. 2b. A high density of hyperedges can be observed around the strong members of each community. There are also two big hyperedges of size 43 and 44 that correspond to two welcome parties held by the school and students are assigned randomly to one of the parties. Students are allowed to bring guests and that is why there are several yellow nodes in fig. 2a with no community affiliation. These very large hyperedges carry no meaningful information about the ground-truth communities and are included to investigate the resilience of our method to noisy hyperedges.

Table 3: Performance comparison on real-world datasets over average F1 score and average Jaccard Index measures.

	F1 Score			**Jaccard Index**		
Method	NSF	Scratch	DBLP	NSF	Scratch	DBLP
HySGen	**0.200**	**0.441**	0.117	**0.117**	**0.322**	0.064
BigCLAM	0.168	0.372	0.063	0.095	0.256	0.034
CoDA	0.163	0.396	0.035	0.091	0.284	0.021
MotifCluster	0.161	0.303	0.093	0.096	0.209	0.054
Node2vec	0.172	0.351	**0.124**	0.097	0.237	**0.068**
Link Clust.	0.073	0.250	0.004	0.046	0.157	0.002

The nodes' colors in fig. 2b show the results of running HySGen on this hypergraph, with a 0.1 cutoff on the S values. The regular graph representation of the hypergraph is shown in fig. 2c, where the edges are shown with straight lines connecting pairs of nodes. We can see that using hyperedges allows HySGen to discover communities that are very close to the ground-truth. However, when the hypergraph is converted to a graph, the two big hyperedges overshadow all the other connections, causing graph-based methods to infer incorrect communities.

6.3 Real World Data Experiments

6.3.1 Dataset Description

We extracted three hypergraphs from real-world datasets for our experiments. The pre-processing and hypergraph extraction procedures for NSF and Scratch follow the same steps as is done by Revelle et al. (2015). DBLP is extracted from a dataset provided by Kamiński et al. (2019). For each of these hypergraphs, we took the largest connected component to be used in our experiments and filtered out the communities with fewer than three members. The statistics of the

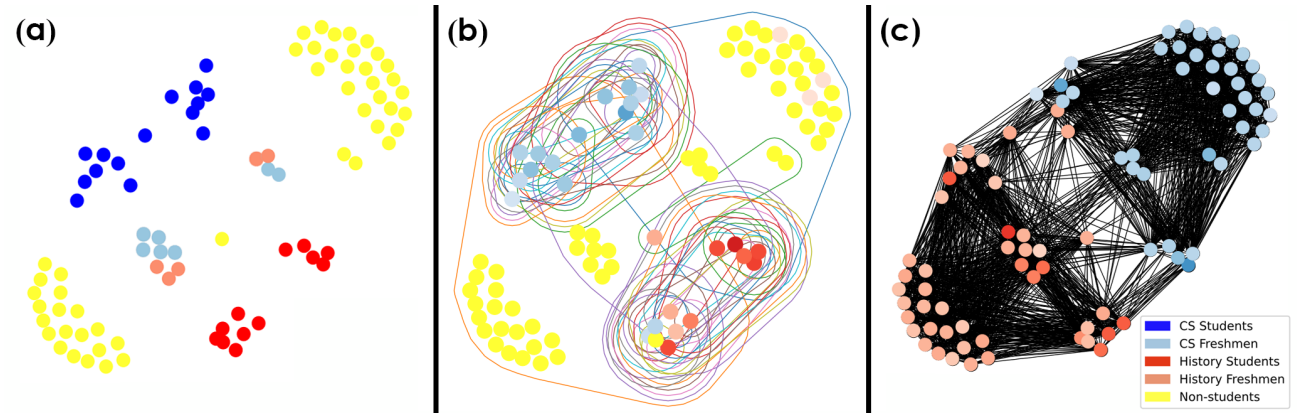

Figure 2: Synthetic example: A hypergraph where nodes are individuals and hyperedges show the recorded gatherings. The nodes are copied in the same location in each sub-figure. a) The ground truth for the hypergraph: Two main communities of CS students (red nodes) and history students (blue nodes), while yellow nodes belong to no community. b) The hypergraph's hyperedges are shown as contours that are colored for better visibility, and the discovered communities as colored nodes with darker blue and red colors show stronger affiliation. The yellow nodes have affiliation less than a threshold to any community. c) The edges of the graph equivalent of the hypergraph are shown as black lines, and the communities inferred form the graph are colored the same way as it is done in (b).

largest connected component of each real-world data is provided in table 2. A summary of each dataset is provided in appendix D.

6.3.2 Comparison Results

We evaluate HySGen against a range of baseline methods by comparing the performance of each method on the real-world datasets. All these methods run on the graph equivalent of the data. BigCLAM (Yang and Leskovec, 2013) is the closest graph algorithm to HySGen and finds the overlapping communities by fitting a generative model to the graph. CoDA (Yang et al., 2014) finds communities that are characterized either by dense inner-connections, or through a two-step distance. Link Clustering (Ahn et al., 2010) creates a dendrogram based on edge similarities for a hierarchical clustering scheme. We set the threshold on the dendrogram so that the number of discovered communities matches the number of the ground truth communities. Motifcluster (Benson et al., 2016) creates a motif adjacency matrix based on the higher-order connection patterns in the graph, and uses spectral clustering to provide a node embedding for the graph. The communities are then found by applying k-means clustering on the nodes' features. Node2vec (Grover and Leskovec, 2016) also provides a node embedding, but it leverages a neural network approach to capture the connection patterns. As in MotifCluster, Node2vec finds the communities by applying k-means on the node features.

Table 3 shows the results of applying the methods to the real-world datasets. The numbers in the table are computed by using (16) to compare each method against the ground truth communities. To have a more solid comparison, we produced two performance measures based on F1-score and Jaccard similarity. The results show that HySGen outperforms the baselines in both measures for the NSF and Scratch datasets. In DBLP, Node2vec has a narrow lead over our method, but HySGen still outperforms the other baselines with a noteworthy margin. Compared to NSF and Scratch, DBLP has a comparable number of nodes, but it has significantly fewer hyperedges and ground truth communities. One may argue that the high-dimensional embedding of Node2vec is able to better capture the relatively little amount of connection information provided in this data. HySGen can still outperform Node2vec when the hypergraph is not too sparse. Confirming this conjecture requires further investigation and experiments.

7 CONCLUSION

We proposed Hypergraph Simultaneous Generators (HySGen), a probabilistic model for discovering overlapping communities in hypergraphs. We showed that our proposed inference method can accurately detect communities by leveraging the higher order connections of the hyperedges. Thanks to the proposed computation reduction scheme and our efficient implementation, the method is relatively fast and scalable to

large hypergraphs with tens of thousands of hyperedges. Nevertheless, the inference can be slow when the hypergraph has millions of hyperedges and tens of thousands of communities are to be discovered. Since most of the computations in each iteration go for the line search in practice, this problem can be significantly improved by developing a parallel processing implementation of the updating algorithm for the community membership values of each node independently and then using those values for a fast sequential updating of ψ_n via (15). HySGen is the first method of its kind that is based on affiliation networks, and can be used as a basis to develop many other methods by changing the generative model's assumptions and structure. It is developed for simple, unweighted, and undirected hypergraphs, and can be extended to be applied to other types of hypergraphs.

References

E. Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie. Beyond pairwise clustering. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 838–845. IEEE, 2005.

S. Agarwal, K. Branson, and S. Belongie. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, pages 17–24. ACM, 2006.

Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761–764, 2010.

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

L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

G. Boros and V. Moll. *Irresistible integrals: symbolics, analysis and experiments in the evaluation of integrals*, page 14. Cambridge University Press, 2004.

S. R. Bulò and M. Pelillo. A game-theoretic approach to hypergraph clustering. In *Advances in neural information processing systems*, pages 1571–1579, 2009.

P. Chodrow and A. Mellor. Annotated hypergraphs: Models and applications. *Applied Network Science*, 5(1):9, 2020.

P. S. Chodrow. Configuration models of random hypergraphs and their applications. *arXiv preprint arXiv:1902.09302*, 2019.

P. S. Chodrow, N. Veldt, and A. R. Benson. Generative hypergraph clustering: from blockmodels to modularity. *arXiv preprint arXiv:2101.09611*, 2021.

S. Fortunato and D. Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.

D. Ghoshdastidar and A. Dukkipati. Consistency of spectral partitioning of uniform hypergraphs under planted partition model. In *Advances in Neural Information Processing Systems*, pages 397–405, 2014.

D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 597–605, 2012.

A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9):1074–1085, 1992.

D. Heckerman. Causal independence for knowledge acquisition and inference. In *Uncertainty in Artificial Intelligence*, pages 122–127. Elsevier, 1993.

M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram. The total variation on hypergraphs-learning on hypergraphs revisited. In *Advances in Neural Information Processing Systems*, pages 2427–2435, 2013.

J. Huang, X. Liu, and Y. Song. Hyper-path-based representation learning for hyper-networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 449–458, 2019.

D. Jin, Z. Yu, P. Jiao, S. Pan, P. S. Yu, and W. Zhang. A survey of community detection approaches: From statistical modeling to deep learning. *arXiv preprint arXiv:2101.01669*, 2021.

B. Kamiński, V. Poulin, P. Prałat, P. Szufel, and F. Théberge. Clustering via hypergraph modularity. *PloS one*, 14(11), 2019.

Z. T. Ke, F. Shi, and D. Xia. Community detection for hypergraph networks via regularized tensor power iteration. *arXiv preprint arXiv:1909.06503*, 2019.

C. Kim, A. S. Bandeira, and M. X. Goemans. Community detection in hypergraphs, spiked tensor models, and sum-of-squares. *arXiv preprint arXiv:1705.02973*, 2017.

C. Kim, A. S. Bandeira, and M. X. Goemans. Stochastic block model for hypergraphs: Statistical limits and a semidefinite programming approach. *arXiv preprint arXiv:1807.02884*, 2018.

S. Lattanzi and D. Sivakumar. Affiliation networks. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 427–434, 2009.

M. Leordeanu and C. Sminchisescu. Efficient hypergraph clustering. In *International Conference on Artificial Intelligence and Statistics*, pages 676–684, 2012.

J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

P. Li and O. Milenkovic. Inhomogeneous hypergraph clustering with applications. In *Advances in Neural Information Processing Systems*, pages 2308–2318, 2017.

M. Revelle, C. Domeniconi, M. Sweeney, and A. Johri. Finding community topics and membership in graphs. In *Machine Learning and Knowledge Discovery in Databases*, pages 625–640. Springer, 2015.

N. Veldt, A. Wirth, and D. F. Gleich. Parameterized objectives and algorithms for clustering bipartite graphs and hypergraphs. *arXiv preprint arXiv:2002.09460*, 2020.

D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux. Revisiting user mobility and social relationships in lbsns: A hypergraph embedding approach. In *The World Wide Web Conference*, pages 2147–2157, 2019.

J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.

J. Yang and J. Leskovec. Structure and overlaps of ground-truth communities in networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(2):26, 2014.

J. Yang, J. McAuley, and J. Leskovec. Community detection in networks with node attributes. In *Data Mining (ICDM), 2013 IEEE 13th international conference on*, pages 1151–1156. IEEE, 2013.

J. Yang, J. McAuley, and J. Leskovec. Detecting cohesive and 2-mode communities indirected and undirected networks. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 323–332, 2014.

D. Zhou, J. Huang, and B. Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in neural information processing systems*, pages 1601–1608, 2007.

Supplementary Material: Hypergraph Simultaneous Generators

A PARTIAL DERIVATIVES COMPUTATION REDUCTION FORMULATION

As discussed in section 5, to have a computationally tractable inference of the communities, we need to develop an efficient way to compute the summations on $\bar{e} \in \bar{E}$ in both (5) and (6). In section 5.1, we developed a mathematical transformation for the summation on $\bar{e} \in \bar{E}$ in (5) to make it accessible as a function of $\psi_n(\Upsilon'_c, c)$ in $O(1)$. The formulations were completed by developing an $O(1)$ procedure to compute updated values of each $\psi_n(\Upsilon'_c, c)$ in section 5.2. In this section, we follow similar steps to the ones in section 5.1 toward developing an $O(1)$ procedure for computing the summation on $\bar{e} \in \bar{E}$ in (6). We start with rewriting (6) as:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathcal{H}}(S_{vc})}{\partial S_{vc}} = \\ \sum_{\substack{\{e \in E \\ |v \in e\}}} \frac{\prod\limits_{m \in (e-\{v\})} S_{mc} \prod_{b \in ([C]-\{c\})} \left(1 - \prod_{u \in e} S_{ub}\right)}{1 - \prod_{b=1}^{C} \left(1 - \prod_{u \in e} S_{ub}\right)} \\ - \Psi^{(\partial)}(\substack{\{\bar{e} \in \bar{E} \\ |v \in \bar{e}\}}, c) - \lambda_S I_1(S_{vc})\end{aligned} \tag{A.1}$$

where $\Psi^{(\partial)}(A, c)$ is the partial derivative equivalent of (8) and defined as given below:

$$\Psi^{(\partial)}(A, c) = \frac{1}{S_{vc}} \sum_{\hat{\varepsilon} \in A} \frac{\prod_{u \in \hat{\varepsilon}} S_{uc}}{1 - \prod_{u \in \hat{\varepsilon}} S_{uc}} \tag{A.2}$$

Taking the same considerations for $\prod_{u \in \varepsilon} S_{vc}$ as we did in section 5.1 for (10), a Taylor expansion representation for $\Psi^{(\partial)}(A, c)$ is yielded as:

$$\begin{aligned}\Psi^{(\partial)}(A, c) =& \frac{1}{S_{vc}} \sum_{\varepsilon \in A} \sum_{n=1}^{n_\infty} \left(\prod_{u \in \varepsilon} S_{uc}\right)^n \\ =& \frac{-1}{S_{vc}} \sum_{n=1}^{n_\infty} n \times \psi_n(A, c)\end{aligned} \tag{A.3}$$

where $\psi_n(A, c)$ is the same function that is used in (10).

It is trivial to show that $\Psi^{(\partial)}(\substack{\{\bar{e} \in \bar{E} \\ |v \in \bar{e}\}}, c) = \Psi^{(\partial)}(\Upsilon_{vc}, c) - \Psi^{(\partial)}(\substack{\{e \in E \\ |v \in e\}}, c)$. Analogous to the discussion in section 5.1, the key to reducing the time complexity of computing $\Psi^{(\partial)}(\substack{\{\bar{e} \in \bar{E} \\ |v \in \bar{e}\}}, c)$ is to compute $\Psi^{(\partial)}(\Upsilon_{vc}, c)$. As (A.3) indicates, the latter can be computed in $O(1)$ by applying (13) to the same $\Psi^{(\partial)}(\Upsilon'_c, c)$ values that are initialized and updated in section 5.2. This means that, for an efficient computation of the gradient vectors, we only need what we have already computed, and no additional initialization and updating steps are necessary. Note that for (A.3) to work, we should have $S_{vc}^{old} > 0$. However, if $S_{vc}^{old} = 0$, its value can be computed by only taking the $n = 1^{st}$ term of the Taylor expansion as $\Psi^{(\partial)}(S, c) = -\psi_1(\Upsilon'_c, c)$.

B COMPLEXITY ANALYSIS

Our inference algorithm (sections 4 and 5) iteratively updates the community membership values until reaching a local maximum for (4). At each iteration, we should update the S_{vc} values for all N nodes and C communities. To update each S_{vc}, since we are using the method of coordinate ascent, we should compute the partial derivatives

in (6); and because we use Armijo-Goldstein's backtracking line search (Armijo, 1966) for finding the step sizes, we should also compute (5). We proceed the analysis with (6) as it is the computationally dominant formula and has more operations for each output S_{vc} component. Without the computation reduction platform in section 5, computing (6) includes 2^{N-1} summation terms for each potential hyperedge adjacent to v. Each term in the summation corresponding to the hyperedges includes a product of C sub-terms, and each sub-term includes multiplying as many items as the size of the hyperedge. Considering the closed-form evaluation of the sum of the sizes of all possible hyperedges adjacent to v (Boros and Moll, 2004), computing (6) takes $O(CN2^{N-2})$ operations. Since this should be computed for all the nodes and communities, a complete inference iteration is done in $O(C^2N^2 2^{N-2})$.

By introducing the computation reduction scheme in sections 5.1 and 5.2, we still have to update $C \times N$ values of S_{vc} at each iteration. But the change in the complexity order comes from reducing the number of computations for updating each S_{vc}. This reduction is two-fold, one for the summations over $\{e \in E\}$ and the other for the summations over $\{\bar{e} \in \bar{E}\}$ in (4) and (6).

For the $\{e \in E\}$ summation, instead of directly computing all parts of a term, we follow an initialization+update scheme for the following subformulas: 1) $\prod_{u \in e} S_{uc}$ for M hyperedges and C communities, 2) $\prod_{c=0}^{C} \left(1 - \prod_{u \in e} S_{uc}\right)$ for M hyperedges, and 3) $\sum_{c=0}^{C} \log\left(1 - \prod_{u \in \bar{e}} S_{uc}\right)$ for M hyperedges. Initializing these subformulas with a fixed S_{vc} value for the initially assigned members of the communities takes a total of $O(C \times M)$ calculations. Then for every new $S_{vc}^{(t)}$ value computed during a parameter inference iteration, the saved values for the above subformulas that include S_{vc} should be updated. It is trivial to show that updating each one of them can be done in $O(1)$. In a complete iteration, the terms corresponding to all hyperedges should be updated, and for each hyperedge, the corresponding term is updated as many times as the size of the hyperedge. Consequently, the overall complexity of updating these terms in an iteration is $O(C \times vol_{\mathcal{H}})$.

For the $\{\bar{e} \in \bar{E}\}$ summation, the analytical initialization solution in (14) and updating formula in (15) for $\psi_n(\Upsilon'_c, c)$ are both done in $O(1)$. For every newly computed $S_{vc}^{(t)}$, n_∞ number of $\psi_n(\Upsilon'_c, c)$ values should be updated. However, the terms of the Taylor expansion approach zero after few terms, and setting n_∞ to a small constant at the beginning of the inference gives sufficient precision. In our experiments we set this parameter to 30 and the 30^{th} term has never exceeded 10^{-5} (most of the times it goes to zero in much less terms). This means the total computational cost of updating the $\psi_n(\Upsilon'_c, c)$ values for all $n = 1, \ldots, n_\infty$ is $O(1)$. We should also emphasize that while we set $n_\infty = 30$ in our implementation, setting a larger value 1) does not have any affect on the output for the hypergraphs used in our experiments, and 2) has virtually no effect on the inference speed, as the algorithm stops going to $n + 1$ after $\psi_n(\Upsilon'_c, c)$ becomes less than a threshold (which is set to 10^{-5} in our implementation. As explained in section 5 and appendix A, all the intermediate steps from (13) up to computing $\Psi(\binom{\{\bar{e} \in \bar{E}}{|v \in \bar{e}\}}, c)$ and $\Psi^{(\partial)}(\binom{\{\bar{e} \in \bar{E}}{|v \in \bar{e}\}}, c)$ also take $O(1)$ operations. So the total computational cost for the $\{\bar{e} \in \bar{E}\}$ summations in (4) and (6) is $O(C \times N)$, which is dominated by the cost of $\{e \in E\}$ summations. This means the total computational cost of one updating iteration in our inference algorithm is reduced to $O(C \times vol_{\mathcal{H}})$.

C OPTIMIZATION AND PARAMETER ASSIGNMENTS

Our inference method uses Block Coordinate Ascent to update the community membership values S as explained in section 4. In each round of the updating iterations, the nodes are shuffled in a random order, then each node v is chosen one at a time to update the vector $S_{v\cdot}$. At each iteration, our inference follows a *stochastic* gradient ascent scheme: after calculating the new value for v, the change is applied on S and the stored term mentioned in section 5.2 and appendix B are updated, for the next nodes in the agenda. To update each $S_{v\cdot}$ in (7), we use the Backtracking Line Search method to find the α coefficients. The solution space for S is constrained and each $0 \leq S_{vc} \leq 1$. It can be shown that the Karush-Kuhn-Tucker (KKT) conditions satisfy the subproblem of (5) for all its associated inequality constraints. We incorporated the second KKT condition (Complimentary Slackness) in our updating iterations and allow the components of $S_{v\cdot}$ to adjust their update in (7) according to their binding or non-binding standing. Specifically, we start the search direction as $p = \nabla \mathcal{L}_{\mathcal{H}}(S_{v\cdot})$ for every α that has to be tried during the line search. Then if $S_{vc} + \alpha p_c$ exceeds an inequality constraint for some component c, we set p_c so that $S_{vc} + \alpha p_c$ becomes binding to that constraint. After a satisfying p_c is chosen, we change its corresponding component in p to the new value and use the vector p instead of the original $\nabla \mathcal{L}_{\mathcal{H}}(S_{v\cdot})$ in (7), to update $S_{v\cdot}$.

The value we set for the null community parameter reflects our judgement about the noise level of the hyperedges. We set this parameter to $S_0 = \frac{1}{N}$ for all the datsets. The L_1 regularization term λ, along with many other parameters, are set as inputs to our method, and the values are adjusted over many trials to yield the highest log-posterior value at the end of the iterations. The reported values in table 3 for our method are set by running it ten times with the same input parameters and yielding the results with the highest log-posterior. Since the performance measure that we used does not acknowledge the soft membership weights inferred by HySGen, we put a cutoff value of 0.1 for our computed membership values for the NSF and Scratch datasets, and 0.0 for DBLP.

D REAL-WORLD DATASETS

- **NSF:** This dataset is gathered from NSF awards conceded between 1995 and 2014 by the NSF's Directorate for Computer and Information Science and Engineering (CISE). The award archive is accessible on the NSF website[2]. A collaboration hypergraph was extracted from the data by taking the researchers as nodes, and each awarded proposal as a hyperedge to connect the researchers who collaborated in writing the proposal. The researchers are associated with programs, so we took the programs with at least three researchers as the ground truth communities.

- **Scratch:** It is an online community for software developers to share code and collaborate on projects. An archival dataset of five years of public activity in this community is accessible through Harvard Dataverse repository[3]. In this environment, users can *remix* a project by copying and modifying it. We created a co-remix hypergraph from the interactions recorded in the year 2009. We assigned the users to nodes and their co-remixing projects to hyperedges. Project galleries are collections of selected projects created by a user who moderates the requests for adding new projects to the gallery. We used galleries corresponding to three or more users as ground truth communities.

- **DBLP:** This dataset is introduced by Kamiński et al. (2019) and is accessible at the authors' GitHub repository[4]. The actual hypergraph statistics extracted from the data do not match the description provided in the repository description and its corresponding paper. The dataset provides a fully connected citation hypergraph where the nodes are authors and hyperedges are papers. The 14 ground truth communities *partition* the nodes into non-overlapping groups, where the smallest has five members, and the largest has 2467 members.

[2]https://www.nsf.gov/awardsearch/download.jsp
[3]https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/KFT8EZ
[4]https://gist.github.com/pszufe/02666497d2c138d1b2de5b7f67784d2b