Construction and Random Generation of Hypergraphs with Prescribed Degree and Dimension Sequences

Naheed Anjum Arafat[1](✉), Debabrota Basu[2], Laurent Decreusefond[3], and Stéphane Bressan[1]

[1] School of Computing, National University of Singapore, Singapore, Singapore
`naheed_anjum@u.nus.edu`

[2] Data Science and AI Division, Chalmers University of Technology, Gothenburg, Sweden

[3] LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Abstract. We propose algorithms for construction and random generation of hypergraphs without loops and with prescribed degree and dimension sequences. The objective is to provide a starting point for as well as an alternative to Markov chain Monte Carlo approaches. Our algorithms leverage the transposition of properties and algorithms devised for matrices constituted of zeros and ones with prescribed row- and column-sums to hypergraphs. The construction algorithm extends the applicability of Markov chain Monte Carlo approaches when the initial hypergraph is not provided. The random generation algorithm allows the development of a self-normalised importance sampling estimator for hypergraph properties such as the average clustering coefficient.

We prove the correctness of the proposed algorithms. We also prove that the random generation algorithm generates any hypergraph following the prescribed degree and dimension sequences with a non-zero probability. We empirically and comparatively evaluate the effectiveness and efficiency of the random generation algorithm. Experiments show that the random generation algorithm provides stable and accurate estimates of average clustering coefficient, and also demonstrates a better effective sample size in comparison with the Markov chain Monte Carlo approaches.

1 Introduction

While graphs are the prevalent mathematical models for modern applications, being natural representations of varied objects such as transportation, communication, social and biological networks [19], to mention a few, they only capture binary relationships. Hypergraphs introduce the opportunity to represent n-ary relationships and thus create a more general, albeit more complex and generally more computationally expensive, alternative [4,15,22].

Indeed many real-world systems are more naturally modelled as hypergraphs, as exemplified by the cases of multi-body systems, co-authorship networks and

© Springer Nature Switzerland AG 2020
S. Hartmann et al. (Eds.): DEXA 2020, LNCS 12392, pp. 130–145, 2020.
https://doi.org/10.1007/978-3-030-59051-2_9

parliamentary relations [3,21,22]. While the applications are numerous, the properties of the underlying hypergraphs are yet to be fully understood. Just as it is the case for graphs in network science [5,12,18], configuration modelling or the random generation of hypergraphs with prescribed degree and dimension sequences precisely allows the fabrication of suitable hypergraphs for empirical and simulation-based studies of hypergraph properties [8].

We study and propose algorithms for construction and random generation of hypergraphs with prescribed degree and dimension sequences (Sect. 4, 5). In addition, we present the necessary background on hypergraphs and $(0, 1)$-matrices in Sect. 2 and synthesise related works in Sect. 3.

Recently, Chodrow [8] proposed a Markov chain Monte Carlo (MCMC) algorithm to address this problem of generating a labelled hypergraph with a prescribed degree and dimension sequences. The limitation of the MCMC algorithm is that it requires an initial hypergraph with the prescribed degree and dimension sequences as a starting point. It is not always the case that such an initial hypergraph is available. Therefore, we present in Sect. 4 a deterministic algorithm for constructing an initial hypergraph as a starting point for the existing MCMC approach. At each iteration, our algorithm constructs the edge with the largest dimension using distinct vertices having the largest degrees.

We present in Sect. 5 a random generation algorithm for generating hypergraphs as an alternative to the existing MCMC approach. Our generation algorithm leverage properties and methods devised for $(0, 1)$-matrices [20] with row- and column-sums coinciding with the hypergraph specification. If no row or column in the matrix contains only zeros, every $(0, 1)$-matrix corresponds to the incidence matrix of a hypergraph with parallel-edges but no loop [4, Chap. 17]. The column-sums of an incidence matrix represent degrees of the vertices and the row-sums represent dimensions of the edges of a hypergraph. At each iteration, the random generation algorithm generates the edge with the largest dimension using distinct, randomly selected vertices that satisfy the characterisation theorem for $(0, 1)$ matrices (Theorem 1).

We further leverage our random generation algorithm to propose a self-normalised importance sampling (SNIS) estimator [16] for estimating hypergraph properties in Sect. 6.

We prove the correctness of both the algorithms (Theorems 2 and 3). Furthermore, we prove that the generation algorithm generates any random hypergraph having prescribed degree and dimension sequences with non-zero probability (Theorem 4). We evaluate the effectiveness (Sect. 7) of the MCMC algorithm enabled with our construction algorithm and the random generation algorithm with SNIS estimator by estimating the average clustering coefficient of the projected graphs of the family of hypergraphs having prescribed degree and dimension sequence and also computing corresponding effective samples size [16].

We conclude in Sect. 8 by summarising our findings.

2 Hypergraphs and $(0, 1)$-matrices

In this section, we describe selected concepts of hypergraphs and inequalities involving $(0, 1)$-matrices relevant for the transposition of properties and algorithms for $(0, 1)$-matrices to hypergraphs.

Definition 1 (Hypergraph [4]). *A hypergraph $H = (V, E)$ is a tuple of a vertex set $V = \{v_1, \ldots, v_n\}$ and an edge set $E = \{e_1, \ldots, e_m\}$ where each edge is a subsets of V. Here V is a set and E is a multiset.*

Unless otherwise stated, hypergraphs are *labelled*, may contain *parallel-edges* but *no self-loop*. The polyadic relations i.e. the edges connecting the vertices in a hypergraph is presentable as a $(0, 1)$-matrix, called the incidence matrix.

Definition 2 (Incidence Matrix). *The incidence matrix $M = [m_{ij}]_{i,j=1,1}^{m,n}$ of a labelled hypergraph $H = (V, E)$ is a $(0, 1)$-matrix with columns representing labels of vertices in V and rows representing edge set E where*

$$m_{i,j} = \begin{cases} 1 & \textit{if } v_j \in e_i, \\ 0 & \textit{otherwise.} \end{cases}$$

The incidence matrix of a hypergraph is not unique. Even if the vertices are arranged in a total-order, such as in descending order of degrees and lexicographic order of labels among the vertices with same degree, any permutation of the rows would provide another incidence matrix of the same hypergraph.

Property 1. Every incidence matrix of a hypergraph whose degree sequence is $(a)_n$ and dimension sequence is $(b)_m$ is contained in the set of $(0, 1)$-matrices of dimension $m \times n$ whose column-sums are $(a)_n$ and row-sums are $(b)_m$. *Thus, any algorithm that uses the characterisation of sequences $(a)_n$ and $(b)_m$ to construct an $m \times n$-dimensional $(0, 1)$-matrix with column-sum $(a)_n$ and row-sums $(b)_m$ can be leveraged to construct a hypergraph with degree-sequence $(a)_n$ and dimension sequence $(b)_m$.* This observation constitute the core of our random hypergraph generation proposed in Sect. 5.

Property 2. In order to design the proposed algorithms and to prove their correctness, we use the Gale-Rysers characterisation of $(0, 1)$-matrices (Theorem 1). Before discussing the theorem, we intend to remind us the notion of dominance between sequences.

Definition 3 (Dominance [17]). $(a)_n$ is defined to be dominated by $(b)_m$ *if the corresponding zero-padded sequences $(a^*)_l$ and $(b^*)_l$, where $l = \max\{m, n\}$ satisfy -*
(i) *sum of the first k components of $(a^*)_l$ is smaller than or equal to sum of the first k components of $(b^*)_l$ and*

(ii) *sum of all the components of* $(a^*)_l$ *is equal to the sum of all the components of* $(b^*)_l$. *Mathematically,*

$$(a)_n \prec (b)_m \Longleftrightarrow \begin{cases} \sum_{i=1}^{k} a_i^* \leq \sum_{i=1}^{k} b_i^*, & k = 1, 2, \ldots, l-1 \\ \sum_{i=1}^{l} a_i^* = \sum_{i=1}^{l} b_i^*. \end{cases}$$

$a_i^* = a_i$ *for* $i \leq n$, $a_i^* = 0$ *for* $i > n$, $b_i^* = b_i$ *for* $i \leq m$ *and* $b_i^* = 0$ *for* $i > m$.

Theorem 1 (Gale-Rysers [14,20] **Characterisation of Matrices).** *If* $(a)_n = (a_1, a_2, \ldots, a_n)$ *and* $(b)_m = (b_1, b_2, \ldots, b_m)$ *are two monotonically non-increasing, non-negative integer sequences, the necessary and sufficient condition for the existence of a* $(0,1)$*-matrix with column sums* $(a)_n$ *and row sums* $(b)_m$ *is that* $(a)_n$ *is dominated by the conjugate sequence of* $(b)_m$.

The conjugate sequence of $(b)_m$ is a sequence whose i^{th} component is the number of components in $(b)_m$ that are greater than or equal to i. We denote the conjugate sequence of $(b)_m$ as $\bar{b}_n$.[1] A sequence-pair $((a)_n, (b)_m)$ satisfying the dominance condition in Gale-Rysers characterisation is said to be *realisable by a* $(0,1)$*-matrix.* Conversely, such a matrix is said to *realise* $(a)_n, (b)_m$.

Property 3. Another observation is that if we construct a new sequence $(a')_{n-1} = (a_2, a_3, \ldots, a_n)$ from a monontonically non-increasing positive integer sequence $(a)_n = (a_1, a_2, a_3, \ldots, a_n)$, the conjugate sequence of $(a)_{n-1}$ can be derived from the conjugate sequence of $(a)_n$ by reducing the first a_1 components of $\bar{a}$ by 1.

Lemma 1 ([17]**).** *Let* $(a)_n = (a_1, a_2, \ldots, a_n)$ *be a positive monotonically non-increasing. If we construct a new sequence* $(a')_{n-1} \triangleq (a_2, \ldots, a_n)$, *then the conjugate sequence of* $(a')_{n-1}$ *is*

$$(\bar{a'}) = (\bar{a}_1 - 1, \ldots, \bar{a}_{a_1} - 1, \bar{a}_{a_1+1}, \ldots, \bar{a}_n).$$

Example 1. Let $(a) = (4, 2, 2, 1)$. Its conjugate sequence is $(\bar{a}) = (4, 3, 1, 1, 0, \ldots)$. By removing a_1, we get a new sequence $(a') = (2, 2, 1)$. The conjugate sequence of (a') is $(3, 2, 0, 0, \ldots)$ which is exactly the sequence derived from $(4, 3, 1, 1, 0, \ldots)$ by reducing first four components by 1 i.e. $(4-1, 3-1, 1-1, 1-1, 0, \ldots)$.

Fulkerson and Ryser [13] state a necessary condition that preserves dominace after reducing the values of a fixed number of components by 1 in sequences $(a)_n$ and $(b)_n$ related by dominance.

[1] Since the number of 1's in a row of an $m \times n$-dimensional $(0,1)$ matrix cannot exceed n, the length of the conjugate sequence of row sums $(b)_m$ is upper bounded by n.

Lemma 2 (Fulkerson-Ryser's Lemma [13]). *Let $(a)_n$ and $(b)_n$ be two monotonically non-increasing integer sequences. Let $(u)_n$ be sequence obtained from $(a)_n$ by reducing components at indices $i_1, i_2, \ldots, i_K$ by 1. Similarly, let $(v)_n$ be obtained from (b) by reducing components at indices $j_1, j_2, \ldots, j_K$ by 1.*

If $i_1 \leq j_1, i_2 \leq j_2, \ldots, i_K \leq j_K$, and $(a)_n \prec (b)_n$, we get $(u)_n \prec (v)_n$.

We leverage this lemma to prove correctness of our construction algorithm (Sect. 5).

3 Related Works

3.1 Graphs with a Prescribed Degree Sequence

There are two main frameworks for the generation of random graphs with a prescribed degree sequence. The first framework is *direct sampling* [12], that constructs the graph incrementally edge-by-edge. Among algorithms based on direct sampling, [6] and [2] introduced the concept of *stubs* and the procedure of stub-matching as an algorithm for counting the number of labelled graphs with a prescribed degree sequence. The stub-matching procedure may generate graphs with loops and parallel-edges, which is often undesirable. Rejecting the generated random graph until a simple graph is generated is proposed as a remedy. However, this approach is inefficient for large degree values as an exponential number of samples might get rejected [5,12]. Furthermore, there is no obvious way to extend this algorithm for graphs into an algorithm for hypergraphs [8].

As an alternative to the stub-matching algorithm, [5] proposed an algorithm that uses the Erdös-Gallai's characterisation to generate simple graphs. This algorithm generates all simple graphs following a given degree sequence with a non-zero probability. [5] also proposes an importance sampling scheme to estimate the number of simple graphs following the prescribed degree sequence. Motivated by their work on simple graphs, in this paper, we devise a self-normalised importance sampling scheme (Sect. 6) using our random generation algorithm (Sect. 5) to estimate average clustering coefficient of projected graphs of hypergraphs having a prescribed degree and dimension sequences (Sect. 7).

The second framework proposes *MCMC* algorithms [12,18,19] that iteratively switch edges of an initial graph with a given degree sequence to obtain the final graph. MCMC algorithms try and show that the intermediate hypergraphs form a Markov chain whose stationary distribution converges to the uniform distribution over the set of all graphs with the given degree sequence [12]. However, it is challenging to prove mixing-time bounds for algorithms in this family, and mixing results are known only for regular graphs [5].

3.2 Hypergraphs with Prescribed Degree and Dimension Sequences

Chodrow [8] proposed a hypergraph configuration modelling approach to the uniform distribution of labelled hypergraphs with prescribed degree and dimension sequence. The hypergraphs under investigation have parallel-edges but no

Algorithm 1. Constructing initial hypergraphs

```
Input: Degree and dimension sequences, $(a)_n$ and $(b)_m$, sorted in descending order
Output: Hypergraph $H = (V, E)$
1: Initialise: $V \leftarrow \{1, \dots, n\}$, $E \leftarrow \phi$, $(a)^1 \leftarrow (a)_n$, $(b)^1 \leftarrow (b)_m$
2: for $j = 1, 2, \cdots, m$ do
3:     Construct edge $e_j = \{v_1, \cdots, v_{b_1^j}\}$
4:     Construct $(a)_n^{j+1}$ by reducing the first $b_1^j$ components of $(a)_n^j$ by 1.
5:     Construct $(b)^{j+1} = (b_2^j, b_3^j, \dots, b_m^j)$
6:     $E \leftarrow E \cup \{e_j\}$
7:     Sort sequence $(a)^{j+1}$ in descending order.
8: end for
```

self-loop. He proposed an MCMC algorithm that, as it is done in similar algorithms for graphs, sequentially switches edges of a labelled initial hypergraph satisfying the prescribed degree and dimension sequences. As the lag at which to sample a hypergraph from the Markov chain tends to infinity, he showed that the algorithm outputs uniformly at random a hypergraph from the set of all hypergraphs having the prescribed degree and dimension sequences.

However, in practice, the initial hypergraph is not always available. Additionally, due to lack of mixing time results about the chain, there is no principled guideline for choosing lag. These observations motivated us to develop both a deterministic algorithm to construct an initial hypergraph facilitating the MCMC algorithm, as well as a random generation algorithm that does not need an initial hypergraph as an alternative to the MCMC algorithm.

4 Construction of an Initial Hypergraph

We leverage the properties elaborated in Sect. 2 to construct a hypergraph with prescribed degree and dimension sequences. The construction algorithm addresses the limitation of the MCMC algorithm [8] by providing a starting point for it. Our algorithm uses the methodology proposed by Ryser [20] for $(0,1)$-matrices and by Gale [14] for flows in bipartite-graphs. We illustrate the pseudocode in Algorithm 1. *At each iteration, Algorithm 1 constructs the edge with the largest dimension using distinct vertices having the largest degrees.*

In Algorithm 1, the aim is to construct a hypergraph with n vertices, m edges, degree sequence $(a)_n$, and dimension sequence $(b)_m$. Algorithm 1 takes non-increasingly sorted sequences $(a)_n$ and $(b)_m$ as input. It initialises $(a)^1$ as $(a)_n$ and $(b)^1$ as $(b)_m$. At each iteration $j \in \{1, \dots, m\}$, it constructs an edge by selecting b_1^j distinct vertices with maximal non-zero degrees[2]. Then it constructs $(a)^{j+1}$ by reducing the degrees of the selected vertices in $(a)^j$ by 1 and refers to $(b_2^j, \dots, b_m^j)$ as $(b)^{j+1}$. It proceeds to construct the next edge using $(a)^{j+1}$ and $(b)^{j+1}$, and continues until all m edges are constructed.

[2] Here the ties are broken using the lexicographic order of the vertex-labels.

We prove that the construction of edge e_j at every iteration j is feasible, meaning, the residual sequences $(a)^{j+1}$ and $(b)^{j+1}$ are realisable by a hypergraph.

Theorem 2. *If the sequences $(a)^j$ and $(b)^j$ are realisable by a hypergraph with m edges and n vertices, the sequences $(a)^{j+1}$ and $(b)^{j+1}$, constructed at iteration j, are realisable by a hypergraph with $(m-1)$ edges and n vertices.*

Proof Sketch. We prove the theorem by induction on m. If $m = 0$, the algorithm terminates with a hypergraph with empty edges ($E = \phi$), which is the only hypergraph with 0 edges and n vertices.

Suppose $m > 0$. By induction hypothesis, $(a)^j, (b)^j$ are realisable by a hypergraph H with m edges. Taking an incidence matrix M of H and applying Theorem 1, we get $(a)^j \prec (\bar{b})^j$. By construction, $(a)^{j+1}$ is the same as $(a)^j$ except the first b_1^j components are reduced by 1. By construction of $(b)^{j+1}$ and Lemma 1, the conjugate $(\bar{b})^{j+1}$ is the same as $(\bar{b})^j$ except the first b_1^j components reduced by 1. Thus Lemma 2 implies that $(a)^{j+1} \prec (\bar{b})^{j+1}$. By Theorem 1, an $(m-1) \times n$-dimensional incidence matrix M' of some hypergraph H' exists that realises sequences $(a)^{j+1}, (b)^{j+1}$.

5 Random Generation of Hypergraphs

In this section, we propose a random generation algorithm (Algorithm 2) using the characterisation (Theorem 1) for $(0,1)$-matrices. In Algorithm 2, we iteratively construct edges in descending order of cardinality and stochastically assign the vertices to the edges such that Theorem 1 is satisfied. Algorithm 2 leverages design methods proposed for $(0,1)$-matrices in [7].

Three observations are central to the development of Algorithm 2.

Observation 1. *If there are two sequences $(b)^j = (b_1^j, b_2^j, \ldots, b_m^j)$ and $(b)^{j+1} = (b_2^j, \ldots, b_m^j)$, Lemma 1 implies that we can construct the conjugate sequence of $(b)^{j+1}$, namely $(\bar{b})^{j+1}$, from the conjugate sequence of $(b)^j$, namely $(\bar{b})^j$, by reducing first b_1^j components of $(\bar{b})^j$ by 1.*

Observation 2. *If we randomly select K non-zero components from $(a)^j$ whose indices are $i_1, \ldots, i_K$ and reduce them by 1, we obtain a residual sequence $(a)^{j+1}$. If we select those K components in such a way that after reduction the dominance $(a)^{j+1} \prec (\bar{b})^{j+1}$ holds, we can construct an $(m-1) \times n$-dimensional $(0,1)$-matrix with residual column-sums $(a)^{j+1}$ and row-sums $(b)^{j+1}$. This is direct consequence of Gale-Rysers theorem (Theorem 1). The constructed $(0,1)$-matrix is an incidence matrix of a hypergraph with $m-1$ edges and n vertices having degree sequence $(a)^{j+1}$ and dimension sequence $(b)^{j+1}$.*

Observation 3. *Since our interest is to reduce K non-zero components of $(a)^j$ by 1 while preserving the dominance $(a)^{j+1} \prec (\bar{b})^{j+1}$, we search for the indices*

Algorithm 2. Generating random hypergraphs

```
Input: Degree and dimension sequences, $(a)_n$ and $(b)_m$, sorted in descending order
Output: Hypergraph $H = (V, E)$
 1: Initialise: $V \leftarrow \{1, \ldots, n\}$, $E \leftarrow \phi$, $(a)^1 \leftarrow (a)_n$, $(b)^1 \leftarrow (b)_m$.
 2: $(\bar{b})^1 \leftarrow$ conjugate sequence of $(b)^1$
 3: for $j = 1, 2, \cdots, m$ do
 4:     Construct $(\bar{b})^{j+1}$ from $(\bar{b})^j$ by reducing first $b_1^j$ components in $(\bar{b})^j$ by 1.
 5:     Compute critical indices $\{k_1^j, k_2^j, \ldots\}$ where $(a)^j \not\prec (\bar{b})^{j+1}$ (Equation (1)).
 6:     Compute corresponding margins of violation $\{n_1^j, n_2^j, \ldots\}$ (Equation (2)).
 7:     $e_j \leftarrow \phi$, $k_0^j \leftarrow 0$
 8:     while $k_i^j \in \{k_1^j, k_2^j, \ldots\}$ do
 9:         $o_i \rightarrow$ An integer sampled from $[n_i^j - |e_j|, \min(b_1^j - |e_j|, k_i^j - k_{i-1}^j)]$ uniformly
            at random.
10:         $O_i \rightarrow o_i$ indices selected from $\mathcal{I}_i^j = [k_{i-1}^j + 1, k_i^j]$ uniformly at random.
11:         $e_j \leftarrow e_j \cup O_i$
12:         Reduce components in $(a)^j$ at positions $O$ by 1.
13:     end while
14:     $E \leftarrow E \cup \{e_j\}$.
15:     $(a)^{j+1} \leftarrow (a)^j$ sorted in descending order.
16:     Construct $(b)^{j+1} = (b_2^j, b_3^j, \ldots, b_m^j)$.
17: end for
```

*in $(a)^j$ where the violation of dominance $(a)^j \not\prec (\bar{b})^{j+1}$ occur. We label an index $1 \leq k < n$ to be **critical** if*

$$\sum_{i=1}^{k} a_i^j > \sum_{i=1}^{k} \bar{b}_i^{j+1}. \tag{1}$$

k being a critical index implies that in order to preserve dominance $(a)^{j+1} \prec (\bar{b}^{j+1})$ within integer interval $[1, k]$, we need to reduce at least

$$n \triangleq \sum_{i=1}^{k} a_i^j - \sum_{i=1}^{k} \bar{b}_i^{j+1} \tag{2}$$

*number of 1's at or before index k in $(a)^j$. We say n is the **margin of violation** corresponding to the critical index k. At every iteration, we enlist all the critical indices and their corresponding margins of violation.*

Algorithm 2 takes the degree and dimension sequences, $(a)_n$ and $(b)_m$ respectively, sorted in descending order as input. We refer to them as $(a)^1 = (a)_n$ and $(b)^1 = (b)_m$ (Line 1). Following that, it constructs the conjugate $(\bar{b})^1$ of the initial dimension sequence $(b)^1$ (Line 2).

At each iteration $j \in \{1, \ldots, m\}$, the algorithm constructs a conjugate sequence for dimensions of $(m - j)$ edges, namely $(\bar{b})^{j+1}$, from the conjugate sequence for dimensions of $(m - j + 1)$ edges, namely $\bar{b}^j$, by reducing the first b_1^j components in $(\bar{b})^j$ by 1 (Line 4). This is a consequence of Observation 1.

Following Observation 3, Algorithm 2 uses $(a)^j$ and $(\bar{b})^{j+1}$ to compute all the critical indices $\{k_1^j, k_2^j, \ldots\}$ (Line 5) and their corresponding margins of violations $\{n_1^j, n_2^j, \ldots\}$ (Line 6). The critical indices partition $\{1, \ldots, n\}$ in integer intervals $\mathcal{I}_i^j \triangleq [k_{i-1}^j + 1, k_i^j]$.

Now, we select indices from these partitions and aggregate them to resolve the critical indices. These selected indices construct a new edge e_j. Following Observation 2, this operation would reduce the problem of generating $(m-j+1)$ edges satisfying $(a)^j$ and $(b)^j$ to generating $(m-j)$ edges satisfying $(a)^{j+1}$ and $(b)^{j+1}$ conditioned on e_j.

Specifically, in Line 7, the algorithm begins the edge construction considering the edge e_j to be empty. In Lines 8–13, Algorithm 2 selects *batches of vertices from integer interval $\mathcal{I}_i^j$ of indices and reduce* 1 *from them till all the critical vertices k_i^j's are considered.* As these batches of vertices are selected, they are incrementally added to e_j.

Now, we elaborate selection of the batches of vertices from these intervals as executed in Lines 9–10. At the i^{th} step of selecting vertices, the algorithm uniformly at random select o_i indices from $\mathcal{I}_i^j$. o_i is an integer uniformly sampled from the following lower and upper bounds:

- *Lower bound:* Since at least n_i^j vertices have to be selected from $[1, k_i^j]$ to reinstate dominance and $|e_j|$ vertices have already been selected from $[1, k_{i-1}^j]$, the algorithm needs to select at least $n_i^j - |e_j|$ vertices from $\mathcal{I}_i^j$
- *Upper bound:* There are $(k_i^j - k_{i-1}^j)$ indices in interval $\mathcal{I}_i^j$. After selecting $|e_j|$ vertices, the algorithm can not select more than $b_1^j - |e_j|$ vertices. Thus, the maximum number of vertices selected from $\mathcal{I}_i^j$ is $\min(k_i^j - k_{i-1}^j, b_1^j - |e_j|)$.

Subsequently, the algorithm adds the o_i vertices at those indices to the partially constructed edge e_j (Line 11) and reduce the components at those selected indices in sequence $(a)^j$ by 1 (Line 12).

After adding the edge e_j to the edge set E (Line 14), the algorithm sorts $(a)^j$ in descending order to construct $(a)^{j+1}$, removes b_1^j from $(b)^j$ to construct $(b)^{j+1}$ (Line 15–16). In next iteration, the algorithm focuses on generating $(m-j)$ edges satisfying $(a)^{j+1}$ and $(b)^{j+1}$ conditioned on e_j.

In order to prove correctness of Algorithm 2, we prove Theorems 3 and 4.

Theorem 3. *If the sequences $(a)^j$ and $(b)^j$ are realisable by a hypergraph with m edges and n vertices, the sequences $(a)^{j+1}$ and $(b)^{j+1}$ as constructed by the algorithm at iteration $j+1$ are realisable by a hypergraph with $(m-1)$ edges and n vertices.*

Proof Sketch. This proof is similar to the proof of Theorem 2 in spirit. The only difference is in the inductive step, where we need to prove that the choice of batches of vertices leads to $(a)^{j+1}$ and $(b)^{j+1}$ such that $(a)^{j+1} \prec (\bar{b})^{j+1}$.

After reducing 1 from the selected indices in $(a)^j$, the resulting sequence $(a)^{j+1}$ follows the inequality $\sum_{i=1}^{k} a_i^{j+1} \leq \sum_{i=1}^{k} \bar{b}_i^{j+1}$ at every index $k < n$.

Following Eq. 2, if index k is critical, $\sum_{i=1}^{k} a_i^{j+1} \leq \sum_{i=1}^{k} a_i^j - n_k = \sum_{i=1}^{k} \bar{b}^{j+1}$. If k is not critical, $\sum_{i=1}^{k} a_i^{j+1} < \sum_{i=1}^{k} a_i^j \leq \sum_{i=1}^{k} \bar{b}^{j+1}$ by Eq. (1). After all the critical indices are considered, $\sum_{i=1}^{n} a_i^{j+1} = (\sum_{i=1}^{n} a_i^j) - b_1^j = (\sum_{i=1}^{n} b_i^j) - b_1^j = \sum_{i=1}^{n} b_i^{j+1}$. Consequently, $(a)^{j+1} \prec (\bar{b})^{j+1}$.

Theorem 4. *Algorithm 2 constructs every hypergraph realisation of $(a)_n, (b)_m$ with a non-zero probability.*

Proof sketch. Let us begin with an arbitrary hypergraph realisation $H_1 = (V, E_1 = \{e_1, \ldots, e_m\})$ of sequences $(a)^1 = (a)_n, (b)^1 = (b)_m$ such that $|e_1| \geq |e_2| \geq \ldots \geq |e_m|$.

At iteration 1, Algorithm 2 allocates vertices to edge e_1 with a probability $\mathbb{P}(e_1) = \frac{o_1}{k_1(\min(b_1^1, k_1^1) - n_1^1 + 1)} \frac{o_2}{(k_2^1 - k_1^1)(\min(k_2^1 - k_1^1, b_1^1 - o_1) - (n_2^1 - o_1) + 1)} \ldots$. $\mathbb{P}(e_1)$ is non-zero. Compute the conditional probabilities $\mathbb{P}(e_2|e_1), \ldots, \mathbb{P}(e_m|e_{m-1}, \ldots, e_1)$ in a similar manner. Each of the probabilities is non-zero as a consequence of Theorem 3. *The joint probability* with which the algorithm constructs the edge-sequence $E_1 \triangleq \{e_1, \ldots, e_m\}$ is $\mathbb{P}(E_1) \triangleq \mathbb{P}(e_1)\mathbb{P}(e_2|e_1)\ldots\mathbb{P}(e_m|e_{m-1}, \ldots, e_1)$. $\mathbb{P}(E_1) > 0$ as it is a product of non-zero terms. There are $c(E_1) = \frac{m!}{\prod_j mult^{E_1}(e_j)!}$ distinct permutations of E_1 that result in the same hypergraph as H_1. Here, $mult^{E_1}(e_j)$ is the multiplicity of edge e_j in multiset E_1. Let$[E_1]$ be the set of all permutations of E_1. Thus, the algorithm constructs H_1 with probability $\mathbb{P}(H_1) \triangleq \sum_{E \in [E_1]} \mathbb{P}(E)$. $\mathbb{P}(H_1)$ being a sum of non-zero terms, is non-zero.

6 Self-Normalised Importance Sampling Estimator

In practice, it is desirable to apply a generation algorithm that samples hypergraphs from an uniform distribution over the population of hypergraphs $\mathcal{H}_{\text{ab}}$ having the prescribed degree and dimension sequences $(a)_n$ and $(b)_m$. Uniform generation is desirable, as uniformly generated sample hypergraphs from $\mathcal{H}_{\text{ab}}$ can be used to estimate properties of the hypergraph population $\mathcal{H}_{\text{ab}}$.

However, enumerating all hypergraphs from the population $\mathcal{H}_{\text{ab}}$ is computationally infeasible as the problem of explicit enumeration of $(0,1)$-matrices with given row- and column-sums is **#P**-hard [11]. This result not only makes unbiased estimation of properties of $\mathcal{H}_{\text{ab}}$ computationally infeasible but also hardens the validation of uniformity or unbiasedness of any random generation algorithm. Testing whether a generation algorithm is uniform using state-of-the art algorithms for uniformity testing [1] for the unknown discrete space $\mathcal{H}_{\text{ab}}$ is also computationally infeasible due to the astronomically large number of samples required by the testing algorithm.

The inaccessibility of population space $\mathcal{H}_{\text{ab}}$ motivates us to design an *importance sampling based estimator*. We use this estimator to estimate properties of hypergraphs in $\mathcal{H}_{\text{ab}}$ even if the induced distribution of generation algorithm is not uniform. Importance sampling assigns weights to estimates derived from non-uniformly generated hypergraph samples using the probability at which the hypergraphs are generated. Importance sampling has been adopted to design estimators for properties of matrices [7] and graphs [5]. We adopt it to design an estimator for properties of hypergraphs.

Stepwise Design of the Estimator. Let the uniform distribution over hypergraphs $H \in \mathcal{H}_{\text{ab}}$ be $\mathcal{U}(H) \triangleq \frac{1}{|\mathcal{H}_{\text{ab}}|}$. We are interested in estimating expected value $\mathbb{E}_{\mathcal{U}}[f]$ of some hypergraph property $f : \mathcal{H}_{\text{ab}} \rightarrow \mathbb{R}$. For example, f can be the average clustering coefficient of the projection of the hypergraphs to graphs. If we are able to access $\mathcal{U}$ and draw N i.i.d samples $H'_1, \ldots, H'_N$ accordingly, the Monte Carlo estimate of $\mu(f) \triangleq \mathbb{E}_{\mathcal{U}}[f]$ is $\frac{1}{N}\sum_{i=1}^{N} f(H'_i)$. In practice, computing $\mu(f)$ is not feasible as $|\,\mathcal{H}_{\text{ab}}\,|$ is computationally infeasible to compute.

Thus, we draw N independent edge-sequences $E_1, \ldots, E_N$ from the space of edge-sequences $\mathcal{E}_{ab}$ leading to the hypergraphs in $\mathcal{H}_{ab}$. Using Algorithm 2, we generate N such edge-sequences $\{E_i\}_{i=1}^{N}$ with probabilities $\{\mathbb{P}(E_i)\}_{i=1}^{N}$ respectively. We denote the hypergraph associated to an edge-sequence E_i as $H(E_i)$. The uniform distribution $\mathcal{U}$ over the space of hypergraphs $\mathcal{H}_{ab}$ induces a distribution denoted as $\hat{\mathcal{U}}$ over the space of edge-sequences $\mathcal{E}_{ab}$. Subsequently, we evaluate property f on the generated hypergraphs $\{H(E_i)\}_{i=1}^{N}$ and apply Eq. 3 to estimate the population mean $\mu(f)$.

$$\hat{\mu}(f) = \frac{1}{N}\sum_{i=1}^{N} \frac{\hat{\mathcal{U}}(E_i)}{\mathbb{P}(E_i)} f(H(E_i)) \triangleq \sum_{i=1}^{N} w_i f(H(E_i)) \tag{3}$$

This is analogous to endowing an *importance weight* w_i to a sample $H(E_i)$. The sample mean $\hat{\mu}$ is an unbiased estimator of population mean μ, the proof of which we omit for brevity.

Computing $\hat{\mu}$ is infeasible, as it requires computing $\hat{\mathcal{U}}$ which again requires $|\,\mathcal{H}_{\text{ab}}\,|$ and consequently $|\mathcal{E}_{ab}|$ to be computed. Hence we adopt a self-normalised importance sampling estimator (SNIS) that uses normalised weights $w_i^{\text{SNIS}} \triangleq \frac{w_i}{\sum_i w_i}$. Although SNIS is a biased estimator, it works well in practice [5,7].

$$\tilde{\mu}(f) \triangleq \frac{\sum_{i=1}^{N} \frac{\hat{\mathcal{U}}(E_i)}{\mathbb{P}(E_i)} f(H(E_i))}{\sum_{i=1}^{N} \frac{\hat{\mathcal{U}}(E_i)}{\mathbb{P}(E_i)}} = \sum_{i=1}^{N} \frac{\frac{1}{\mathbb{P}(E_i)}}{\sum_{i=1}^{N} \frac{1}{\mathbb{P}(E_i)}} f(H(E_i)) \triangleq \sum_{i=1}^{N} w_i^{\text{SNIS}} f(H(E_i)) \tag{SNIS}$$

The effectiveness of the importance sampling estimator $\hat{\mu}$ in estimating f is theoretically defined as the effective sampling size $ESS \triangleq N\frac{Var[\mu(f)]}{Var[\tilde{\mu}(f)]}$ and often approximated for SNIS estimator $\tilde{\mu}$ as $\frac{1}{\sum_{i=1}^{N}(w_i^{\text{SNIS}})^2}$ [16]. ESS represents the number of i.i.d samples from $\mathcal{U}$ required to obtain a Monte Carlo estimator $\tilde{\mu}$ with the same accuracy as that of the uniform estimator μ.

7 Performance Evaluation

In order to evaluate the effectiveness of Algorithm 2, we generate multiple random hypergraphs, project the random hypergraphs into simple unweighted graphs, and empirically estimate ($\tilde{\mu}(CC)$) the *average clustering coefficient* (CC) on the projected graphs. For simplicity, *we use the alias SNIS to imply the algorithmic pipeline of generating several sample hypergraphs using Algorithm 2 and then applying the estimator of Equation SNIS.*

In order to evaluate the efficiency of our generation algorithm (Algorithm 2), we measure the CPU-time to generate a certain number of random hypergraphs on different datasets.

7.1 Datasets

We use six graphs and two hypergraph datasets to evaluate the performance of Algorithm 2 and compare with that of the MCMC algorithm.

Graphs. We use the pseudo-fractal family of scale-free simple graphs [10]. Pseudo-fractal graphs are a family (G_t), for integer t, of simple graphs where every graph G_t has $3^t, \ldots, 3^2, 3, 3$ vertices of degree $2, \ldots, 2^t, 2^{t+1}$ respectively. The average clustering coefficient CC_t of graph G_t is $\frac{4}{5}\frac{6^t+3/2}{2^t(2^t+1)}$ and approaches $4/5$ as t grows [10]. We are unaware of any analytical form for the average clustering coefficient of projected random graphs[3] generated following the same degree sequence as G_t. However, we observe (Fig. 1) that the empirical expected value of CC_t converges to $\sim$0.27 as t grows. We construct six graphs $\{G_1, \ldots, G_6\}$ from this family. $\{G_1, \ldots, G_6\}$ have degree sequences of sizes 6, 15, 42, 123, 366 and 1095 respectively, and dimension sequence of sizes 9, 27, 81, 243, 729 and 2187 respectively. The dimension sequence of each graph is a sequence of 2's.

Hypergraphs. We use the Enron email correspondences and legislative bills in US congress as hypergraph datasets [3](https://www.cs.cornell.edu/~arb/data/). In *Enron* dataset, the vertices are email addresses at Enron and an edge is comprised of the sender and all recipients of the email. The degree and dimension sequences are of sizes 4423 and 15653 respectively. In *congress-bills* dataset, the vertices are congresspersons and an edge is comprised of the sponsor and co-sponsors (supporters) of a legislative bill put forth in both the House of Representatives and the Senate. The degree and dimension sequences are of sizes 1718 and 260851 respectively.

7.2 Competing Algorithms

We compare the performance of *SNIS algorithm*, i.e. the SNIS estimator built on our random generation algorithm, with the *MCMC algorithm* [8]. We make two design choices regarding the MCMC algorithm. At first, as the choice for initial hypergraph, we use our construction Algorithm 2. Secondly, as the choice for how many iterations to run the Markov chain, we perform autocorrelation analysis on the Markov chain to select a lag value l. After selecting l, we select

[3] Parallel-edges are absent after projection.

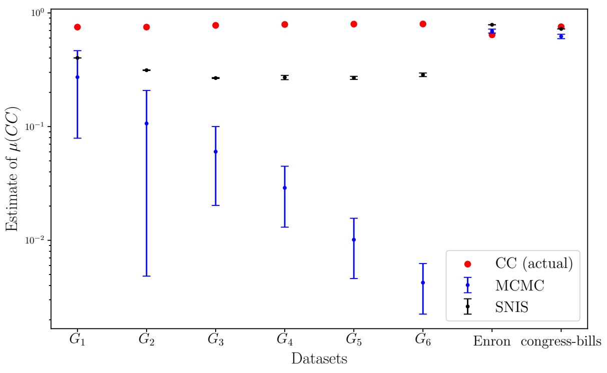

Fig. 1. Average clustering coefficients (in log-scale) of the projected random hypergraphs of different datasets and corresponding estimates $\mu(CC)$ using SNIS and MCMC algorithms.

random hypergraphs from the chain at every l-th hop until required number of hypergraphs are generated. Following standard autocorrelation analysis on Markov chain literature [9], l is selected as the lag at which the autocorrelation function of average clustering coefficient estimate drops below 0.001. On datasets G_1, G_2, G_3, G_4, G_5, G_6, Enron and congress-bills, we observed and used lag values of 17, 23, 115, 129, 90, 304, 9958 and 905 respectively.

7.3 Effectiveness

Comparative Analysis of Estimates of $\mu(CC)$. On graph dataset G_1–G_6, we construct 500 random graphs (without loops) using both MCMC and Algorithm 2. On dataset *Enron* and *congress-bills*, we generate 100 and 20 random hypergraphs (without loops) respectively using both MCMC and Algorithm 2. In Fig. 1, we illustrate CC estimates derived using SNIS, MCMC and the actual dataset. In Fig. 1, we observe that on average after projection the value of clustering coefficient of the multi-graph is much less than that of a simple graph. We also observe that the average clustering coefficient for the hypergraphs empirically converge to 0.27 while the average clustering coefficient of corresponding simple graphs converge to 0.8. This observation is rather expected as parallel-edges decrease the number of triadic closures that would have existed in simple graph. We also observe that, the standard deviation of the SNIS estimates are in significantly smaller than that of the MCMC estimates and closer to the CC of actual data. On Enron and congress-bills hypergraphs, MCMC and SNIS yield comparable estimate for CC. *Figure 1 indicates that in practice the efficiency and stability of SNIS is either competitive or better than that of MCMC.*

Effective Sample Sizes of Estimates of $\mu(CC)$. Effective sample size (ESS) (Sect. 6) represents the number of i.i.d samples from an uniform sampler required to obtain a uniform Monte Carlo estimator with the same accuracy

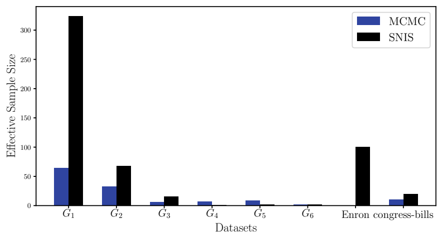

Fig. 2. Effective sample sizes of SNIS and MCMC algorithms on G_1–G_6, Enron and congress-bills datasets. Higher effective sample size indicates better quality of samples.

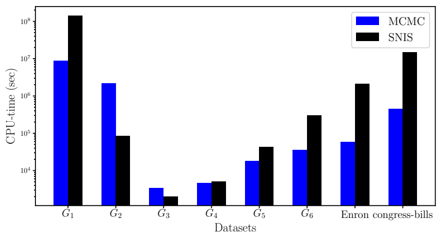

Fig. 3. CPU-time (in second, log-scale) to generate 500 hypergraphs for G_1–G_6, 100 hypergraphs for Enron and 20 hypergraphs for congress-bills datasets. Lower CPU-time is better.

as that of the SNIS estimator E_p. ESS of SNIS algorithm is approximated by $(\sum_{i=1}^{N}(w_i^{\text{SNIS}})^2)^{-1}$. The ESS of MCMC samples is defined as $\frac{N}{1+2\sum_{l=1}^{\infty}\rho(CC^l)}$ [9], where $\rho(CC^l)$ is the *autocorrelation* function at lag l. We consider the summation up-to the lag value for which the autocorrelation drops less than 0.001. We compute the ESS of estimate of CC from both MCMC and SNIS algorithms and plot them in Fig. 2. In Fig. 2, we observe that the SNIS estimate of CC exhibits higher effective sample size than the estimate using MCMC algorithm. *This observation implies that one can estimate CC using much less number of SNIS samples than MCMC samples.* Although the distinction is not much when the hypergraphs are dense, as apparent from similar values of SNIS for graphs G_4, G_5 and G_6.

7.4 Efficiency

We measure the total CPU time (in seconds) taken by the MCMC and Algorithm 2 to generate 500 random graphs for the datasets G_1–G_6, 100 random hypergraphs for Enron dataset, and 20 random hypergraphs for congress-bills datasets respectively. We plot the CPU times in Fig. 3 for the datasets under consideration. In Fig. 3, we observe that the MCMC algorithm is time-efficient than Algorithm 2. In particular, it takes less CPU time in generating random hypergraphs with relatively large number of vertices and edges. However, since each run of Algorithm 2 generates hypergraphs independently from previous runs, multiple such hypergraphs can be generated in parallel for the purpose of property estimation. However, such generation is not possible using MCMC algorithm, as previously generated hypergraph are used to switch edges and generate a new hypergraph. We leave potential parallelism as a future work.

8 Conclusion

We present two algorithms for construction as well as random generation of hypergraphs with prescribed degree and dimension sequences. Our algorithms leverage the transposition of properties and algorithms devised for $(0, 1)$-matrices with prescribed row- and column-sums to hypergraphs. We prove the correctness of the proposed algorithms. We also prove that the generation algorithm generates any random hypergraph following prescribed degree and dimension sequences with non-zero probability.

We propose a self-normalised importance sampling (SNIS) estimator to estimate hypergraph properties and use it to empirically evaluate the effectiveness of random generation.

We compare the effectiveness of the generation algorithm by comparing the SNIS and MCMC estimates of the average clustering coefficient of the projected graphs obtained from the family of hypergraphs having prescribed degree and dimension sequences. As another measure of quality, we compare the effective sample sizes of the SNIS and MCMC estimates.

Experimental results reveal that the SNIS estimates are often more accurate and stable at estimating the average clustering coefficient and have higher effective sample sizes compared to the MCMC estimates. Although the present implementation of our generation algorithm takes longer to generate the same number of samples than the MCMC algorithm, we are currently devising a parallel version of our algorithm.

Acknowledgement. This work was funded by Singapore Institute for Data Science under project WATCHA, by MOE2017-T2-1-141 grant from Singapore Ministry of Education, by WASP-NTU joint grant for Adversarial Machine Learning in Big Data Era by the Knut and Alice Wallenberg Foundation, and by National Research Foundation (NRF) Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd.

References

1. Batu, T., Canonne, C.L.: Generalized uniformity testing. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). IEEE (2017)
2. Bender, E.A., Canfield, E.R.: The asymptotic number of labeled graphs with given degree sequences. J. Combin. Theory, Ser. A **24**(3), 296–307 (1978)
3. Benson, A.R., Abebe, R., Schaub, M.T., Jadbabaie, A., Kleinberg, J.: Simplicial closure and higher-order link prediction. Proc. Nat. Acad. Sci. **115**(48), E11221–E11230 (2018)
4. Berge, C.: Graphs and Hypergraphs. Elsevier Science Ltd. (1985)
5. Blitzstein, J., Diaconis, P.: A sequential importance sampling algorithm for generating random graphs with prescribed degrees. Internet Math. **6**(4), 489–522 (2011)
6. Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. Eur. J. Comb. **1**(4), 311–316 (1980)
7. Chen, Y., Diaconis, P., Holmes, S.P., Liu, J.S.: Sequential Monte Carlo methods for statistical analysis of tables. J. Am. Stat. Assoc. **100**(469), 109–120 (2005)
8. Chodrow, P.S.: Configuration models of random hypergraphs. arXiv preprint arXiv:1902.09302 pp. 1–20 (2019)
9. Cowles, M.K., Carlin, B.P.: Markov chain Monte Carlo convergence diagnostics: a comparative review. J. Am. Stat. Assoc. **91**(434), 883–904 (1996)

10. Dorogovtsev, S.N., Goltsev, A.V., Mendes, J.F.F.: Pseudofractal scale-free web. Phys. Rev. E **65**(6), 066122 (2002)
11. Dyer, M., Kannan, R., Mount, J.: Sampling contingency tables. Random Struct. Algorithms **10**(4), 487–506 (1997)
12. Fosdick, B.K., Larremore, D.B., Nishimura, J., Ugander, J.: Configuring random graph models with fixed degree sequences. SIAM Rev. **60**(2), 315–355 (2018)
13. Fulkerson, D.R., Ryser, H.J.: Multiplicities and minimal widths for (0, 1)-matrices. Can. J. Math. **14**, 498–508 (1962)
14. Gale, D., et al.: A theorem on flows in networks. Pac. J. Math. **7**(2), 1073–1082 (1957)
15. Klamt, S., Haus, U.U., Theis, F.: Hypergraphs and cellular networks. PLoS Comput. Biol. **5**(5), e1000385 (2009)
16. Kong, A.: A note on importance sampling using standardized weights. University of Chicago, Department of Statistics, Technical report 348 (1992)
17. Marshall, A.W., Olkin, I., Arnold, B.C.: Inequalities: Theory of Majorization and Its Applications, vol. 143. Springer, New York (1979). https://doi.org/10.1007/978-0-387-68276-1
18. Milo, R., Kashtan, N., Itzkovitz, S., Newman, M.E., Alon, U.: On the uniform generation of random graphs with prescribed degree sequences. arXiv preprint cond-mat/0312028 (2003)
19. Newman, M.: Networks: An Introduction. Oxford University Press, Oxford (2018)
20. Ryser, H.: Combinatorial properties of matrices of zeros and ones. Can. J. Math. **9**, 371–377 (1957)
21. Yang, W., Wang, G., Bhuiyan, M., Choo, K.: Hypergraph partitioning for social networks based on information entropy modularity. J. Netw. Comput. Appl. **86**, 59–71 (2016)
22. Wang, Y., Zheng, B.: Hypergraph index: an index for context-aware nearest neighbor query on social networks. Soc. Netw. Anal. Min. **3**(4), 813–828 (2013)