Constructing and sampling directed graphs with given degree sequences

H Kim[1,2], C I Del Genio[3], K E Bassler[4,5] and Z Toroczkai[2]

[1] Department of Physics, Virginia Tech, Blacksburg, VA, 24061, USA
[2] Interdisciplinary Center for Network Science and Applications (iCeNSA), Department of Physics, University of Notre Dame, Notre Dame, IN 46556, USA
[3] Max-Planck-Institut für Physik Komplexer Systeme, Nöthnitzer Str. 38, D-01187 Dresden, Deutschland
[4] Department of Physics, University of Houston, 617 Science and Research 1, Houston, Texas 77204-5005, USA
[5] Texas Center for Superconductivity, University of Houston, 202 Houston Science Center, Houston, Texas 77204-5002, USA

E-mail: `toro@nd.edu`

Abstract. The interactions between the components of complex networks are often directed. Proper modeling of such systems frequently requires the construction of ensembles of digraphs with a given sequence of in- and out-degrees. As the number of simple labeled graphs with a given degree sequence is typically very large even for short sequences, sampling methods are needed for statistical studies. Currently, there are two main classes of methods that generate samples. One of the existing methods first generates a restricted class of graphs, then uses a Markov Chain Monte-Carlo algorithm based on edge swaps to generate other realizations. As the mixing time of this process is still unknown, the independence of the samples is not well controlled. The other class of methods is based on the Configuration Model that may lead to unacceptably many sample rejections due to self-loops and multiple edges. Here we present an algorithm that can directly construct all possible realizations of a given bi-degree sequence by simple digraphs. Our method is rejection free, guarantees the independence of the constructed samples, and provides their weight. The weights can then be used to compute statistical averages of network observables as if they were obtained from uniformly distributed sampling, or from any other chosen distribution.

PACS numbers: 02.10.Ox, 02.50.Ey, 89.75.Hc, 07.05.Tp

1. Introduction and definitions

In network modeling problems [1, 2, 3, 4, 5, 6, 7], one often needs to generate ensembles of graphs obeying a given constraint. A typical constraint is the case when the only information available is the degrees of the nodes, and not the actual connectivity matrix. Note that the node degrees by themselves, that is the *degree sequence* in general does not determine a graph uniquely: there can be a very large number of graphs having the same degree sequence [8]. Full graph connectivity is uniquely determined by the degree sequence only for a special class of sequences (see Ref. [9] for the case of undirected graphs).

Often, the interest lies in the study of network observables, *as determined* by the given sequence of degrees, and unbiased by anything else. These can be graph theoretical measures, or properties of processes happening on the network (e.g., spreading processes, such as of opinion or disease). The problem of creating and sampling graphs with a given degree sequence, i.e., *degree-based graph construction* [10, 11], is a well-known and challenging problem that has attracted considerable interest amongst researchers [8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30]. There are two main classes of algorithms that are used today to achieve the construction of graphs with given degree sequences. One of them is typically referred to as "switching" or edge-swap based [13, 15, 16, 25, 27], while the other one is usually called "matching" or stub-matching based [8, 12, 14, 17, 31, 32, 33, 26, 34]. Switching methods repeatedly swap the ends of two randomly chosen edges within a Markov Chain Monte-Carlo (MCMC) scheme until a new, quasi-independent, sample is produced. Unfortunately, the mixing time of MCMC schemes for arbitrary sequences is not known in the general case. The other class consists of direct construction methods, which perform pairwise matchings of the half-edges emanating from randomly chosen nodes until all edges are realized. Unfortunately, this method can easily generate multiple edges and self-loops, i.e., edges starting and ending on the same node, after which the sample must be rejected in order to avoid biases [35]. For a comparison of the two classes of methods see Ref. [23].

Recently, a novel degree-based construction [10] and sampling method [11] was introduced for undirected graphs, which has a worst-case scaling of $\mathcal{O}(NM)$, where M is the number of edges ($2M$ is the sum of the degrees, which are given). A similar method was obtained independently in Ref. [34], but that method is less efficient, with a worst-case scaling of $\mathcal{O}(N^2M)$. Although the algorithm in Ref. [11] is a direct construction method using stub-matchings, it is rejection free, the samples are statistically independent and the algorithm also provides a weight for every realization.

In many systems the interaction between two entities is not mutual but has a direction from one to the other, such as in the cases of human relationships in social networks [36], gene interactions in regulatory networks, trophic interactions in food webs [20, 21], etc. Such systems require a representation by directed graphs (digraphs). In fact, undirected graphs can be interpreted as digraphs in which there are two, oppositely directed edges for each connected pair of nodes. Here we present a generalization of the degree-based graph construction problem to directed graphs. Some of the necessary mathematical foundations, laid down in Ref. [30], are here used and expanded to introduce a digraph construction and sampling algorithm. Although the approach follows closely the one introduced by us for the undirected case [11], the generalization is not at all straightforward, and there are significant differences that the directed nature of the links induces.

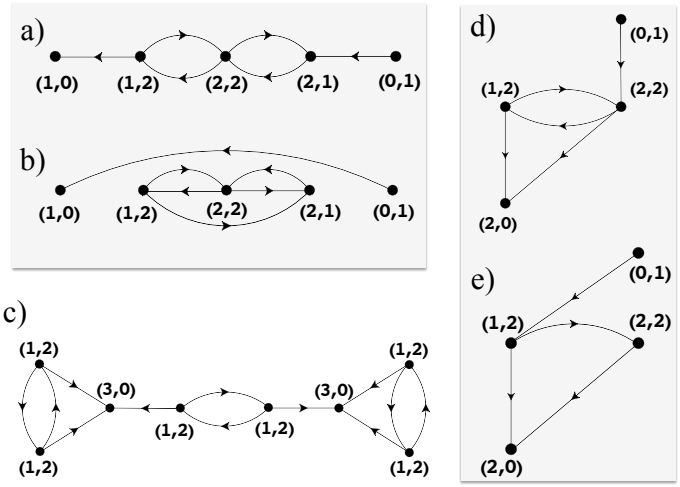

Figure 1. Examples of realizations of graphical bi-degree sequences. Panels a) and b) show two non-isomorphic realizations of the same bds. Panel c) shows a digraph that cannot be obtained via the Havel-Hakimi algorithm for digraphs. Panel d) shows a realization of a different bds. Panel e) illustrates that not all possible connections lead to a simple digraph even if a bds is graphical: in fact, the connections in the figure break the graphical character.

Before we present our algorithm, we introduce some notations, based on Ref [30]. Let us denote by $d_i^{(i)}$ and $d_i^{(o)}$ the in- and out-degrees of a node i. Given the sequence $\mathbf{D} = \left\{ \left(d_1^{(i)}, d_1^{(o)}\right), \left(d_2^{(i)}, d_2^{(o)}\right), \ldots, \left(d_N^{(i)}, d_N^{(o)}\right) \right\}$ of non-negative integer pairs, we want to construct a *simple* directed graph $\mathbf{G}(V, \mathbf{E})$ such that node $k \in V$ has $(d_k^{(i)}, d_k^{(o)})$ for its in- and out-degrees, respectively, for all $k = 1, 2, \ldots, N$. A simple directed graph is a graph that has no self-loops, nor multiple directed edges in the same direction between two nodes. There can be at most two edges between a pair of nodes, oppositely directed. We call the sequence $\mathbf{D}$ a bi-degree sequence (bds for short). When there is a simple digraph with a given bds $\mathbf{D}$ for its degrees, we say that the bds is *graphical* and that the digraph realizes $\mathbf{D}$. Equivalently, we will also talk about "graphicality" as a property. We distinguish realizations as *labeled* digraphs, and do not deal here with isomorphism questions. That is, if two realizations are identical up to a permutation of their indices, i.e., they are isomorphic, we will still consider them distinctly. In order to avoid isolated nodes, in the following we will assume that $d_j^{(i)} + d_j^{(o)} > 0$, for all $j = 1, \ldots, N$. As examples, Figs. 1a) and 1b) show two realizations of the bds $\mathbf{D}_1 = \{(1,0), (1,2), (2,2), (2,1), (0,1)\}$, and Fig. 1c) shows a realization of $\mathbf{D}_2 = \{(3,0), (3,0), (1,2), (1,2), (1,2), (1,2), (1,2), (1,2)\}$. Examples of non-graphical bds are the sequences $\mathbf{D}_3 = \{(2,2), (2,1), (1,3), (1,1)\}$ and $\mathbf{D}_4 = \{(5,6), (5,6), (5,6), (4,3), (3,3), (2,1), (2,1), (1,1)\}$.

Notice that even if a bds is graphical, not all connection sequences are guaranteed to end up with a simple digraph. For example, Fig. 1d) shows a simple digraph realization of $\mathbf{D}_5 = \{(0,1), (2,0), (1,2), (2,2)\}$. However, if we were to place the first four edges as in Fig. 1e), we would break graphicality: from there on, we would not be able to complete the realization of the bds without creating either self-loops or

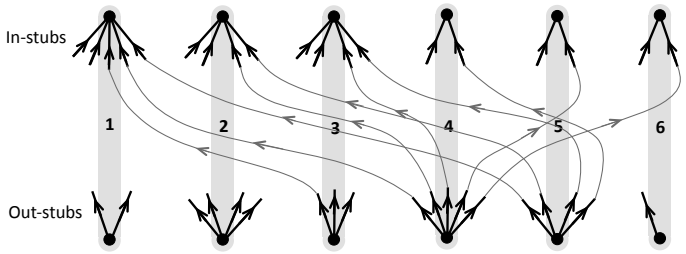

Figure 2. The construction of a digraph realizing a given bds proceeds by connecting the out-stubs of the nodes to the in-stubs of other nodes. In this "bipartite" representation the vertical grey bars represents single nodes.

multiple edges. Hence, it is important to find an algorithm that builds digraphs with a given bds. As we will see, this is a challenging problem in itself.

An algorithm that builds a digraph from a given bds sequentially connects the out-links of a node to the in-links of others. We can think of these out- and in-links as "out-stubs" and "in-stubs" emanating from a node, that are paired up with the corresponding stubs of other nodes. An intuitive representation of this is shown in Fig. 2. As the graph construction algorithm proceeds, the number of stubs of the nodes decreases. At any time during this process we will call the number of remaining in-stubs and out-stubs of a node its *residual* in- and out-degrees, and the corresponding bi-degree sequence $\overline{\mathbf{D}} = \left\{ \left(\bar{d}_1^{(i)}, \bar{d}_1^{(o)}\right), \left(\bar{d}_2^{(i)}, \bar{d}_2^{(o)}\right) \ldots, \left(\bar{d}_N^{(i)}, \bar{d}_N^{(o)}\right) \right\}$ the *residual bds*.

Finally, another concept we will need to use in what follows is the notion of *normal order* [30], which is essentially the lexicographic order on the bds. That is, we say that a bds is in normal order, if for all $1 \leq j \leq N-1$, we have either $d_j^{(i)} > d_{j+1}^{(i)}$ or, if $d_j^{(i)} = d_{j+1}^{(i)}$, then $d_j^{(o)} \geq d_{j+1}^{(o)}$. Thus, the bds $\mathbf{D}_6 = \{(5,2),(4,4),(4,3),(2,5),(2,4),(2,1)\}$ shown in Fig. 2, is arranged in normal order. Once a bds is in normal order, we will use the words 'left' or 'right' to describe the directions towards lower or higher index values in the sequence.

The remainder of this paper is organized as follows: Section 2 introduces the fundamental mathematical notions and algorithmic considerations that are at the basis of our digraph construction algorithm. Section 3 presents the algorithm and its derivation details. Readers interested only in the algorithm itself may skip Subsection 3.1 and proceed to the summary described in the beginning of Section 3 and in Subsection 3.2. Section 4 deals in detail with the digraph sampling problem, provides the derivation of the sample weights and presents a simple example. Section 5 is dedicated to the complexity of the algorithm, and Section 6 concludes the paper.

2. Mathematical foundations

As seen from the examples above, not all sequences of non-negative integer pairs can be realized by simple digraphs. The sufficient and necessary conditions for the realizability of a bds are given by the "FR" theorem [37, 38, 39]:

Theorem 1 (Fulkerson-Ryser) *A sequence of non-negative integer pairs* $\mathbf{D} =$

$\left\{\left(d_1^{(i)}, d_1^{(o)}\right), \ldots, \left(d_N^{(i)}, d_N^{(o)}\right)\right\}$ *with* $d_1^{(i)} \geq d_2^{(i)} \geq \ldots \geq d_N^{(i)}$ *is graphical iff*

$$d_i^{(i)} \leq N-1\ , \quad d_i^{(o)} \leq N-1, \quad 1 \leq i \leq N \tag{1}$$

$$\sum_{i=1}^{N} d_i^{(i)} = \sum_{i=1}^{N} d_i^{(o)}\ , \tag{2}$$

and for all $1 \leq k \leq N-1$:

$$\sum_{i=1}^{k} d_i^{(i)} \leq \sum_{i=1}^{k} \min\left\{k-1, d_i^{(o)}\right\} + \sum_{i=k+1}^{N} \min\left\{k, d_i^{(o)}\right\}\ . \tag{3}$$

Given a bds, we can easily test if it is graphical using this theorem, and thus we will also refer to it as the "FR test". Condition (1) states that both the number of in- and out-degrees for all nodes must be no larger than the number of other nodes it could connect to, or receive connections from. Condition (2) is a consequence of the requirement that every out-stub must join an in-stub somewhere else; the sequence $\mathbf{D}_3$ given in one of the above examples is not graphical because it fails this condition. Condition (3) is less intuitive. Its left hand side is the total number of in-stubs that the group of k highest in-degree nodes can receive. *Within this group*, a node's out-stubs can absorb no more of those in-stubs from the same group than its out-degree or $k-1$ (it cannot absorb from itself), whichever is smaller (giving the first sum on the rhs of (3)). *Outside of this group*, a node cannot absorb more of those in-stubs than its out-degree or k, whichever is smaller (the second sum on the rhs of (3)). Hence, the necessity of (3). For the complete proof see Refs. [38, 39]. Note that the example sequence $\mathbf{D}_4$ above fails condition 3 for $k = 3$. The FR test is the directed version of the Erdős-Gallai (EG) theorem (test) for undirected graphs.

An important note is that bi-degree sequences are *less constraining* than undirected ones. The out-stub of a node is always connected to an in-stub of another, not affecting that node's out-stubs, whereas such distinction does not exist for the undirected case. Alternatively, if we disregard for a moment the directionality of the links and consider the degree of the node to be the sum of its in- and out-degrees, then the corresponding graph realizing the bds can have two edges running between the same pair of nodes, whereas this is not allowed in the undirected case.

2.1. Algorithmic considerations

The FR theorem only tests for graphicality, but it does not provide an algorithm for constructing the digraph(s) realizing the given bds. At first sight this might not seem an issue. However, the sequence $\mathbf{D}_5$ in Figs. 1d and 1e) reminds us that graphicality can easily be broken by a careless connection of stubs. Clearly, for the purposes of digraph construction, it should not matter which edges we create first, as long as we make sure that every connection made does not break graphicality. In other words, the possibility to create the rest of the edges, so that a simple digraph results in the end, must always be preserved. Thus, the key for the creation of an algorithm that builds simple digraphs realizing a given bds without rejections is in a theorem that allows us to check if we would break graphicality by placing a specific connection. Indeed, such theorems exist, and they will be discussed below. However, interestingly, they require that connections be made from the *same node*, until all its stubs are used away into edges. That is, assuming that we already made some connections from a given node i, preserving graphicality, these theorems give necessary and sufficient conditions for

keeping graphicality by the next connection *still* involving node i. Simply put, they won't work in general, if we would attempt a new connection from j to k, where $j, k \neq i$, while node i still has dangling stubs.

The connections already made from i to some set of nodes $\mathcal{X}_i$ represent a *constraint* for the new connections from i, as these novel connections must avoid the set $\mathcal{X}_i$. We call such a constraint associated to a node a *star constraint* on that node. Once all the stubs of node i are connected into edges while preserving graphicality, we obtain a graphical residual sequence $\mathbf{D}'$ on at most $N-1$ nodes. Clearly, the new connections we make from this point on will not be constrained in any way by the connections we made from node i. For the purposes of realizing the sequence $\mathbf{D}'$ we can just simply remove node i with its fully completed connections, create a realization by a simple graph of $\mathbf{D}'$, then, in the end, add back node i with its connections to this graph in order to obtain a realization of $\mathbf{D}$. The comments above hold both for the undirected and directed cases.

One might think of using the EG test for the undirected case and the FR test for the directed case on a residual degree sequence to decide if graphicality was broken after attempting a new connection from the same node. For the *undirected case*, we have shown in Ref. [11] that the passing of the EG test by the residual sequence is only a *necessary condition*, if there is already a star constraint on a node. For example, consider the graphical degree sequence $\mathbf{d} = \{6, 5, 5, 3, 3, 2, 1, 1\}$, and assume that we made connections from node $i = 3$ to nodes $\mathcal{X}_3 = \{1, 6, 7\}$. The residual sequence after these connections is $\mathbf{d}' = \{5, 5, 2, 3, 3, 1, 0, 1\}$. It is easy to check that it passes the EG test. However, we will break graphicality with *every* realization of $\mathbf{d}'$, because it will form a double edge with one of the existing connections from node $i = 3$ to $\mathcal{X}_3$. Thus, additional considerations have to be made to ensure the graphicality of the residual sequence for the undirected case, as described in [11]. For the directed case here we use the sufficient and necessary conditions for graphicality under star constraints as provided by Theorem 2 below, proven in Ref. [30].

From now on, we will always talk about algorithms that first finish all the out-stubs of a node before moving onto another node with non-zero out-degree. In the case of a graphical bds, once all the out-degrees of all the nodes have been connected into directed edges, we are guaranteed to have completed a digraph, because the total number of in-stubs equals the total number of out-stubs, according to property (2).

2.2. Theorems on which the algorithm is based

An algorithm that builds graphical realizations of degree sequences of simple *undirected* graphs is the Havel-Hakimi (HH) algorithm [40, 41]: we choose any node with non-zero residual degree, then we connect all its stubs to nodes with the largest residual degrees avoiding self and multiple connections. This process is repeated with other nodes until all stubs of all nodes are used. There is a corresponding version of the HH algorithm for bi-degree sequences as well, introduced first in Ref. [42], then rediscovered independently in Ref. [30], the latter providing an alternative proof. The HH algorithm for bds proceeds as follows: given a normal-ordered bds, choose any node with non-zero residual out-degree, then connect all its out-stubs to nodes with the largest residual in-degrees, without creating multiple edges running in the same direction, nor self-loops. Reorder in normal order the residual sequence and repeat this process until all stubs of all nodes are used. While for any given bds, the HH algorithm will construct a set of digraphs, it cannot construct *all possible* digraphs

realizing the same sequence, as shown in Ref. [30]. For example, the HH algorithm can never result in the digraph shown in Fig. 1c) realizing the example sequence $\mathbf{D}_2$ above. It is easy to see why: there are two kinds of nodes in this example, with bi-degrees $(3,0)$ and $(1,2)$. The only nodes with non-zero out-degrees are the $(1,2)$ types. Using the HH algorithm, we would have to connect both out-stubs of such a node to the nodes with the largest in-degrees, that is to the two $(3,0)$ types. However, the digraph in Fig. 1c) does not have a $(1,2)$ node being connected to both $(3,0)$ nodes, yet it realizes the sequence. The limitation of the HH algorithm comes from the fact that it prescribes to connect the out-stub of a node i to an in-stub of the node with the *largest* residual in-degree that does not yet receive a connection from node i. However, there can be other nodes whose in-stubs can form a connection with an out-stub of i without breaking graphicality. This shows the importance of finding a method able to build not just *a* realization of a bds, but *all* the possible realizations of any given bds.

In the remainder, given a residual bds $\overline{\mathbf{D}}$, we denote by $\mathcal{A}_i\left(\overline{\mathbf{D}}\right)$ the *allowed set* of i, i.e., the set of all nodes to which an out-stub of i can be connected without breaking graphicality. Also, let us denote by $\mathcal{X}_i\left(\overline{\mathbf{D}}\right)$ the set of nodes to which connections were already made from i, thus representing the star constraint at that stage.

The graphicality test under a star constraint on node i is provided as Theorem 2 below. In order to announce it, however, we need to introduce one more definition. Consider a bds $\mathbf{D}$ and a given node i with out-degree $d_i^{(o)} > 0$ from this bds. Let us also consider a subset of nodes $S \subset V$ such that $|S| \leq d_i^{(o)}$, where $|S|$ denotes the number of nodes in S, i.e., its size, and for every node $j \in S, d_j^{(i)} > 0$. Next, we take $\mathbf{D}$ and reduce by unity the in-degrees of all its nodes in S, then reduce by $|S|$ the out-degree of node i. The bds $\mathbf{D}'$ thus obtained will be called the bds *reduced by* S *about node* i *from bds* $\mathbf{D}$. Equivalently, $\mathbf{D}'$ is the residual sequence obtained from $\mathbf{D}$ after connecting an out-stub from i to an in-stub of every node from S.

Theorem 2 (Star-constrained graphicality) *Let* $\mathbf{D}$ *be a bds in normal order on* N *nodes, and let* $\mathcal{X}_i$, $|\mathcal{X}_i| \leq N-1-d_i^{(o)}$, *be a set of nodes whose in-stubs are forbidden to be connected to the out-stubs of node* i *(including* i*). Define* $\mathcal{L}_i$ *as the set of the first ("leftmost")* $d_i^{(o)}$ *nodes in* $\mathbf{D}$ *but not from* $\mathcal{X}_i$*. Then, there exists a simple digraph which realizes* $\mathbf{D}$ *and avoids connections from* i *to* $\mathcal{X}_i$*, if and only if the bds* $\mathbf{D}'$ *reduced by* $\mathcal{L}_i$ *about node* i *from* $\mathbf{D}$ *is graphical.*

The proof of this theorem is found in Ref. [30]. What this theorem does is to turn a star-constrained graphicality problem for bds $\mathbf{D}$ into an *unconstrained one* on the reduced bds $\mathbf{D}'$. The graphicality of $\mathbf{D}'$ is then easily tested via the FR theorem. The set $\mathcal{L}_i$ as defined above will be called the *leftmost* set for node i.

Although announced in its full generality, as $\mathcal{X}_i$ could be any predefined subset of nodes with $|\mathcal{X}_i| \leq N-1-d_i^{(o)}$, this theorem applies directly to the digraph construction process when $\mathcal{X}_i$ represents the set of nodes to which connections were already made in previous steps from the same node i, hence forbidding us to make further connections from i to these very same nodes. In this case, the bds $\mathbf{D}$ represents the residual sequence $\overline{\mathbf{D}}$ at that stage of the construction process.

As discussed above, in order for us to be able to construct all the simple digraphs that realize a given bds, we need to find the allowed set $\mathcal{A}_i\left(\overline{\mathbf{D}}\right)$ for the next out-stub of i. Clearly, after every connection from the same node i, the residual sequence changes, and along with it the allowed set may change as well. In order to find $\mathcal{A}_i\left(\overline{\mathbf{D}}\right)$

for the next out-stub of node i, we could just simply attempt connections sequentially to every node with non-zero in-degree *not in* $\mathcal{X}_i\left(\overline{\mathbf{D}}\right) \cup \{i\}$, and test for graphicality after each attempt using Th. 2. The set of nodes for which graphicality would have been preserved would form $\mathcal{A}_i\left(\overline{\mathbf{D}}\right)$.

However, this would be inefficient and, actually, not needed. In fact, we can exploit a result which states that, if graphicality is broken by a connection, it will be broken by all other connections to the right of the previous one, in the normal order sense. This is expressed in the following:

Theorem 3 *Let* $\mathbf{D}$ *be a graphical bds in normal order and let* $\mathcal{X}_i$ *be a forbidden set for node* i*, with* $i \in \mathcal{X}_i$*. Let* $j < k$ *be two nodes such that* $j, k \notin \mathcal{X}_i$*. If the residual bds* $\mathbf{D}_j$ *obtained from* $\mathbf{D}$ *after forming an edge directed from* i *to* j *is not graphical, then the bi-degree sequence* $\mathbf{D}_k$ *obtained from* $\mathbf{D}$ *by forming a directed edge from* i *to* k *is also not graphical.*

This theorem follows from the direct contraposition of Lemma 6 in Ref. [30]. Thus, what we need to do is to find efficiently the *leftmost node* q in the residual sequence in normal order, a connection to which would break graphicality. We will refer to this node q as the *leftmost fail-node*. All connections to this node and to nodes to its right are guaranteed to break (star-constrained) graphicality, whereas all connections to its left (with the exception of forbidden nodes and self) are guaranteed to preserve the graphical character.

Note that both theorems 2 and 3 are based on the HH theorem for bi-degree sequences. In fact theorem 2 is a generalization of the HH theorem to include star constraints. Also note that, while for the FR theorem only the in-degrees must be ordered non-increasingly, for the HH theorem and hence for both theorems 2 and 3, the bds must be in normal order, as ordering by in-degrees only is not sufficient. This is easily seen from the following example of graphical bds (not in normal order) $\mathbf{D}_7 = \{(2,0),(2,1),(0,1),(0,2)\}$. Using the HH theorem, if we do not worry about normal ordering, but just order by in-degree, we could choose to connect the out-stub of node $(0,1)$ to an in-stub of node $(2,0)$, then the out-stub of node $(2,1)$ to the remaining in-stub of $(2,0)$ (connecting to the largest residual allowed residual in-degree), after which we have clearly broken graphicality: both out-stubs of $(0,2)$ now must be connected to the two in-stubs of $(2,1)$.

We are now ready to present our digraph construction algorithm, which produces random samples from the set of all *possible* simple digraphs realizing a given bds.

3. The algorithm

Given a graphical bi-degree sequence $\overline{\mathbf{D}}$ in *normal order* (initially $\overline{\mathbf{D}} = \mathbf{D}$):

1) Define as *work-node* the lowest-index node i with non-zero (residual) out-degree.
2) Let $\mathcal{X}_i$ be the set of forbidden nodes for the work-node, which includes i, nodes with zero in-degrees and nodes to which connections were made from i, previously. In the beginning, $\mathcal{X}_i$ includes only the work-node and zero in-degree nodes.
3) Find the set of nodes, $\mathcal{A}_i$ that can be connected to the work-node without breaking graphicality.
4) Choose a node $m \in \mathcal{A}_i$ uniformly at random and connect an out-stub of i to an in-stub of m.
5) After this connection add node m to $\mathcal{X}_i$.

6) If node i still has out-stubs, bring the residual sequence in normal order, then repeat the procedure from 3) until all out-stubs of the work node are connected away into edges.
7) If there are other nodes left with out-stubs, reorder the residual degree sequence in normal order, and repeat from 1).

The most involved step of the algorithm is finding the allowed set (step 3)), which is described next.

3.1. Finding the allowed set

Let i be the work-node chosen as in 1) and let $\overline{\mathbf{D}}$ denote the *normal ordered*, residual sequence obtained after having connected some of the out-stubs of i to in-stubs of other nodes, such that graphicality is still preserved. These previous connections from node i form the set of forbidden nodes $\mathcal{X}_i$ for the next out-stub σ of i. $\mathcal{X}_i$ also contains the work-node i itself $i \in \mathcal{X}_i$ and all other nodes with zero in-degrees. Let $\mathcal{L}_i$ be the set of the first (lowest index) $\overline{d}_i^{(o)}$ nodes from $\overline{\mathbf{D}}$, *not in* $\mathcal{X}_i$. As $\overline{\mathbf{D}}$ is (star constrained) graphical, we can connect σ to any of the nodes in $\mathcal{L}_i$ without breaking graphicality (due to Theorem 2), hence $\mathcal{L}_i \subseteq \mathcal{A}_i$.

Let m be the last element of $\mathcal{L}_i$ in the normal ordered bds $\overline{\mathbf{D}}$ and let us "color" (label) red all the non-forbidden nodes, i.e., all the nodes not in $\mathcal{X}_i$, to the right of node m. Please note that these color labels are associated with the nodes, defined by their bi-degrees, and not with their indices of location in the sequence. This set of red nodes $\mathcal{R}_i$ forms the set of candidates for the leftmost fail-node q. All other nodes are colored (labelled) black. To find the leftmost fail-node we could simply connect out-stub σ to an in-stub of a red node ℓ, add the new connection temporarily to the set of forbidden nodes, bring the new residual sequence into normal order, then test for graphicality using Theorem 2. This procedure could then be repeated sequentially, with ℓ going over all the red nodes from left to right, until graphicality would fail for the first time at $\ell = q$. However, the considerations in the following paragraphs allow us define a better method.

For the sake of argument let us perform the sequential testing as explained above. It would imply the following steps for a given red node ℓ :

(a) Reduce the out-degree at the work-node i and the in-degree at ℓ by unity, that is $\overline{d}_i^{(o)} \mapsto \overline{d}_i^{(o)} - 1$ and $\overline{d}_\ell^{(i)} \mapsto \overline{d}_\ell^{(i)} - 1$, resulting in a new residual bds $\overline{\mathbf{D}}_\ell$.
(b) Bring $\overline{\mathbf{D}}_\ell$ into normal order (required by Theorem 2). Note that ℓ is the only node whose in-degree has changed and only the work-node had its out-degree changed (its in-degree was not affected). Thus, when bringing $\overline{\mathbf{D}}_\ell$ into normal order, the relative positioning of all the other nodes does not change. The work-node might have shifted to the right to a new position i' within the block of nodes with the *same* in-degree ($i' \geq i$), and the red node's new position ℓ' might have also moved to the right in the normal ordered sequence ($\ell' \geq \ell$).
(c) Add ℓ' to the forbidden set for the work-node.
(d) Now, as required by Theorem 2, reduce by unity the in-degrees of the nodes in the left-most adjacency set $\mathcal{L}_{i'}$, and reduce the out-degree of the work-node i' to zero. This results in the new sequence $\overline{\mathbf{D}'}_{\ell'}$.
(e) Order the bds $\overline{\mathbf{D}'}_{\ell'}$ by in-degrees, non-increasingly.
(f) Apply the FR theorem to test for graphicality.

Thus, whether the connection of the work-node i to ℓ breaks graphicality, ultimately depends on whether the residual bds $\overline{\mathbf{D}'}_{\ell'}$ fails (or passes) the FR test. However, as we noted before, for the FR test we do not need to have the bds $\overline{\mathbf{D}'}_{\ell'}$ in normal order, we only need to have it ordered non-increasingly by the in-degrees. Additionally, observe that in step (d) the reduction of the in-degrees always happens on the *same* set of nodes, independently of the red node ℓ, that is the left-most adjacency set $\mathcal{L}_{i'}$ is the same for all ℓ. Thus, in this particular case of Theorem 2's application, ultimately we do not need to bring $\overline{\mathbf{D}}_\ell$ into normal order (step (b)), only non-increasingly by in-degrees, which would be done anyway in step (e). That means we can just skip step (b), we do not need to move around any of the nodes at that stage. Thus, the only difference between the sequences $\overline{\mathbf{D}'}_{\ell'}$ for different ℓ-s is at the position of this node after the reordering in (e), *with respect to the rest of the sequence.*

These observations suggest that we should define a bds $\overline{\mathbf{D}'}$ obtained from the bds $\overline{\mathbf{D}}$ by reducing by unity the in-degrees of all nodes in the set $\mathcal{L}_i \setminus \{m\}$ and by $\overline{d}_i^{(o)} - 1$ the out-degree of the work-node i, leaving only one out-stub (out-stub σ) at i. Clearly, the bds $\overline{\mathbf{D}'}$ is graphical (connecting out-stub σ to an in-stub of node m surely preserves graphicality, by Theorem 2). Let us now order $\overline{\mathbf{D}'}$ non-increasingly by its in-degrees, *in a specific way*, described as follows. Shift only the reduced in-degree nodes in $\overline{\mathbf{D}}$ to the right with respect to the rest of the sequence such as to restore non-increasing ordering by the in-degrees (if needed). Since only the in-degrees of the nodes in the set $\mathcal{L}_i \setminus \{m\}$ have been reduced, *keep* the relative ordering of all other nodes in $\overline{\mathbf{D}'}$ exactly the same as in $\overline{\mathbf{D}}$. Thus the *relative ordering* of the red nodes and of the work node have been preserved as well. Let us denote the new location of the work node in $\overline{\mathbf{D}'}$ by j ($j \leq i$). Connecting now σ to an in-stub of a red node ℓ in this sequence will produce the same set of residual bi-degrees as in step (d) above. To be able apply the FR theorem, then all we need to do is to shift to the right node ℓ in the sequence (if needed) to make sure that it is non-increasingly ordered by in-degrees. Since only the in-degree at ℓ was modified (reduced by unity), this reordering is very simple: if x denotes the location of the last node of the block of nodes with the same in-degree as node ℓ in $\overline{\mathbf{D}'}$ ($x \geq \ell$), then we simply swap the node at ℓ with the node at x after the reduction of the in-degree at ℓ. Let us denote the obtained sequence by $\overline{\mathbf{D}''}$. Clearly, it is non-increasingly ordered by in-degrees, and thus we can apply the FR theorem to see if it is graphical. Note: it could happen that $x = j$ (e.g., there are many nodes with zero out-degree but larger in-degree than the work-node as defined in 1)), however, the steps below can be applied just the same.

Next, we show how to identify the leftmost red fail-node q by investigating how the inequalities in (3) break down. Since $\overline{\mathbf{D}'}$ is graphical, we have for all $1 \leq k \leq n-1$ (n is the last element of $\overline{\mathbf{D}'}$) that $L'(k) \leq R'(k)$, where L' and R' are the left hand side (lhs) and the right hand side (rhs) of inequalities (3) written for $\overline{\mathbf{D}'}$:

$$L'(k) = \sum_{s=1}^{k} \overline{d'}_s^{(i)} \,, \tag{4}$$

$$R'(k) = \sum_{s=1}^{k} \min\left\{k-1, \overline{d'}_s^{(o)}\right\} + \sum_{s=k+1}^{n} \min\left\{k, \overline{d'}_s^{(o)}\right\}. \tag{5}$$

Let us denote by $L''(k)$ and $R''(k)$ the lhs and rhs of the inequality (3) corresponding to $\overline{\mathbf{D}''}$. Since the rhs of (3) involves only out-degrees, and we only reduced the out-degree of the work-node from 1 to 0, we will always have $R''(k) = R'(k) - 1$, *except*

when $k = 1$ and the work-node is at $j = 1$, in which case $R''(1) = R'(1)$. However, in this case, $L''(1) = L'(1)$, because only the in-degree of $j = 1$ appears, which does not get changed. Thus, since $L'(1) \le R'(1)$ ($\overline{\mathbf{D}'}$ is graphical), graphicality cannot be broken at $k = 1$ when $j = 1$. Let us now consider that the work-node is still at position $j = 1$, but $k > 1$. For $1 < k < x$, the in-degrees in $\overline{\mathbf{D}''}$ are the same as those in $\overline{\mathbf{D}'}$, hence $L''(k) = L'(k)$. For $k \ge x$, however, we have $L''(k) = L'(k) - 1$. Now consider $j > 1$. For $1 \le k < x$, we have $L''(k) = L'(k)$ and for $k \ge x$, $L''(k) = L'(k) - 1$. The following summarizes the relationships above:

(A) $j = 1$:
- (A.1) $k = 1$: $L''(1) = L'(1)$, $R''(1) = R'(1)$.
- (A.2) $1 < k < x$: $L''(k) = L'(k)$, $R''(k) = R'(k) - 1$.
- (A.3) $x \le k$: $L''(k) = L'(k) - 1$, $R''(k) = R'(k) - 1$.

(B) $j > 1$:
- (B.1) $1 \le k < x$: $L''(k) = L'(k)$, $R''(k) = R'(k) - 1$.
- (B.2) $x \le k$: $L''(k) = L'(k) - 1$, $R''(k) = R'(k) - 1$.

Since $L'(k) \le R'(k)$ for all k, graphicality for $\overline{\mathbf{D}''}$ can only be broken (that is to have $L'' > R''$ for some k), if $L'(k) = R'(k)$, namely in cases (A.2) and (B.1) above. Observe that $L'(k)$ and $R'(k)$ are computed from $\overline{\mathbf{D}'}$, hence they are independent from ℓ or x. This gives us the following simple procedure for finding the leftmost fail-node, if it exists. Starting from $k = 2$ for $j = 1$, and $k = 1$ for $j > 1$, find the smallest k_0 for which $L'(k_0) = R'(k_0)$. If no such k_0 exists, then there are no fail-nodes and all non-forbidden nodes are to be included in the allowed set. If there is such a k_0, the first red node q' to the right of k_0 ($q' \ge k_0 + 1$) is the leftmost fail-node of $\overline{\mathbf{D}''}$, which when identified in the original bds $\overline{\mathbf{D}}$ will give the leftmost fail-node q. All non-forbidden nodes to the left of q are to be included in the allowed set.

3.2. Summary for finding the allowed set

What we discussed in detail in the previous subsection corresponds to step (3) of the main algorithm described in the beginning of Section 3. Given the normal-ordered bds $\overline{\mathbf{D}}$ at the end of step 2) of the main algorithm:

(3.1) Identify $\mathcal{L}_i$ from the first $\overline{d}_i^{(o)}$ nodes not in $\mathcal{X}_i$.

(3.2) Identify the "red" set $\mathcal{R}_i$ as those nodes that are neither in $\mathcal{L}_i$ nor in $\mathcal{X}_i$. Note, the color label is associated with the node, not its index.

(3.3) Build $\overline{\mathbf{D}'}$ as follows:

$$\overline{d'}_b^{(i)} = \begin{cases} \overline{d}_b^{(i)} - 1 & \text{if } b \in \mathcal{L}_i \setminus \{m\} \\ \overline{d}_b^{(i)} & \text{otherwise} \end{cases}$$

and

$$\overline{d'}_c^{(o)} = \begin{cases} 1 & \text{if } c = i \\ \overline{d}_c^{(o)} & \text{otherwise} \end{cases}$$

where m is the last node in $\mathcal{L}_i$.

(3.4) Shift nodes from $\mathcal{L}_i \setminus \{m\}$ to the right in the sequence (and only these) such as to restore ordering non-increasingly by in-degrees (if needed), preserving the color labels of the nodes in the process. The work-node may have shifted to a new location j after this step. This is the updated sequence $\overline{\mathbf{D}'}$.

(3.5) Starting from $k = 1$ if $j \neq 1$ or from $k = 2$ if $j = 1$, find k_0 as the smallest k such that $L'(k) = R'(k)$, where $L'(k)$ and $R'(k)$ are computed from *the reordered* (after step (3.4)) $\overline{\mathbf{D}'}$ using (4) and (5). If there is no such k_0, then the allowed set $\mathcal{A}_i$ is all the nodes in $\overline{\mathbf{D}}$ except nodes from the forbidden set $\mathcal{X}_i$.

(3.6) Otherwise find the leftmost red node q' in the updated bds $\overline{\mathbf{D}'}$ to the right of k_0, that is with $q' > k_0$. Then the corresponding node q in $\overline{\mathbf{D}}$, will be the leftmost fail node. Note that q' is the new position of the node at q in $\overline{\mathbf{D}}$ after the reordering in (3.4).

(3.7) The allowed set $\mathcal{A}_i$ is formed by all nodes in $\overline{\mathbf{D}}$ not in $\mathcal{X}_i$, and to the left of q.

4. The sampling problem

The algorithm generates an independent sample digraph every time it runs, *without restarts or rejections,* and it guarantees that *every possible* realization of a graphical bds by simple digraphs can be generated with a non-zero probability. However, the algorithm realizes the digraphs with non-uniform probability. Nevertheless, knowing the relative probability for every digraph's occurrence allows us to calculate network observable averages as if they were obtained from a uniform sampling. In particular, the following expression, which is a well-known result in biased sampling [43, 44], provides these averages as:

$$\langle Q \rangle = \frac{\sum_{j=1}^{M} w(\mathbf{s}_j) Q(\mathbf{s}_j)}{\sum_{j=1}^{M} w(\mathbf{s}_j)} \,, \tag{6}$$

where Q is an observable measured from the samples $\mathbf{s}_j$ generated by an algorithm. The $w(\mathbf{s}_j)$ sample weight is the inverse of the relative probability of the occurrence of $\mathbf{s}_j$ and M is the number of the samples generated. In Subsection 4.1 we give a detailed derivation of this formula, specialized to our graph construction problem. The weights of the samples generated by our algorithm are given by

$$w(\mathbf{s}) = \prod_{i} \prod_{j=1}^{d_i^{(o)}} k_i(j) \,. \tag{7}$$

where i runs over all the nodes with non-zero out-degree as they are picked by the algorithm to become work-nodes, and $k_i(j) = |\mathcal{A}_i(j)|$ is the size of the allowed sets $\mathcal{A}_i(j)$ just before connecting the j-th out-stub of i. Note that $w \geq 1$ since there always exists at least one digraph realizing the bds. Subsection 4.2 gives a derivation of (7).

4.1. Biased sampling over classes

Our algorithm sequentially connects all stubs from a series of work nodes and finishes with a simple, labeled digraph. This process can be uniquely described by a *path* of connection sequences. Having chosen a work node i_1 for the first time, it determines the allowed set $\mathcal{A}_{i_1}$. We next choose uniformly at random a node $j_1(i_1) \in \mathcal{A}_{i_1}$ and connect a stub of i_1 to a stub at $j_1(i_1)$. We could have chosen $j_1(i_1)$ following any other criterion, but in that case the expression (7) of the weights would have to be modified accordingly. After this connection we recompute the new allowed set $\mathcal{A}_{j_1}(i_1)$, then connect another stub of i_1, and so on until all the stubs have been used up at i_1. Let us denote by $\mathbf{s}$ such a path of connection sequences:

$$\mathbf{s} = \left\{ i_1, j_1(i_1), \ldots, j_{\bar{d}_{i_1}^{(o)}}(i_1); i_2, j_2(i_2), \ldots, j_{\bar{d}_{i_2}^{(o)}}(i_2) \ldots \right\} \tag{8}$$

where $\bar{d}_i^{(o)}$ denotes the residual out-degree of node i. A path $\mathbf{s}$ uniquely defines the digraph $\mathbf{G}(\mathbf{s})$ created, as the collection of all connections in (8) forms the edge set of the created graph $\mathbf{G}(\mathbf{s})$. However, several paths may lead to the same digraph. Also note that the order of the connections in (8) matters in the calculation of the weight, as the corresponding allowed sets in general depend on history of connections up to that point. For a finite bi-degree sequence the number of distinct samples (paths) is also finite. Let us denote this set of paths by:

$$\Pi = \{\pi_1, \ldots, \pi_P\} \ .$$

Let us now assume that we built with our algorithm a sequence of samples $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_M$, and that the sample number M is large enough for us to see all elements of Π sufficiently many times. Given some path $\mathbf{s}$ we compute a quantity $Q(\mathbf{s})$, and we are interested in calculating the average of Q over path ensembles. In our case Q is defined on the final graph itself $Q(\mathbf{s}) = Q(\mathbf{G}(\mathbf{s}))$, but for now we will not consider that, explicitly. If we were just simply computing the average of Q over the set of samples, we would obtain an average *biased* by the way the algorithm builds the paths from Π:

$$\langle Q \rangle = \frac{1}{M} \sum_{i=1}^{M} Q(\mathbf{s}_i) = \sum_{k=1}^{P} \frac{M_k}{M} Q(\pi_k) \ , \tag{9}$$

where M_k is the number of times we have seen path π_k appear in the sequence of samples. Clearly,

$$\rho_k = \lim_{M \to \infty} \frac{M_k}{M} \tag{10}$$

is the probability by which path π_k is generated via the algorithm. We now assume that we can compute analytically the path probabilities ρ_k, from knowing how the algorithm works. Instead of (9) we want to compute the average as if it was measured over the uniform ensemble of paths, that is:

$$\langle Q \rangle_{up} = \frac{1}{P} \sum_{k=1}^{P} Q(\pi_k) \ . \tag{11}$$

If we form:

$$\begin{aligned} \langle Q \rangle_{bp} &= \frac{\sum_{i=1}^{M} \frac{1}{\rho(\mathbf{s}_i)} Q(\mathbf{s}_i)}{\sum_{i=1}^{M} \frac{1}{\rho(\mathbf{s}_i)}} \\ &= \frac{\sum_{k=1}^{P} \frac{M_k}{M\rho(\pi_k)} Q(\pi_k)}{\sum_{k=1}^{P} \frac{M_k}{M\rho(\pi_k)}} \ , \end{aligned} \tag{12}$$

we have $\lim_{M\to\infty} \langle Q \rangle_{bp} = \langle Q \rangle_{up}$, due to (10). Thus, the weighted average (12) should be used in order to obtain averages according to uniform sampling in the $M \gg 1$ limit.

Let us assume that there is an equivalence relation "$\sim$" between paths, hence inducing a partitioning of Π into K equivalence classes: $\Pi = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_K$, where $\mathcal{C}_\ell = \left\{\pi_{k_1^\ell}, \ldots, \pi_{k_{\mu_\ell}^\ell}\right\}$. The size of class $\mathcal{C}_\ell$ is denoted by $\mu_\ell = |\mathcal{C}_\ell|$. We have $\sum_{\ell=1}^{K} \mu_\ell = P$. Alternatively, for some given path π, we will denote by $\mathcal{C}(\pi)$ the equivalence class of π and by $\mu(\pi) = |\mathcal{C}(\pi)|$ its size. Let us also assume that if $\mathbf{s}, \mathbf{r} \in \mathcal{C}_\ell$, that is $\mathbf{s} \sim \mathbf{r}$, then $Q(\mathbf{s}) = Q(\mathbf{r})$. For example, in our case distinct paths may lead to the same digraph. We introduce the equivalence relation "$\sim$" and say that two paths

$\mathbf{s}$ and $\mathbf{r}$ are equivalent, $\mathbf{s} \sim \mathbf{r}$ if they produce the same labeled digraph, $\mathbf{G}(\mathbf{s}) = \mathbf{G}(\mathbf{r})$. Clearly, if Q depends only on the constructed graph, i.e., $Q(\pi) = Q(\mathbf{G}(\pi))$ for all $\pi \in \Pi$, then $Q(\mathbf{s}) = Q(\mathbf{r})$ whenever $\mathbf{s} \sim \mathbf{r}$.

Our goal is to obtain the average of Q uniformly over the equivalence classes, that is:

$$\langle Q \rangle_{uc} = \frac{1}{K} \sum_{\ell=1}^{K} Q\left(\pi_{k_1^\ell}\right) \ , \tag{13}$$

where we chose to write the first element of $\mathcal{C}_\ell$ in the argument of Q, but of course, any other element could have been chosen from the same class, as Q is constant within a class. In general, (12) will not produce $\langle Q \rangle_{uc}$, but a sum weighted by class sizes. Instead, let us consider:

$$\langle Q \rangle_{bc} = \frac{\sum_{i=1}^{M} \frac{1}{\mu(\mathbf{s}_i)\rho(\mathbf{s}_i)} Q(\mathbf{s}_i)}{\sum_{i=1}^{M} \frac{1}{\mu(\mathbf{s}_i)\rho(\mathbf{s}_i)}} \ . \tag{14}$$

It is then easy to see that:

$$\langle Q \rangle_{bc} = \frac{\sum_{k=1}^{P} \frac{M_k/M}{\mu(\pi_k)\rho(\pi_k)} Q(\pi_k)}{\sum_{k=1}^{P} \frac{M_k/M}{\mu(\pi_k)\rho(\pi_k)}} \overset{M\to\infty}{\longrightarrow} \langle Q \rangle_{uc} \ .$$

In order for (14) to be useful in practice, one has to be able to compute the size of the equivalence class $\mu(\mathbf{s})$ from seeing $\mathbf{s}$ and knowing how the algorithm works. Fortunately this is possible in our case, as shown next.

4.2. Computing the weights

First, let us note that when connecting the out-stubs of a work-node we are not affecting the out-stubs of any other nodes, but only in-stubs. Hence, all nodes with non-zero out-degrees will eventually be picked as work-nodes by the algorithm. Since normal ordering is first by in-degrees, the *order* in which nodes will become work-nodes depends on the sequence of connections. Let us now calculate the probability of the path $\mathbf{s}$ in (8). Given a residual sequence, the work-node i_1 is uniquely determined by the algorithm as described before. Since the next connection is picked uniformly at random, the probability of the link from i_1 to $j_1(i_1) \in \mathcal{A}_{i_1}(j_1)$ is $|\mathcal{A}_{i_1}(j_1)|^{-1}$. Let $k_i(j) = |\mathcal{A}_i(j)|$ denote the number of nodes in $\mathcal{A}_i(j)$. Then, it is easy to see that the probability of a path $\mathbf{s}$ is given by:

$$\rho(\mathbf{s}) = \left[\prod_{k} \prod_{j=1}^{d_{i_k}^{(o)}} k_{i_k}(j) \right]^{-1} \tag{15}$$

where $i_1, i_2, \ldots,$ denote the work-nodes in the order in which they are picked by the algorithm. This expression can be computed readily in a computer as the algorithm progresses. In order for us to use (14) it seems that we would need also to obtain the size $\mu(\mathbf{s})$ of the class to which path $\mathbf{s}$ belongs. Clearly, two different paths $\mathbf{s}$ and $\mathbf{s}'$ will result in the same graph ($\mathbf{s} \sim \mathbf{s}'$) if and only if the sequence of connections in one path is a permutation of the connections in the other path. Hence, the class size $\mu(\mathbf{s})$ is nothing but the number of permutations of the connections, which is the same for all paths, that is, all classes have the same size μ. Since all connections are made from a node first before moving on to another, we have $\mu = \prod_{i=1}^{N} d_i^{(o)}!$. However,

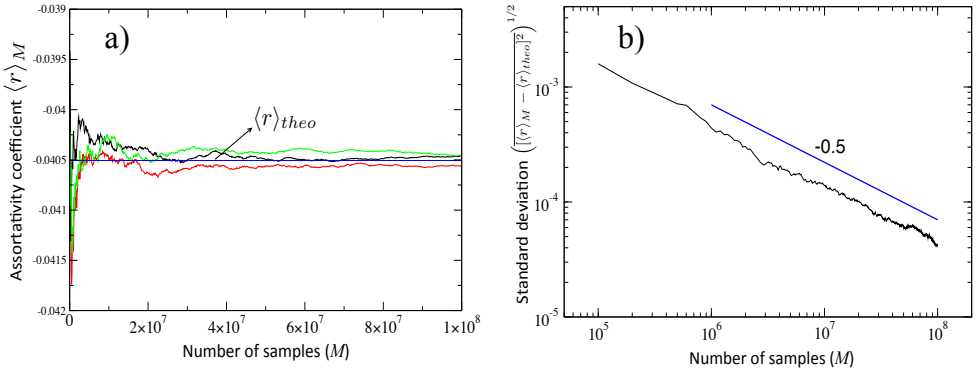

Figure 3. Biased sampling on the example bds $\mathbf{D}_8$. The measure monitored is Newman's assortativity coefficient r [45]. In b) the ensemble average was taken over 50 runs.

we actually don't need to use this number: one can simply multiply by μ both the numerator and the denominator of (14) to obtain (6-7).

4.3. A simple example

In this subsection we illustrate the algorithm on a simple sequence: $\mathbf{D}_8 = \{(2,2),(2,1),(1,3),(1,1),(1,0)\}$. There are 11 distinct labeled digraphs realizing this sequence and there are 2!1!3!1!0! = 12 paths in a class, leading to the same graph. Two paths that lead to different graphs are for example $\mathbf{s}_1 = \{(1,4)(1,2);(3,1),(3,5),(3,2);(2,1);(4,3)\}$ (connect an out-stub of node 1 to an in-stub of node 4, etc.) and $\mathbf{s}_2 = \{(1,2)(1,3);(2,1);(4,1);(3,4),(3,5),(3,2)\}$. For the former, $w(\mathbf{s}_1) = [\rho(\mathbf{s}_1)]^{-1} = 8$ and for the latter it is $w(\mathbf{s}_2) = [\rho(\mathbf{s}_2)]^{-1} = 54$. Let us now consider the Pearson coefficient r of degree-degree correlations, or the assortativity coefficient defined for directed graphs [45] as our network observable $Q = r$. For each one of the 11 graphical realizations of $\mathbf{D}_8, r$ can be calculated exactly, as can the uniform average over this ensemble, obtaining $\langle r\rangle_{theo} = -0.040506$. We will refer to $\langle r\rangle_{theo}$ as the "theoretical value". We then let our algorithm run on this sequence to produce M samples and using (6-7) to obtain the corresponding coefficient $\langle r\rangle_M$. Fig 3a) shows a few runs with different seeds and their convergence to the theoretical value. Fig 3b) shows the standard deviation $(\overline{[\langle r\rangle_M - \langle r\rangle_{theo}]^2})^{1/2}$ where the overline denotes an ensemble average over runs.

5. Complexity of the algorithm

To determine the theoretical upper bound for the complexity of the algorithm, that is the worst-case complexity, notice that there are only three steps in the algorithm that require more than $O(1)$ computational operations, or steps, to complete.

First, after each connection is placed, one must bring the residual sequence into normal order, steps 6) or 7). To accomplish this, both the work-node i and the target node m will have to move to the right, but the relative positions of all other nodes will remain unchanged. In other words, if we were to remove nodes i and m, the rest of the bds would already be sorted. Thus, in order to complete these steps, one

only has to find the new positions of nodes i and m and insert them into the already sorted bds. Therefore, the complexity of either one of step 6) and step 7) is simply $O\left(2\log N + N\right) \approx O\left(N\right)$, where N is the number of the nodes in the sequence being ordered.

Second, the allowed set $\mathcal{A}$ must be built before placing each connection (step 3). Following the summary of this step, given in Subsection 3.2, notice that steps 3.1 to 3.4 can be all finished during a single scan of the residual bds. This is clearly so for the creation of the leftmost set $\mathcal{L}_i$ and for setting the "red" color labels (or flags) (steps (3.1) and (3.4)). Concerning the ordering of the bds $\overline{\mathbf{D}'}$, it is possible to create it already sorted by simply scanning the bds $\overline{\mathbf{D}}$ while keeping track of the in-degree $d^\star$ of the nodes currently being copied and the index a in $\overline{\mathbf{D}'}$ of the first node with that in-degree. Then, because $\overline{\mathbf{D}}$ is in normal order, the only possibility for a node in $\overline{\mathbf{D}'}$ to break the order is if its in-degree equals $d^\star + 1$. In this case, it can be simply swapped with the node at a, because, as argued in Subsection 3.1, the mechanism to build the allowed set is entirely based on the FR theorem, which does not require the bds to be in normal order, but to be simply ordered non-increasingly by its in-degrees. Thus, steps (3.1) to (3.4) can be completed in $\mathcal{O}\left(N\right)$ steps.

Third, the computation of the sums L', R' and their comparison must be conducted, which is the same step as (3) in an FR test. To determine the complexity of an FR test note that computing the repeated sums for each one of the inequalities (3) is quite inefficient. Instead, below we derive recurrence relations that allow us to complete the FR test in a linear, $\mathcal{O}\left(N\right)$ number of steps.

The steps of the main algorithm are done sequentially, and thus can all be completed in a total of $\mathcal{O}\left(N\right)$ steps. They must, however, be repeated for each edge in the digraph. Thus, the maximum complexity of the algorithm is $\mathcal{O}\left(NM\right)$ where $M = \sum_i d_i^{(o)}$ is the number of edges. Since $\mathcal{O}\left(M\right) \leq \mathcal{O}\left(N^2\right)$ the maximum complexity of the algorithm is $O(N^3)$. It is important to note though, that for a given bds the complexity of the algorithm can be substantially smaller, similar to the case for our undirected graph sampling algorithm [11].

5.1. The Fulkerson-Ryser test revisited

The most complex part of the Fulkerson-Ryser test is to compute the lhs and the rhs of inequalities (3), which we rewrite here for the sake of readability:

$$L(k) = \sum_{s=1}^{k} d_s^{(i)} \,,$$

$$R(k) = \sum_{s=1}^{k} \min\left\{k-1, d_s^{(o)}\right\} + \sum_{s=k+1}^{N} \min\left\{k, d_s^{(o)}\right\} .$$

Our goal is to find recursion relations for $L(k)$ and $R(k)$. For the lhs the relation is simply

$$L(k+1) = L(k) + d_k^{(i)} \,,$$

with $L(1) = d_1^{(i)}$.

For the rhs, first note that one can write it as

$$R(k) = -k + \sum_{i=1}^{N} \min\left\{k, g_i^{(o)}(k)\right\} \,, \tag{16}$$

where $g_i^{(o)}(k)$ is the family of integer sequences defined as

$$g_i^{(o)}(k) = \begin{cases} d_i^{(o)} + 1 & \forall i \leqslant k \\ d_i^{(o)} & \forall i > k \end{cases} .$$

Now, let us introduce $G_k(p) = \sum_{i=1}^{N} \delta_{p,g_i^{(o)}(k)}$, that is, the number of indices i for which $g_i^{(o)}(k) = p$. Then, from (16) follows that

$$R(k) = -k + \sum_{p=1}^{k} pG_k(p) + k \sum_{p=k+1}^{N} G_k(p) , \tag{17}$$

hence

$$R(1) = N - 1 - G_1(0) , \tag{18}$$

where we used the fact that $\sum_{p=0}^{N} G_k(p) = N$.

Furthermore, let us introduce the following notations:

$$\Delta G_k(p) \equiv G_k(p) - G_{k-1}(p)$$
$$\tilde{G}_k(q) \equiv \sum_{i=0}^{q} G_k(i) .$$

Then, after some simple manipulations, from (17) it follows that

$$R(k) - R(k-1) = N - 1 - \tilde{G}_{k-1}(k-1) + \sum_{p=1}^{k-1} p\Delta G_k(p) + k \sum_{p=k}^{N} \Delta G_k(p) . \tag{19}$$

Finally, notice that $\Delta G_k(p) = \delta_{p,d_k^{(o)}+1} - \delta_{p,d_k^{(o)}}$. Substituting it into (19), we obtain:

$$R(k) = \begin{cases} R(k-1) + N - \tilde{G}_{k-1}(k-1) & \forall d_k^{(o)} < k \\ R(k-1) + N - \tilde{G}_{k-1}(k-1) - 1 & \forall d_k^{(o)} \geqslant k \end{cases} \tag{20}$$

Thus, we have turned the problem of finding a recursion relation for $R(k)$ into the problem of finding $\tilde{G}_k(k)$. To solve this, first note that

$$\tilde{G}_k(k) = \tilde{G}_{k-1}(k-1) + G_{k-1}(k) - \delta_{k,d_k^{(o)}} ,$$

with $\tilde{G}_1(1) = G_1(0) + G_1(1)$. The above equation constitutes a recursion relation for $\tilde{G}_k(q)$. Such a relation can be rewritten as

$$\tilde{G}_k(k) = \tilde{G}_{k-1}(k-1) + G_1(k) + S(k) ,$$

where

$$S(k) = \sum_{t=2}^{k-1} \delta_{k,d_t^{(o)}+1} - \sum_{t=2}^{k} \delta_{k,d_t^{(o)}} .$$

Observe that $S(k)$ and $G_1(k)$ can be easily computed while scanning the bds, and then calculating $L(k)$ and $R(k)$ for each k requires a single operation. Thus, the entire FR test can be completed in $O(N)$ steps.

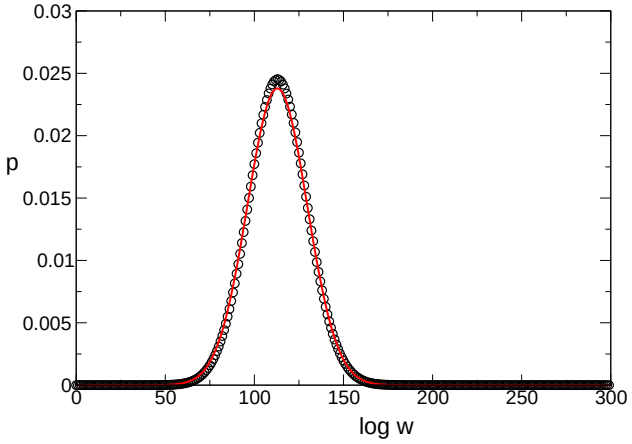

Figure 4. Probability distribution p of the logarithm of weights for an ensemble of bi-degree sequences on $N = 100$ nodes. The in-degrees were drawn from a normalized power-law distribution $\sim d_{in}^{-\gamma}$ with $\gamma = 3$ and the out-degrees were drawn from a Poisson distribution $e^{-\lambda}\lambda^{d_{out}}/d_{out}!$, with the same average as the average *in-degree*, $\lambda = \langle d_{in} \rangle$. The black circles are the simulation data and the red continuous line is a Gaussian fit.

6. Discussion

In summary, we have developed a graph construction and sampling algorithm to construct simple directed graphs realizing a given sequence of in- and out-degrees. Such constructions are needed in practical modeling situations, ranging from epidemics and sociology through food-webs to transcriptional regulatory networks, where we are interested in learning about the statistical properties of the network observables as determined *only by the bi-degree sequence* and nothing else.

Unlike existing algorithms such as the Configuration Model, which is affected by uncontrolled biases and unacceptably long running times except for a very restricted class of sequences, our algorithm is rejection-free. Also, it guarantees the independence of the produced samples, unlike MCMC methods, which have unknown mixing times. While its mathematical underpinnings are nontrivial, the algorithm itself is straightforward to implement. In principle, our approach can be extended to include more complex constraints, such as a given sequence of motifs frequencies, but we have only concentrated on degree sequences since they are, arguably, the most fundamental of constraints. The algorithm can also be used to sample from given in- and out-degree distributions, not just sequences: given such distributions, one first samples a graphical bds from these, then one applies our algorithm to generate digraphs. In this case, however, the sample weights (7) must be modified to reflect the probability of the occurrence of the given graphical bds when drawn from the distributions.

Just as in the case of undirected graphs, we can expect the distributions of the weights for large graphs to be log-normal, as shown in Ref. [11]. As an example, figure 4 shows the distribution for bi-degree sequences in which the in-degrees follow a power law with exponent $\gamma = 3$ and the out-degrees a Poisson distribution whose

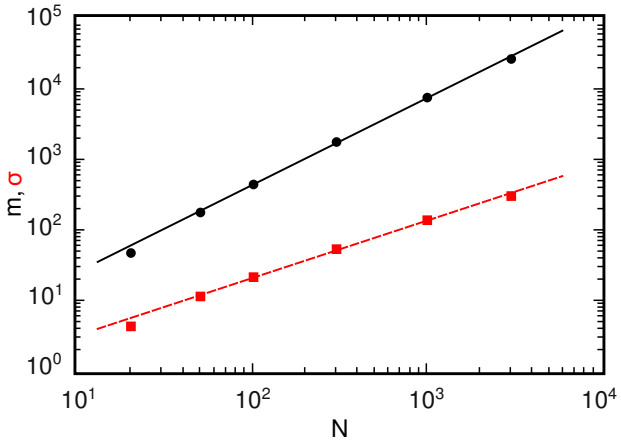

Figure 5. Mean m (black circles) and standard deviation σ (red squares) of the distributions of the logarithm of the weights vs. number of nodes N of samples. In-degrees and out-degrees are both drawn from a power-law distribution $P(d) \sim d^{-\gamma}$, with $\gamma = 3$. The solid black line and the dashed red line are data fit results, showing that m and σ follow power-law scaling laws $m \sim N^{\alpha}$ and $\sigma \sim N^{\beta}$. The values of the exponents, given by the slopes of the lines are $\alpha = 1.23 \pm 0.02$ and $\beta = 0.81 \pm 0.02$.

mean matches the average in-degree. Indeed, the distribution of the weight logarithms is well approximated by a Gaussian. Similarly the undirected case, we find for all the examples we studied numerically, that the standard deviation σ of the distributions of weight logarithms grows slower than the mean m with the number of nodes N; see figure 5 showing the scaling of m and σ for bi-degree sequences in which both in-degrees and out-degrees follow a power law distribution with exponent $\gamma = 3$. Thus, we may expect that typically, in the $N \to \infty$ limit, the rescaled weight distribution becomes a delta function, making the sampling asymptotically uniform.

Bounds on the complexity of the algorithm could easily be obtained by inspecting the algorithm, showing a maximum complexity on the order of $\mathcal{O}(NM)$ where M is the total number of edges, $M = \sum_{i=1}^{N} \bar{d}_i^{(o)}$.

In developing our results, we also provided an efficient way of implementing the Fulkerson-Ryser test, whose scope of application goes beyond our present algorithm, as it can be used in any context to determine whether a bi-degree sequence is graphical.

Acknowledgments

HK was supported in part by the US National Science Foundation (NSF) through grant DMR-1005417 and KEB by the NSF grant DMR-0908286. ZT and HK were supported in part by the NSF BCS-0826958, HDTRA 201473-35045 and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any

copyright notation here on.

References

[1] Newman M E J 2010 *Networks: an introduction* (Oxford: Oxford University Press)
[2] Easley D and Kleinberg J 2010 *Networks, crowds, and markets: reasoning about a highly connected world* (Cambridge: Cambridge University Press)
[3] Barrat A, Barthélemy M and Vespignani A 2008 *Dynamical processes on complex networks* (Cambridge: Cambridge University Press)
[4] Newman M E J, Barabási A L and Watts D J 2006 *The structure and dynamics of networks (Princeton Studies in Complexity* (Princeton, NJ: Princeton University Press)
[5] Boccaletti S, Latora V, Moreno Y, Chavez M and Hwang D-U 2006 *Phys. Rep.* **424** 175
[6] Ben-Naim E, Frauenfelder F and Toroczkai Z 2004 *Complex networks (Lecture Notes in Physics)* (Berlin: Springer Verlag)
[7] Dorogovtsev S N and Mendes J F F 2002 *Evolution of networks: from biological nets to the internet and www* (Oxford: Oxford University Press)
[8] Bender E A and Canfield E R 1978 *J. Comb. Th. A* **24** 296
[9] Koren M 1976 *J. Comb. Theor. B* **21** 235
[10] Kim H, Toroczkai Z, Erdős P L, Miklós I and Székely L A 2009 *J. Phys. A: Math. Theor.* **42** 392001
[11] Del Genio C I, Kim H, Toroczkai Z and Bassler K E 2010 *PLoS ONE* **5** (4) e10012
[12] Bollobás B 1980 *Eur. J. Comb.* **1** 311
[13] Taylor R 1982 *SIAM J. Alg. Disc. Meth.* **3** 114
[14] Molloy M and Reed B 1995 *Rand. Struct. Alg.* **6** 161
[15] Rao A R, Jana R and Bandyopadhya S 1996 *Indian J. of Statistics* **58** 225
[16] Kannan R, Tetali P and Vempala S 1999 *Random Struct. Alg.* **14** 293
[17] Newman M E J, Strogatz S H and Watts D J 2001 *Phys. Rev. E* **64** 026118
[18] Chung F and Lu L 2002 *Ann. Combinatorics* **6** 125
[19] Maslov S and Sneppen K 2002 *Science* **296** 910
[20] Milo R, Shen-Orr S, Itzkovitz S, Kashtan N and Chklovskii D 2002 *Science* **298** 824
[21] Morelli L G 2003 *Phys Rev E* **67** 066107
[22] Itzkovitz S, Milo R, Kashtan N, Ziv G and Alon U 2003 *Phys. Rev. E* **68** 026127
[23] Milo R, Kashtan N, Itzkovitz S, Newman M E J and Alon U 2003 `arXiv:cond-mat/0312028v2`
[24] Park J and Newman M E J 2003 *Phys. Rev. E* **68** 026112
[25] Viger F and Latapy M 2005 *Lect. Notes Comp. Sci.* **3595** 440–9
[26] Britton T, Deijfen M and Martin-Löf A 2006 *J. Stat. Phys.* **124** 1377–97
[27] Cooper C, Dyer M and Greenhill C 2007 *Comb. Prob. Comp.* **16** 557–93
[28] Bianconi G, Coolen A C C and Perez Vicente C J 2008 *Phys. Rev. E* **78** 016114
[29] Bianconi G 2009 *Phys Rev E* **79** 036114
[30] Erdős P L, Miklós I and Toroczkai Z 2010 *Elec. J. Comb.* **17** R66
[31] Boguñá M, Pastor-Satorras R and Vespignani A 2004 *Eur. Phys. J. B* **38** 205
[32] Catanzaro M, Boguñá M and Pastor-Satorras R 2005 *Phys. Rev. E* **71** 027103
[33] Ángeles Serrano M and Boguñá M 2005 *Proc. CNET2004 Am. Inst. Phys. Conf.*
[34] Blitzstein J and Diaconis P 2011 *Internet Mathematics* **6** (4) 489
[35] Hartmann A K 1999 *Practical guide to computer simulations* (World Scientific)
[36] Wasserman S and Faust K 1994 *Social network analysis: methods and applications* (Cambridge: Cambridge University Press)
[37] Chartrand G and Lesniak L 1986 *Graphs & Digraphs* (2nd edition, Wadsworth, Inc.)
[38] Fulkerson D R 1960 *Pacific J. Math.* **10** (3) 831
[39] H. Ryser. *Combinatorial mathematics.* Carus Mathematical Monographs, MAA, 1963.
[40] Havel V 1955 *Časopis Pěst. Mat.* **80** 477 (in Czech)
[41] Hakimi S L 1962 *J. SIAM Appl. Math.* **10** 496
[42] Kleitman D J and Wang D L 1973 *Discrete Math.* **6** 79
[43] Cochran W G 1977 *Sampling techniques* 3^{rd} *edition* (Wiley)
[44] Newman M E J and Barkema G T 1999 *Monte-Carlo methods in statistical physics* (Oxford: Oxford University Press)
[45] Newman M E J 2003 *Phys. Rev. E* **67** 026126